Detecting Online Test Cheating Through User Behavior Monitoring

BY

OOI KHAI SHEN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JUNE 2025

COPYRIGHT STATEMENT

This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to express thanks and appreciation to my supervisor, Ts Dr Mogana a/p Vadiveloo who have given me a golden opportunity to involve in the Computer Vision study. Besides that, she has given me a lot of guidance in order to complete this project. When I was facing problems in this project, the advice from them always assists me in overcoming the problems. Again, a million thanks to my supervisor and moderator.

Other than that, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

ABSTRACT

The shift to online testing is becoming a significant trend in the modern learning, yet this transition presents serious challenges to academic integrity and credibility of institutional qualifications. In a traditional test environment, physical supervision like physical examination setup in grand hall and attendance is a must, effectively detect cheating, but such monitoring is not feasible for remote exams, creating opportunities for students to engage in dishonest behaviour. The proposed system, "EyeGuard," aims to assist and solve this issue by employing a computer vision-based eye gaze detection system that uses a student webcam to track their eye movements during an online test. By analysing eye gaze patterns and browser incident to identify inappropriate activity, such as looking at unauthorized materials during the test and switching the tab, the system can detect potential cheating in real-time, allowing any suspicious behaviour to be reported instantly for necessary investigation. The principal objective of this project is to provide a reliable and effective solution for monitoring online exams that reduces the reliance on human supervisors, which can be costly and impractical at scale. Through the automated detection of suspicious behaviour, "EyeGuard" fosters a more confident and fairer environment for online test, ensuring the integrity of the examination process while offering a scalable, and low-cost solution for educational institutions.

Area of Study: Computer Vision, Chrome Extension Development

Keywords: Eye Gaze Tracking, Online Test Monitoring, Cheating Detection, Real-Time Monitoring and Alerting, AWS Integration

TABLE OF CONTENTS

TITLE PAGE i

COPYRIGHT STATEMENT ii

ACKNOWLEDGEMENTS iii

ABSTRACT iv

TABLE OF CONTENTS v

LIST OF FIGURES ix

LIST OF TABLES xii

LIST OF ABBREVIATIONS xiii

CHAPTER 1 INTRODUCTION 1

1.1 Problem Statement and Motivation 1

1.2 Objectives 3

1.3 Project Scope and Direction 4

1.4 Contributions 5

1.5 Report Organization 6

CHAPTER 2 LITERATURE REVIEW 7

2.1 Introduction 7

2.2 Real-Time Object Detection in Online Exam Proctoring 7

2.3 Facial Landmark Detection 10

2.3.1 Google MediaPipe Face Mesh 11

2.3.2 WebGazer 12

2.4 Head Pose Estimation 14

2.4.1 Tools for Head Posture Estimation 16

2.5 Non-Computer Vision Approaches for Exam Proctoring 16

2.5.1 Browser and Keystroke Monitoring 16

2.5.2 Audio Analysis 17

2.6 Discussion on Selected Proctoring Techniques and Tools 17

2.7 Existing Methods for Detecting Online Cheating 18

2.7.1 ProctorU 19

2.7.2 Respondus 20

2.7.3 Proctorio 23
2.8 Critical Analysis of Existing System 25
2.8.1 ProctorU 26
2.8.2 Respondus 27
2.8.3 Proctorio 29
2.9 Proposed Solution 30

CHAPTER 3 SYSTEM DESIGN 31
3.1 System Architecture 31
3.2 Use Case Diagram 34
3.2.1 Use Case Descriptions 35
3.3 Activity Diagram 41

CHAPTER 4 SYSTEM METHODOLOGY/APPROACH 44
4.1 Agile Development Methodology 44
4.2 System Requirement 47
4.2.1 Hardware Requirements 47
4.2.2 Software Requirements 48
4.2.2.1 Development Platform and Tools 48
4.3 Timeline 50
4.3.1 Timeline of FYP1 50
4.3.2 Timeline of FYP2 51
4.4 Core Algorithms and Detection Logic 52
4.3.1 The Gaze Boundary Polygon Algorithm 52
4.3.1.1 Comparison with Object Detection (YOLO) 53
4.3.2 The Temporal Filtering Algorithm 54
4.3.3 The Integrity Scoring and Risk Assessment Algorithm 55

CHAPTER 5 SYSTEM IMPLEMENTATION 56
5.1 Backend Setup (Flask Server) 56
5.1.1 Environment File 56
5.1.2 Dependency Management 57
5.1.3 Backend Execution 57

5.2 Frontend Implementation 58
5.2.1 Loading the Extension for Development 58
5.3 Backend Implementation (Python Flask Server) 59
5.3.1 User Authentication Module 59
5.3.2 Session and Event Processing Modules 60
5.3.3 Analysis and Reporting Module 63
5.4 Frontend Implementation (Chrome Extension) 64
5.4.1 Developing Core Extension Architecture and Control 64
5.4.1.1 manifest.json 64
5.4.1.2 background.js 65
5.4.2 Developing the Student Authentication Interface 66
5.4.3 Creating the Pre-Proctoring Guideline Module 67
5.4.4 Constructing the Main Extension Control Panel 70
5.4.5 Developing the Sandboxed Gaze Tracking Module 70
5.4.6 Developing the Post-Session Visualization Report 71
5.4.7 Implementing Real-time Violation Alerts 74
5.4.8 Integration with Cloud Services (AWS) 76
5.4.8.1 Trigger Email Alerts (AWS SES) 76
5.4.8.2 PDF Report Uploads (AWS S3) 77

CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION 79
6.1 Functional Verification 79
6.2 Technology Justification and Accuracy 80
6.3 Performance and Effectiveness Analysis 80
6.3.1 Validation of Gaze Violation Logic 81
6.3.2 Gaze Violation Test Cases Discussion 82
6.3.3 Analysis of Client-Side and System-Level Resource Impact 85
6.4 Comparative Analysis against FYP1 88
6.4.1 Remark on Test Findings 89

CHAPTER 7 CONCLUSION AND RECOMMENDATION **90**

7.1 Conclusion and Novelty 90

7.2 Recommendations 90

REFERENCES **92**

APPENDIX **A-1**

POSTER **A-3**

LIST OF FIGURES

Figure Number	Title	Page
Figure 2.1	Type of Object Detection	8
Figure 2.2	Overview of One-Stage Object Detection	8
Figure 2.3	Overview of Two-Stage Object Detection	9
Figure 2.4	Demonstration of facial landmark detection	11
Figure 2.5	Official logo for WebGazer.js	12
Figure 2.6	Parameter for HPE (Pitch, Yaw, and Roll) and Their Corresponding Direction	14
Figure 2.7	Landmark-based Head Pose Estimation	15
Figure 2.8	The Steps to Train a Landmark-Free Methods for HPE	16
Figure 2.9	ProctorU Test-Taker Exam Session Interface	20
Figure 2.10	List of Respondus LMS Partners	21
Figure 2.11	Lockdown Browser Installation of Respondus	21
Figure 2.12	Pre-examination Check of Respondus	22
Figure 2.13	Proctorio Allow Limited Configuration to the User	24
Figure 2.14	Examination Result of Proctorio	24
Figure 3.1	System Architecture of the EyeGuard Proctoring System.	32
Figure 3.2	Use Case Diagram	34
Figure 3.3	Activity Diagram	41
Figure 4.1	Agile Methodology in System Development	45
Figure 4.2	Agile Development Lifecycle with Iterative Nature	46
Figure 4.3	The “SmartGazeMonitor” Temporal Window Illustration	54
Figure 5.1	env. File	56
Figure 5.2	requirement.txt File	57
Figure 5.3	Running a Backend with Command “python main_app.py”	57
Figure 5.4	Files are Loaded to Chrome Extension through Developer Mode	58
Figure 5.5	Mock Student Info that Stored in Backend	59
Figure 5.6	Login Module	60
Figure 5.7	start_test() Session, Triggered when User Start the Test	61

Figure 5.8	end_test session, Triggered when User End the Test	62
Figure 5.9	get_report Session, Send Final Report Data as JSON to Frontend	62
Figure 5.10	/submit_data() Session, Work as Listener, Receiving JSON Data from the Frontend	63
Figure 5.11	/submit_data, Showing How the System Analyzes Event and Send to Frontend	64
Figure 5.12	manifest.json	65
Figure 5.13	background.js	66
Figure 5.14	Student Authentication Interface.	67
Figure 5.15	Examination Guideline Agreement Module.	68
Figure 5.16	Extension Control Panel in Various States (Idle, Ready, Running).	70
Figure 5.17	The Gaze Tracking Calibration Interface in Action.	71
Figure 5.18	The Final Proctoring Analysis Report Dashboard	72
Figure 5.19	The Alert triggered when User Caught Off-screen Glace	74
Figure 5.20	The Alert triggered when User Caught Minimize Screen	74
Figure 5.21	The Alert triggered when User Caught Switch Between the Tab	75
Figure 5.22	The Critical Alert triggered when User Multiple Violations	75
Figure 5.23	Boto3 as the Coordinator for AWS Services.	76
Figure 5.24	Example Critical Violation Email Alert Sent via AWS SES.	77
Figure 5.25	upload_report_pdf(), to Upload the PDF Report to S3 Bucket	78
Figure 5.26	Administrator able to View the Reports Stored inside S3 Bucket	78
Figure 6.1	Gaze Pattern when User Focused on the Screen	83
Figure 6.2	Gaze Pattern when User Focused Off-screen	83
Figure 6.3	Example System Log when User Focused Entirely on the Screen	84
Figure 6.4	Example System Log when User Focused Entirely on the Screen	84
Figure 6.5	Example System Log when User Make a Random Glance	85

Figure 6.6	Resources Used by the Chrome Before Run the Monitoring	86
Figure 6.7	Resources Used by the Chrome After Run the Monitoring	86
Figure 6.8	Resources Used by the System before Run the Monitoring	87
Figure 6.8	Resources Used by the System After Run the Monitoring	87

LIST OF TABLES

Table Number	Title	Page
Table 2.1	Table 2.1 Advantages and Disadvantages of ProctorU	27
Table 2.2	Advantages and Disadvantages of Respondus	29
Table 2.3	Advantages and Disadvantages of Proctorio	30
Table 3.1	Login to System Use Case Description	35
Table 3.2	Perform gaze calibration description	36
Table 3.3	Start Proctoring Session Use Case Description	37
Table 3.4	End Proctoring Session Use Case Description	38
Table 3.5	View session report use case description	38
Table 3.6	Display on-screen warning use case description	39
Table 3.7	Handle Critical Violation Warning Use Case Description	40
Table 4.1	Development and Testing Environment	47
Table 4.2	End-User Minimum Requirements	47
Table 4.3	Software Components and Tools	48
Table 4.4	Timeline of FYP1	50
Table 4.5	Timeline of FYP2	51
Table 4.6	Comparison of Regression Approach and Classification Approach	53
Table 4.7	Integrity Scoring for each Event	55
Table 4.8	Integrity Scoring with Corresponding Risk Level	55
Table 6.1	Test Cases for User Onboarding and Setup	79
Table 6.2	Test Cases for Core Proctoring and Cloud Integration	79
Table 6.3	Gaze Violation Logic Test Results	81
Table 6.4	Scenario-Based Effectiveness Testing	89

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
AR	Augmented Reality
AWS	Amazon Web Services
CNN	Convolutional Neural Network
CSP	Content Security Policy
FYP	Final Year Project
HPE	Head Pose Estimation
HTML	HyperText Markup Language
HTTPS	HyperText Transfer Protocol Secure
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
LMS	Learning Management System
ML	Machine Learning
PDF	Portable Document Format
S3	Simple Storage Service (AWS)
SDK	Software Development Kit
SES	Simple Email Service (AWS)
SNS	Simple Notification Service (AWS)
SOP	Standard Operating Procedure
UI	User Interface
URL	Uniform Resource Locator
VAD	Voice Activity Detection
YOLO	You Only Look Once

CHAPTER 1 INTRODUCTION

This chapter presents the background and motivation for conducting this project, outlines all the primary objectives and scope, details the key contributions, and provides a general organization for the project itself.

1.1 Problem Statement and Motivation

The transition from traditional physical exams to online exams is one significant trend in the future. This is due to online tests offering lots of benefits compared to the traditional physical test such as convenience and flexibility. In addition, online tests also play significant roles for the universities to explore the possibility of remote courses which to attract the students neither locally nor internationally in the future, expanding institutions reach and opening new learning opportunities in this new era. However, one of the biggest challenges with online tests is ensuring academic integrity in a remote, unsupervised environment. Under physical testing conditions like direct supervision in a physical testing hall, monitoring is much easier and significantly prevents cheating happening. However, in virtual online environment, students have increased opportunities to engage in dishonest behaviors such as consulting unauthorized materials or just simply researching the topic online which occurs in blind spots outside the webcam's view. [1]

These actions then will directly affect the institution’s reputation and the value of the qualifications and certifications. Current proctoring solutions while helpful often have limitations. Relying on the basic webcams monitoring or making the exam questions more complex often fail to effectively detect and prevent cheating behavior because this still requires significant manual monitoring by human, a solution that is not really scalable or consistently effective for large numbers of test-takers.

During the online examination, students take exams using their laptops, and this is the only way that school administrators are able to monitor their behavior, confirming that they are only looking at the screen and didn’t perform any suspicious activity that may be cheating. However, direct webcam monitoring is not scalable and effective solution, and school administrators may overlook some cases of online cheating because human make mistakes is normal. Lockdown browser may be implemented to prevent online cheating as well, preventing users access unauthorized materials online during test but cheating still cannot be completely prevented. This is due to these methods failing to detect critical blind spots that are outside the

screen like students may refer to the additional resource but are not captured by their webcam. Without better solutions, these will affect the integrity of online assessments.

Therefore, to solve the problem stated is to develop a system that can automatically and accurately monitor online exams to detect dishonest behaviors, ensure the exam fairness, and ensure a confidential testing environment. This system could reduce human reliance and be scalable. Manpower could be reduced to only monitor the overall process and take action when alert raised by the system. There are quite a number of applications that have been developed and published online that serve the same objective, aim to develop a better, and fairer test taking environment at the same time ensure the test integrity. These applications are listed in ProctorU, Respondus, and Proctorio. [6][8][9]

However, there are still quite a few limitations with these applications, which shall be discuss in depth in later chapters. The biggest limitation comes in the case that most organizations do not really prefer to use third-party proctoring services due to data confidentiality concerns. Since ProctorU requires access to questions during exams and responses from students during monitoring, those examination questions and students answer which are considered as confidential data will be exposed to third party as well. Even though this can be solved by liability under strict privacy policies but still this will be a risk that the institutions should consider for. As such, while tests like ProctorU [6] are a great platform to ensure exam integrity protection, the dependency on third-party data processing makes them impossible for institutions where in-house management is a concern for tests.

The motivation behind this project is due to the growing need for scalable, real-time examination monitoring solutions that ensure the integrity of online assessments in this digital era. Online testing is not only cost-effective but also provides flexibility for both students and institutions. However, the current solutions face challenges to ensure a fair testing environment. Many current proctoring solutions are too relying on human supervision. Cheating affects those honest students' achievements, and institutions trust in the educational system, and may cause long-term effects for individuals and society. A robust and reliable proctoring system that integrates with advanced technologies listed in computer vision and Artificial Intelligence (AI) can significantly enhance the fairness of online examination, reduce or stop cheating, and support the growing trend of digital learning.

Besides, students taking online exams are often unaware when their actions may be directly or indirectly bringing negative effect to the institutions and eventually the whole

society. This lack of awareness can affect the integrity of the examination process, leading to unfair situations and affect the institution reputation towards academic assessments, which in return can affect the educational standards and trust in society.

1.2 Objectives

The primary objective of this project is to develop an system for detecting cheating during online tests through user behavior monitoring. The system will leverage computer vision techniques, particularly eye gaze tracking to identify potential cheating behaviors. Key objectives include:

1. **To develop a browser extension-based real-time eye-gaze tracking system**

The primary objective is to implement a system that uses computer vision directly within the user browser to monitor eye movements. The system is designed to track eye gaze movement for identifying suspicious activity during online examination. This involves:

- A user-specific calibration process where the student works to define their screen boundaries before the exam begins.
- The gaze tracking is developed in a secure, sandboxed environment to continuously track gaze coordinates.
- Developing a back-end algorithm that analyzes the stream of gaze data to detect suspicious off-screen glances, differentiating them from brief, natural eye movements to minimizing false positives.

2. **To monitor and flag suspicious browser-level user interactions**

Beyond eye-gaze, the system will actively monitor the user browser environment for actions that indicate potential cheating. This includes automatically detecting and logging events such as:

- Switching tabs away from the examination page.
- Changing focus to a different application or window.
- Opening new browser windows during the test session.

3. **To implement a real-time alerting system**

When a violation or suspicious activity is detected, the system will trigger immediate alerts through two distinct channels:

- For the Student: An on-screen alert will be displayed directly on the student exam page, serving as an immediate warning and ask them to keep focus, they are under monitoring.

- For the Administrator: An integration with Amazon Web Service (AWS) Simple Notification Service (SNS) will send an instant notification to the exam administrator, informing them of the suspicious activity so they can take necessary action.

4. **To design and build the system as a client-server architecture with comprehensive reporting**

The system will be architected as a robust client-server application, not a simple website. This involves:

- Developing a Chrome Extension as the client-side component responsible for capturing data which are gaze and browser events and displaying UI to user
- Building a Python Flask back-end server to handle session management, process incoming data, execute the violation detection logic, and trigger alerts.
- Generating a detailed post-exam analysis report that provides administrators with a comprehensive overview of the session, including an "Integrity Score," a visual timeline of all flagged events, and a 2D scatter plot of the student's gaze patterns.

1.3 Project Scope and Direction

The direction of proposed project, "EyeGuard," is to create a dependable system that can help universities manage online tests, whether for midterms or final exams. The reality of online education is that these exams can involve hundreds, or even thousands, of students at once. Having enough supervisors to watch everyone in real-time is simply not practical; it would require a huge amount of manpower, and even then, it's difficult for a person to effectively monitor dozens of video feeds without missing things, especially it is to monitor the eye gaze or head posture, trying to look for suspicious activity.

A key challenge to acknowledge is that even a human proctor finds it hard to tell if a student is actually cheating from a webcam feed alone. It's impossible to see what's happening in the physical blind spots without concrete evidence. Instead, its role is to be a system that monitors and logs suspicious activities. It acts as a tracking layer for human proctors, flagging potential issues so they can focus their attention on the moments that truly require review. This approach keeps a human in control of the final decision, which is crucial because test results can change a student's life and shouldn't be decided by a machine alone.

To accomplish this, this project is scoped on two of the most direct indicators of a student's attention: eye-gaze tracking and browser activity. The system chose to focus on eye-gaze because a student's eyes will move if they are reading off-screen notes, even if their head

stays perfectly still, making it a more reliable clue than head posture. At the same time, the proposed system track browser events like switching tabs, which can be a clear sign of digital cheating. The system is built as a Chrome Extension rather than a separate website or desktop app. The big advantage here is that it can work directly with a university's existing online test portal, adding a layer of security without forcing them to switch to a whole new system.

In practice, when the system logs a suspicious activity, it sends a real-time warning to the student's screen and also notifies the administrator via AWS SNS, in the end of the test, a report generated and shared with the administrator for investigation work if any. To keep the project scope focused and achievable, complex features like microphone audio monitoring or physical object detection are not included. This project aims to prove that this lightweight, browser-based approach is an effective way to detect cheating and provide valuable evidence for review. It establishes a solid foundation for future work, where these other monitoring features could be added to create an even more robust proctoring tool.

1.4 Contributions

This system's primary contribution is to build an eye gaze tracking system to ensure online exam integrity. It is designed to monitor students for suspicious activities in real-time, providing institutions with the evidence needed to evaluate test sessions effectively.

Besides, the project is also flexible and scalable because it able to observe many students at the same time and using algorithms that are trusted, it can observe all the activities that are expected to be cheating without a significant work of test monitoring. This project enhances the possibility of conducting massive online examinations in large institutions like universities just like how the traditional physical examination is conducted. Institutions also offer one new possibility to modify the traditional physical examination to online examination in the future without compromising the integrity of the test and global acceptance of the institution's certificates. Furthermore, this project also reveals the new possibility of the future remote education of the institutions. By developing a solution that addresses the limitations of current online test monitoring systems, this project will enhance the fairness, confidentiality, and integrity of online exams.

1.5 Report Organization

This report details the project's development across seven chapters. Chapter 1 introduces the project, outlining the problem statement, objectives, and scope. Chapter 2 presents a literature review of existing online proctoring systems and the core computer vision technologies that inform the project's design. Chapter 3 outlines the system methodology, detailing the Agile development approach and project timeline. Chapter 4 covers the system design, presenting the overall architecture and key diagrams that illustrate the system's workflow. Chapter 5 describes the implementation, including the setup of the frontend and backend components. Chapter 6 provides a detailed evaluation of the completed system, featuring functional test cases and performance analysis. Finally, Chapter 7 concludes the report by summarizing the project achievements and offering recommendations for future work.

CHAPTER 2 LITERATURE REVIEW

2.1 Introduction

After the era of Covid-19, it is obvious that the world has come into the digital era from examples like the digital payment, e-commerce adoption, remote work and online learning and so on has grown rapidly in the post pandemic world [4] Since then the school should be the same as well. It is necessary to conduct an online examination which is fairly, transparent and confidential. While the previous traditional method will work good for the traditional physical examination with good monitoring and incident control such as prevent unauthorized item brought into the examination hall and accompany the students to the toilet to monitor their action and so on, However, all these actions are not feasible for an online examination because of geographical distance and too much blind spot out of the camera's screen, thus an automated proctoring system powered by AI is essentially important nowadays for online test.

Research has already proposed some commercial applications that are used to monitor the students' behaviors during the online test, like proctorU, and Proctorio. [6][7] To ensure that there is a confidential test environment among the students and organizations. Those applications necessitated the development of robust online proctoring systems to maintain the integrity of assessments. These systems employ various technologies, including computer vision, machine learning, and artificial intelligence, to monitor test-takers' behavior and detect potential cheating. We will be looking into how those previous applications behave later

2.2 Real-Time Object Detection in Online Exam Proctoring

One approach to monitoring user behavior is to treat specific actions within a video frame. For example, a student's gaze could be classified into discrete classes like gaze_left or gaze_center. A leading algorithm for this type of real-time application is YOLO (You Only Look Once) [5], an object detection system that has evolved through several powerful versions (YOLOv1, v2, v3, and so on)

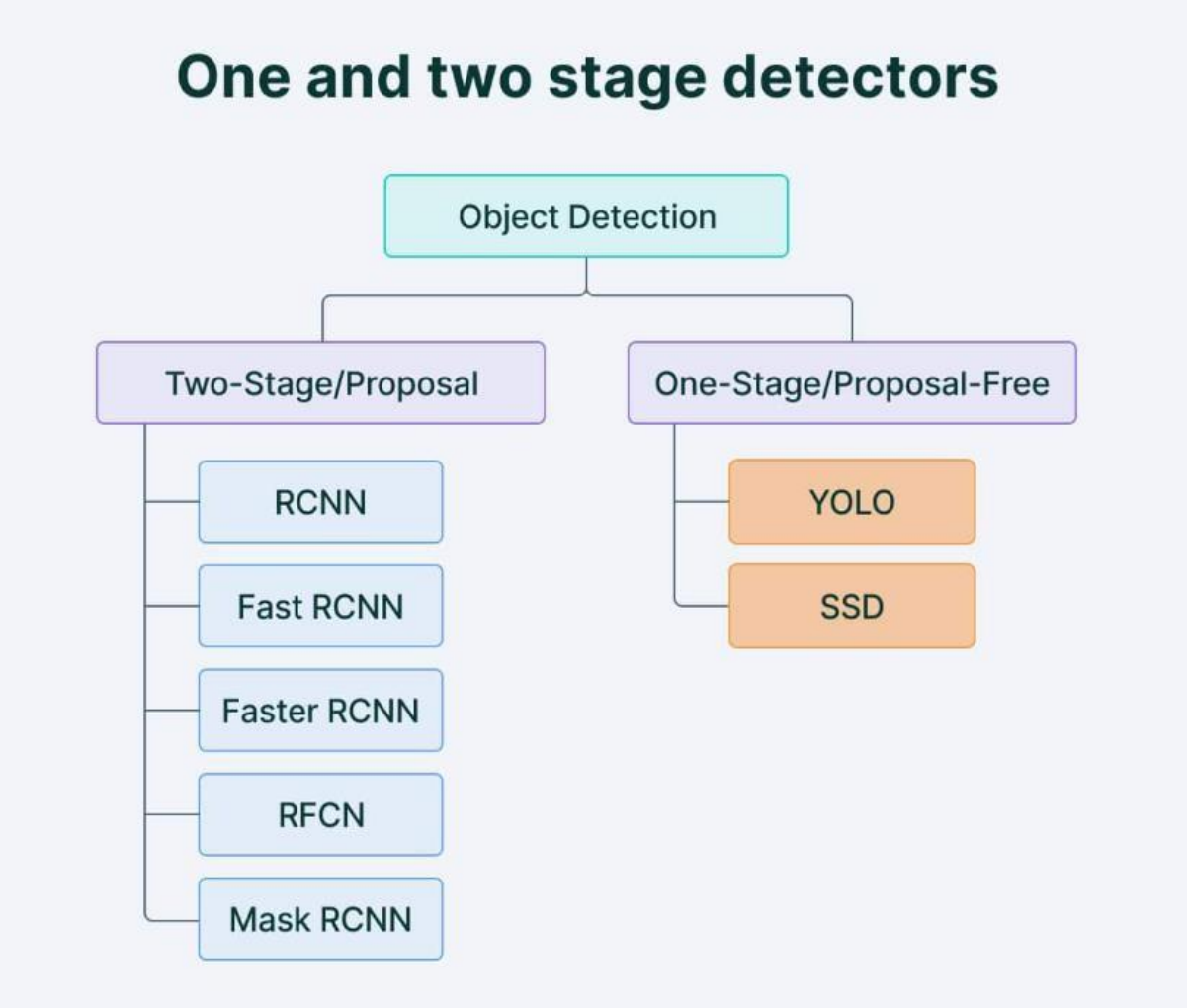


Figure 2.1: Type of Object Detection

Modern object detection can be divided into 2 types which 1 stage and 2 stage detectors [3] which One stage will be more efficient but have slightly lower accuracy in this case [2] while the 2 stages will be achieving higher accuracy because it has an additional step than the one stage object detection which we will discuss afterward, however, this also led to increase in computational resources and slower the process

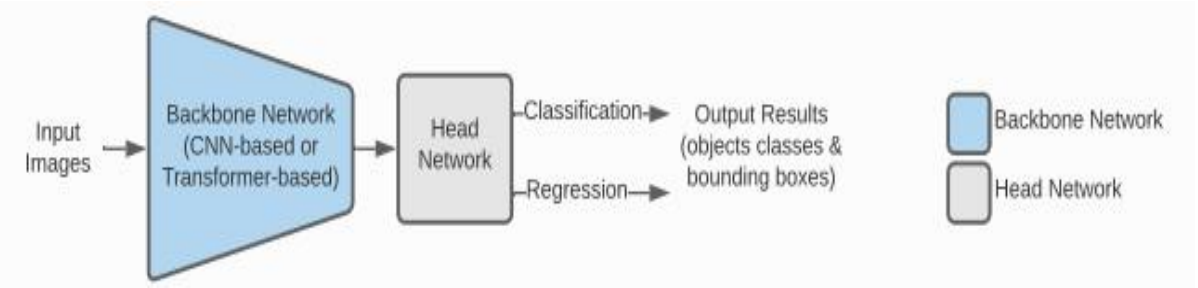


Figure 2.2: Overview of One-Stage Object Detection

Diagram showing the one stage object detection, the input image will go into the backbone network where there is the core of the model, to extract the feature from the image by mapping the capture information at different levels of abstraction like the edges, texture, object parts and so on. Then, the head network will be performing both the classification and regression task simultaneously. Classification means to predict the likelihood of the object to different classes while the regression means to refine the coordinates of the bounding boxes to enclose the objects. Thus, this shows why the speed of the one stage object detection will be faster, because of the simplicity of the step. While the cons will be the accuracy won't be too high. This approach will be suitable for real-time application where some of the accurate trade-offs are acceptable for overall performance.

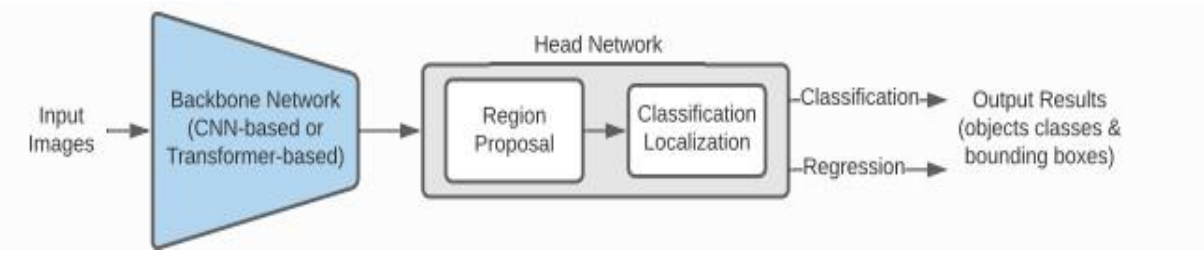


Figure 2.3: Overview of Two-Stage Object Detection

The diagram above shows the two stages object detection, can be seen that the previous steps are the same as the one stage object detection, which the image go into the backbone network where there is the core of the model, to extract the feature from the image. Next, the head network unlike the one stage object detection will be divided into 2 steps. While the first step is a region proposal, it means to predict the possible object location in the image then go into the classification localization to classify the objects within those predicted region. Thus, this model will have higher accuracy but with lower speed than the one stage object detection. However, in different cases, it will be more suitable to use this model when it comes to the scenario that object detection is important, even with the tradeoff of speed we have to focus on the output accuracy. Systems like the R-CNN family (R-CNN, Fast R-CNN, Faster R-CNN) [26] practice this approach. Their careful analysis yields high accuracy, but the computational cost makes them largely unsuitable for real-time video processing.

While for the detecting online test cheating through User behavior monitoring will be having to predict the user behavior in real time, so that alert can be raise to the supervisor or called as proctor of the examination in real time and action could be taken immediately, and also alert can be reached to the students to warn their action. Thus, YOLO will be the ideal

algorithms for this project. While the which YOLO will be considered under several factors: [5]

1. **Accuracy**:
   Generally, the newer YOLO version will have higher accuracy in object detection. However, the newer the version will be increased in computational demands.
2. **Speed**:
   Real-time processing is important for this project and thus speed is an important factor to ensure that the system won't miss capturing the key frame that is important to be the evidence showing cheating
3. **Computational resources**:
   This project will be carried out using a normal home use laptop but not a commercial use with greater computational power and thus newer version YOLO not what to pursue, but an algorithm that just enough and efficient to carry the job will be good enough for this project.
4. **Ease of development**:
   The older version of YOLO also has advantages that have plenty of resources and tutorial available online. Besides it is relatively well- established while the newer version might require additional expertise or adjustments needed.

Thus, to conclude that while YOLO to be deploy will be needed to have a balance between the speed and accuracy of the output, and this will need further study and examine in the future work. The project will be choosing a YOLO version that aligns to the project priorities and objectives.

2.3 Facial Landmark Detection

An alternative for gaze tracking is high-precision facial landmark detection approach instead of object classification like YOLO we mentioned in chapter above. This method is to leverage specialized deep learning models to map a dense mesh of landmarks, on the face in real-time. Instead of trying to classify a general direction (Left, Right, Up, Down, and Center), this method uses the geometric relationship between these landmarks to return a precise gaze vector. The accuracy believed to be much higher accuracy than a general object detector.[16]

2.3.1 Google MediaPipe Face Mesh

MediaPipe Face Mesh is a high-fidelity, real-time face and facial landmark detection solution. It employs machine learning (ML) to map a detailed 3D mesh of human face from camera input, making it a foundational technology for various applications like augmented reality (AR) effects and virtual avatars.[10] This is the technology that was successfully used in the FYP1 proof-of-concept prototype.

The Face Mesh model is a lightweight deep neural network designed for on-device and real-time performance.[11] It predicts 468 3D landmarks that map to the geometry of a human face. In addition, these are not just 2D (x, y) coordinates; the model also provides a metric depth value (z), allowing for a 3D mapping of the face's orientation. This 3D has significant advantage that it can help differentiate between a change in gaze direction. The high density of landmarks around the eyes allows for the precise localization of the iris, from which a robust 3D gaze vector can be calculated.

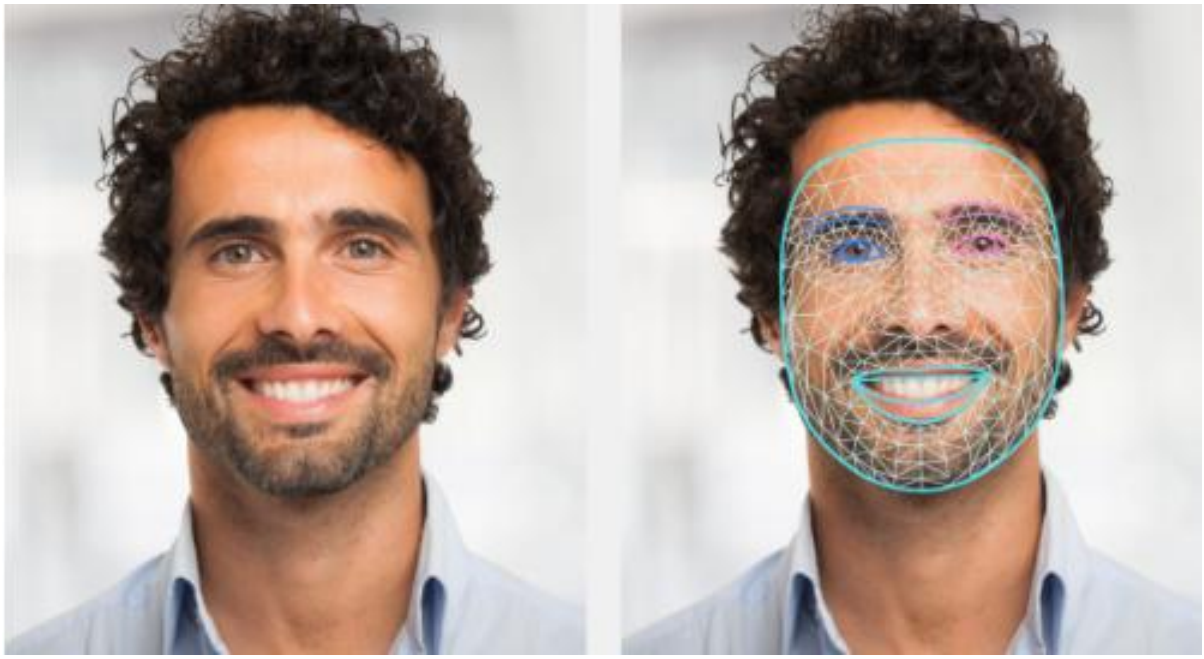


Figure 2.4: Demonstration of facial landmark detection

MediaPipe strength is its delivery of extremely accurate and rich raw data, which can be processed using libraries like OpenCV [17]. For a proctoring system, this data can be used to build a highly precise model of a user's attention. However, MediaPipe's main role is to provide the landmarks; it does not inherently provide a system for mapping these landmarks to on-screen (x, y) coordinates. A developer would need to build a regression and calibration system to make the landmark data useful for tracking on the screen.

2.3.2 WebGazer

WebGazer is an open-source library that represents a more complete, end-to-end application of the facial landmark approach, built specifically for the web browser.

WebGazer also uses a facial landmark detection model to locate eye features. Its primary innovation is use of online learning through user interaction with the goal of democratizing web usability studies. Previously, the research on where users look on a website was a field that required expensive, specialized hardware eye-trackers that is expensive yet complicated, limiting this technology to large corporations only. With WebGazer.js, a free software solution that works with any webcam, makes this powerful analysis tool accessible to everyone. [14]

Figure 2.5: Official logo for WebGazer.js

The library includes a regression model that is continuously trained and updated with every user click on the screen. This process serves as a real-time, personalized calibration, creating a dynamic mapping between the user eye features and their position on the screen. This allows the model to adapt to real-world situations like shifts in lighting, changes in the user posture, or even anatomical differences between users. [12]

WebGazer self-training nature makes it suitable for a proctoring application. Instead of relying on a static model, it creates a personalized tracker for each session, defined by the calibration process. This capability is important for achieving reliable accuracy in the uncontrolled environments of online examinations. [18]

WebGazer achieves its high accuracy through two key innovations:

1. **3D Facial Landmark:**

   This is able to estimate the user head pose, allows it to detect head rotation, solving the head movement problem in simpler 2D implementations.

2. **Adaptive Regression:**

   The model is able to train in real-time through a user-specific calibration process. By having the user click on points, WebGazer builds a personalized map between the 3D eye features and the 2D screen coordinates, making it a highly accurate and personalized tracking tool.

3. **In-Browser Machine Learning:**

   The entire machine learning model runs directly on the client computer inside the web browser. This is highly efficient as it avoids sending the heavy webcam video stream across the internet for analysis, saving significant network bandwidth and reducing the load on the backend server.

2.4 Head Pose Estimation

Head Pose Estimation (HPE) calculates the 3D orientation of a student head pitch (up/down), yaw (left/right), and roll (tilt) to approximate where the head is facing without directly tracking the pupils [21]. This method can be broadly classified into two main approaches which are landmark-based and landmark-free (appearance-based) methods.

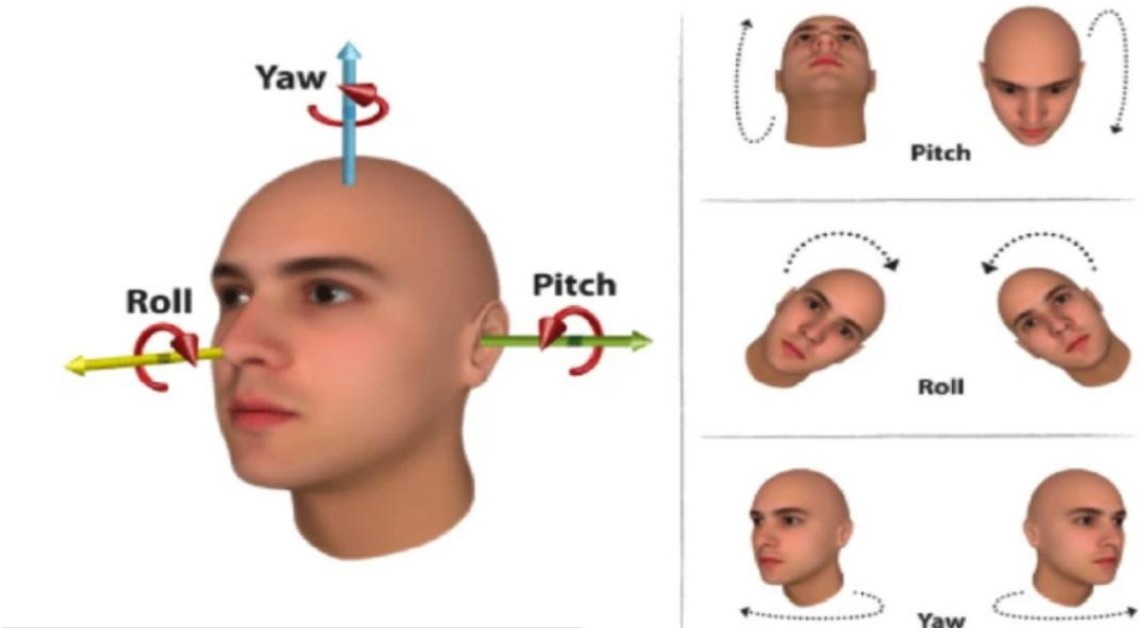


Figure 2.6 Parameter for HPE (Pitch, Yaw, and Roll) and Their Corresponding Direction

In the landmark-based approach, the system first detects key facial landmarks such as the corners of the eyes, nose tip, and mouth.[19] These 2D points are then mapped to a standard 3D head model, and algorithms used to estimate the head rotation and translation in 3D space [21]. This approach is computationally efficient and works well under controlled conditions with good lightning and simple background, ensure that the mapping is working as intended [18].

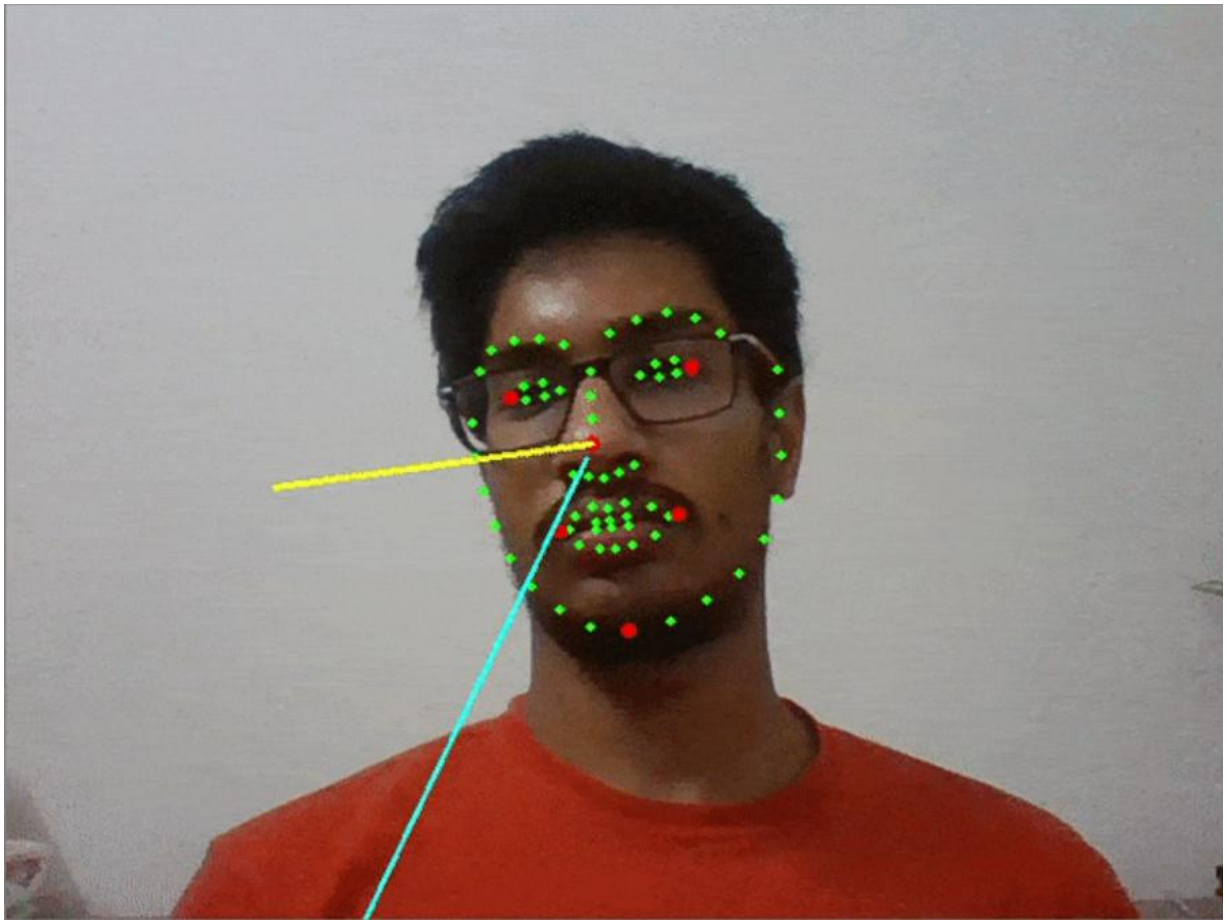


Figure 2.7 Landmark-based Head Pose Estimation

On the other hand, landmark-free methods skip landmark detection. Instead, employ deep learning models, such as CNNs, to directly regress the head pose angles from the entire face image [18]. These methods are generally more robust to challenging environments, such as complex backgrounds, or extreme lighting conditions, because the model learns features automatically from data rather than relying on precise landmark positions. However, landmark-free approaches often require more computationally intensive than traditional landmark-based methods.

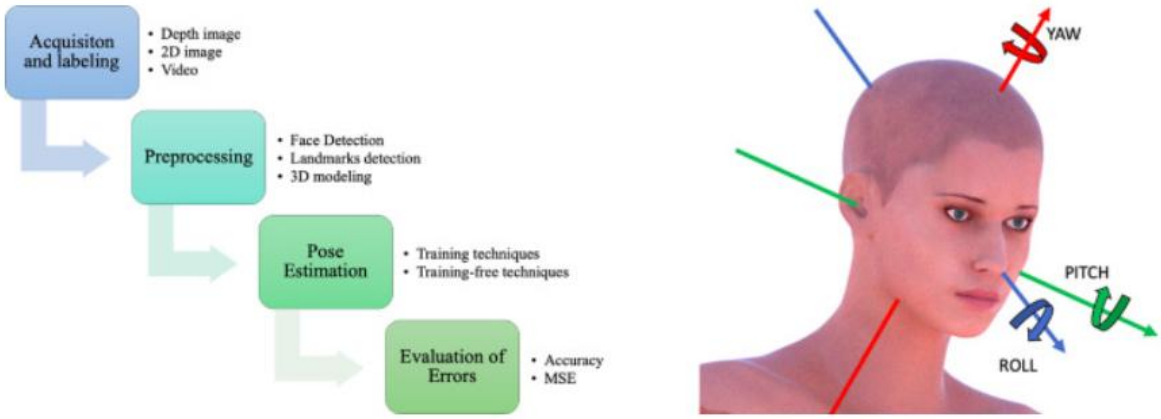


Figure 2.8 The Steps to Train a Landmark-Free Methods for HPE

2.4.1 Tools for Head Posture Estimation

Several tools and libraries support the implementation of head pose estimation system. OpenCV solvePnP function is one of the most widely used methods. It estimates head rotation and translation by mapping 2D facial landmarks, such as eye corners and nose tips, to a predefined 3D face model [22]. This approach is computationally efficient and provides real-time performance, making it suitable for low-resource environments.

MediaPipe Face Mesh is another powerful tool that provides 468 high-fidelity 3D facial landmarks, including depth information, from a single camera input [20]. These landmarks allow for accurate head orientation estimation even under moderate variations in lighting or head positioning.

2.5 Non-Computer Vision Approaches for Exam Proctoring

While computer vision techniques focus on detecting suspicious physical behaviors through camera input, non-computer vision approaches aim to further secure the digital environment and audio environment of online exams. These methods include browser and keystroke monitoring and audio analysis. Non-computer vision has an advantage that the incident that recorded is somehow with strong evidence proof as they are unlike the computer vision to detect the behavior which is indirect to the cheating as the main evidence of cheating which the unauthorized resources are usually placed at the blind spot of camera.

2.5.1 Browser and Keystroke Monitoring

Browser and keystroke monitoring methods analyze a student digital activity rather than their physical behavior. The primary goal is to detect activities such as tab switching, window focus changes, or unusual typing patterns that may indicate cheating attempts.

For browser monitoring, Chrome Extension APIs provides several useful events, such as chrome.tabs.onActivated to detect when a student switches to another tab, and chrome.windows.onFocusChanged to detect when the browser window loses focus which maybe minimizes the window or open another application [4]. These techniques are suitable for both custom browser extensions and web-based exam platforms [24].

In addition, keystroke dynamics analyzes typing behaviors, including key press durations, latencies, and inter-key intervals, to detect anomalies in user behavior. For example, a sudden shift in typing speed may suggest that someone else is typing for the student [23].

2.5.2 Audio Analysis

Audio analysis uses a computer's microphone input to detect suspicious speech or background sounds during an exam session. The Web Audio API provides real-time access to microphone data, while Voice Activity Detection (VAD) algorithms automatically detect when human speech occurs. To further enhance functionality, the speech could converts into text, enabling systems to flag suspicious keywords such as "answer" or "help me"

However, privacy and ethical concerns remain a major challenge. Continuous microphone access may capture sensitive background conversations [6]. Moreover, ambient noise, or overlapping voices can reduce accuracy and lead to false positive incident [25].

2.6 Discussion on Selected Proctoring Techniques and Tools

Based on the reviewed methods, this project focuses on two main approaches: Eye Gaze Tracking under computer vision methods and Browser Event Monitoring under non-computer vision methods. These choices are made after considering the practical constraints and suitability of each method in the context of an online examination environment.

For Eye Gaze Tracking, the key reason is its ability to detect cheating behavior even when the student keeps their head still. Head Pose Estimation (HPE) alone would fail in such scenarios because a student can glance at notes or another screen using only eye movement without turning their head.

Browser Event Monitoring is included because cheating via digital methods such as searching for answers online or switching to unauthorized tabs for external reference is one of the most common and easiest forms of cheating during online examinations. Detecting tab switching, or loss of browser focus can significantly enhance the system capability to prevent academic dishonesty with minimal privacy concerns.

On the other hand, keystroke monitor and audio analysis were excluded for practical reasons. Keystroke dynamics may produce many false positives, as normal variations in typing speed or style could be mistaken for suspicious behavior, especially when many students are being monitored simultaneously. Audio analysis also presents challenges: it requires third-party tools for speech detection, involves privacy and ethical concerns, and struggles with noisy environments or multiple overlapping voices, leading to frequent misclassification.

By combining Eye Gaze Tracking for physical behavior detection and Browser Event Monitoring for digital behavior monitoring, this system strikes a balance between accuracy, simplicity, and practicality, while respecting user privacy and reducing computational complexity.

2.7 Existing Methods for Detecting Online Cheating

Traditional methods for detecting online cheating have primarily relied on physical in-person proctoring together with strict rules and regulations during the examination. However, this is insufficient to address the challenges that may arise during the online examination because the in-person proctoring method that works well for traditional physical proctoring is nearly impossible to monitor the whole online examination process and hard to trace the evidence showing cheating behavior, while it is also expensive to scale.

Besides, students can easily take advantage of cheating beyond the strict rules and regulations, due to online examination being hard to detect cheating. The traditional approach is also impractical to practice during the online examination. Therefore, there is a need to have a more reliable monitoring system towards the student's behavior during examination. There are few numbers of previous applications that are built to solve the problem which are commercialized and target to purchase services by the institutions or organizations, each using AI and computer vision approach in detecting cheating during examination. This section will review ProctorU, Respondus Monitor, and Proctorio[6][7][8], focusing on their methods of behavior monitoring, and strengths and limitations.

In addition, the proposed solution was also drafted to deal with the limitations of these systems stated later.

2.7.1 ProctorU

ProctorU [6] is an online proctoring service that allows subscribing institutions to remotely monitor students during online exams. Unlike the fully automated systems, ProctorU still involves humans in the proctor session because they believe the technology can't replace human in the examination, live proctoring to monitor the whole examination to ensure confidentiality of the examination is important. Therefore, ProctorU combines live proctoring, AI-based monitoring, and recorded sessions to support the entire examination process. ProctorU will have their own trained proctor to help the university to handle the whole examination process, institutions have no need to send supervisor or proctor themselves to overlook the whole examination process, this reduces the manpower required by the institutions to monitor a online examination.

Test-takers that have an examination with ProctorU will need to have 8 to 10 minutes for the setup process before they take the exam. The system will have a strict rule for the test-taker to follow before and during the test ongoing. Before the test begins, test-takers will need to make sure that their desk is free from unauthorized belongings like the second monitor, tablet, mobile phone, reference book, sunglass, earbuds and so on. In addition, test-taker asked to take the test under an ideal environment which free from other people.

Afterward, test-takers are required to install the Chrome or Firefox extension for ProctorU based on the browser they are using for examination, close all the program and restart their computer. They are also not permitted to leave their seats during the test ongoing. Additionally, students must download an applet file (small downloadable file) to proceed.

During the test, test-takers are required to share their screen, take photos of themselves and take photos of their ID for verification purposes. The photo will be required to receive approval from the proctor which the supervisor of the exam, proctor will also be required to check and ask the test taker to remove unpermitted materials on their surroundings like a second monitor on the table and review the test taker open application on the PC. Only then is the test taker able to login into account and take the examination. The test taker screen and webcam will be recorded all the time during the exam and must follow some rules and regulations during the test like don't read the exam question out loudly, this believe is because prevent case that the students read the question and ask help from the unknown person behind the screen, test-taker also not required to leave their seat during the examination and not to allowing anyone to enter the exam area to prevent cheating case from happening.

ProctorU is a well-developed and reliable system that is able to prevent cheating case happening during the online examination with such many exams environment regulation, ID verification and live monitoring on the student's behavior. The system behind the screen also uses eye gaze tracking, head pose and body movement tracking, and screen and audio monitoring to further detect the cheating from happening. After the examination ends, the suspicious activity will also be recorded, and report shared to the institution administrator.

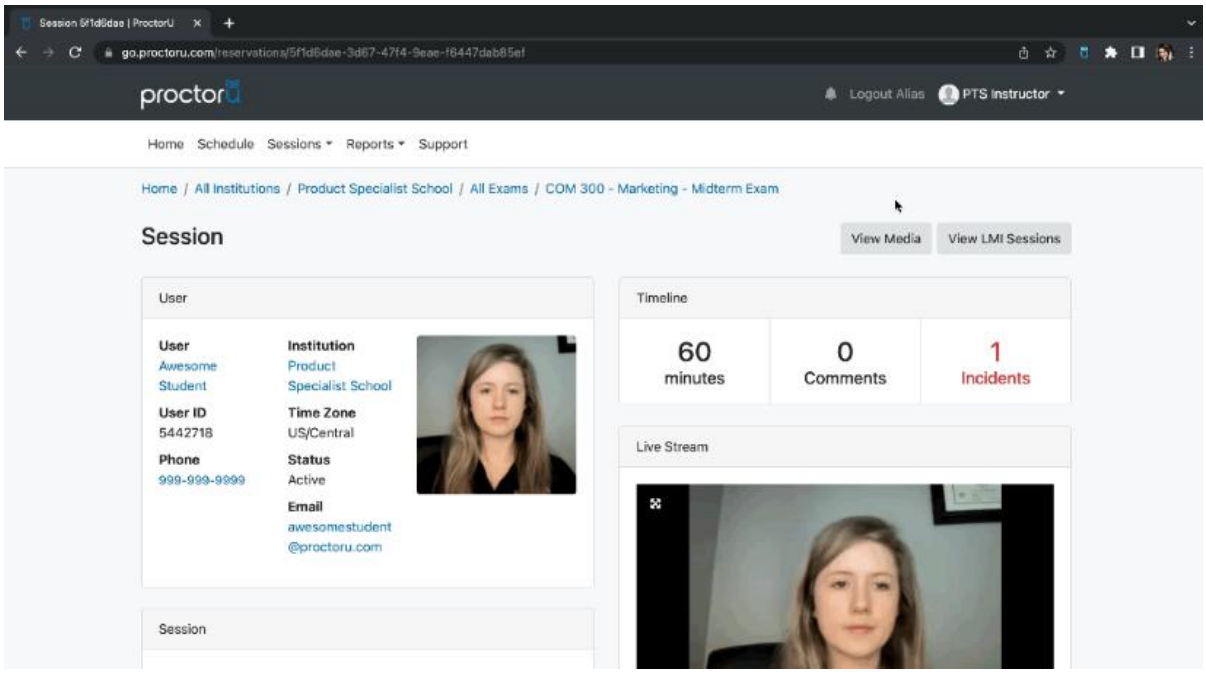


Figure 2.9: ProctorU Test-Taker Exam Session Interface

2.7.2 Respondus

Respondus [8] is also an AI-based proctoring tool designed to work alongside lockdown browser, an innovative method that uses a browser that locks test-takers access to other unauthorized material within a learning management system (LMS). LMS is like a digital school or training center. It is a platform to allow creation and organize online courses, deliver them to public and track the progress. Besides, it allows the users to upload video lessons, quizzes or assignments. These platforms are normally in paid content, some will be assigned professional certificates for those that pass the test. Examples of LMS are like the Alibaba Academy, Canvas and so on.

Respondus enables institutions to remotely proctor online exams without the need for live proctors. In another word, Respondus is fully utilizing automated AI monitoring and video recording to detect suspicious behaviors during the exam, ensuring that test-takers follow to the rules and regulations as well as maintain the integrity of the assessment process. This

scalable solution is especially suited for large institutions looking for a cost-effective way to monitor online exams without real-time human intervention. Thus, save cost in terms of manpower.

Firstly, Students access the exam through their institution's LMS, where Respondus Monitor is integrated to the system. The listed LMS that has cooperated with Respondus are as shown.

Figure 2.10: List of Respondus LMS Partners

Before starting the exam, students are required to install and use the Lockdown Browser. This browser restricts access to other applications, websites, and system functions, preventing students from looking up answers or using unauthorized resources during the exam.

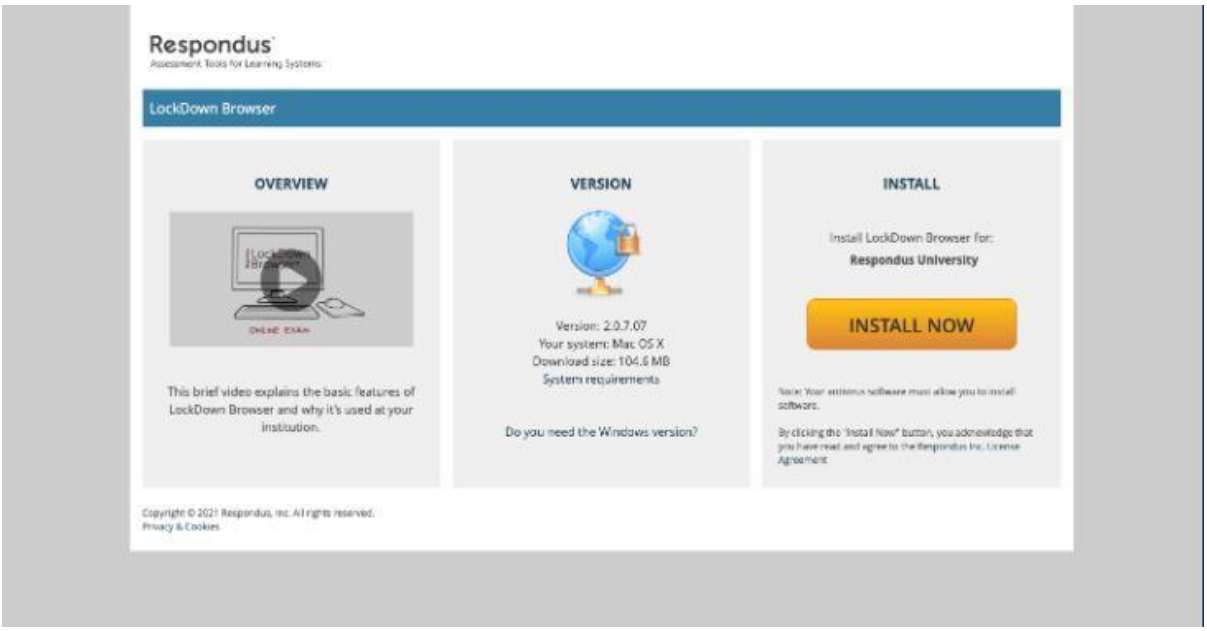


Figure 2.11: Lockdown Browser Installation of Respondus

Afterwards, Respondus performs a series of pre-exam checks to ensure that the student's computer meets at least the minimum technical requirements for examination, including a working webcam and microphone. Students are also asked to complete an environment scan using their webcam to show their surroundings, ensuring there are no unauthorized materials like books, phones, or other people in the room. All of these are

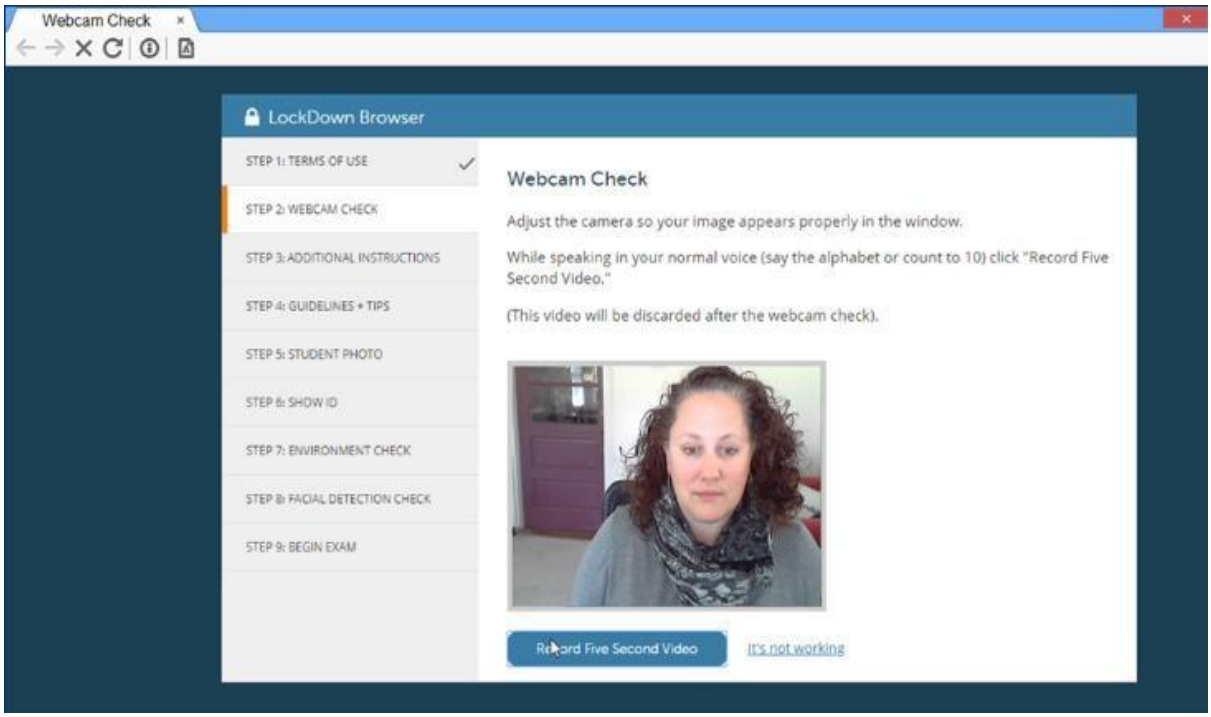


Figure 2.12: Pre-examination Check of Respondus

Before beginning the exam, students must verify their identity by showing a valid photo ID to the webcam. This ensures that the person taking the test is enrolled. Once the setup is complete, the exam begins. Respondus Monitor continuously records the student's webcam and screen activity throughout the test. The system uses AI to monitor the student's behavior, analyzing facial movements, eye tracking, head posture, and background noises to detect any signs of cheating or suspicious activity. The lockdown browser ensures that students cannot access unauthorized websites, applications, or resources during the test.

Respondus Monitor uses AI to analyze student behavior in real-time, detecting possible instances of cheating by identifying suspicious patterns, such as face and eyes as well as head posture and body movements to detect if they are too frequently looking away from the screen, which potentially show that the test-takers is looking for unauthorized materials. If the students perform this action, the system flags the behavior as suspicious.

Besides, the system listens to background noises, this is because students may receive help from someone behind the camera screen which is blind spot. Any unusual noises are flagged as potential cheating behavior. Since Respondus works with the lockdown browser, test-takers are unable to navigate away from the exam screen or open new browser tabs. Any attempt to break out of the browser's restrictions, such as trying to access external content or use keyboard shortcuts, is automatically blocked and flagged.

After the exam is completed, Respondus generates a detailed report to the institution's administrator. The report includes each suspicious activity detected by the AI during the exam flagged in the report. This includes behaviors such as looking away from the screen, excessive movement, or background noise. Each flagged event is recorded with a timestamp and classified by the type of behavior. The system provides administrators with access to the full video recording of the exam session, along with any flagged incidents. This allows administrators to review the footage and make decisions about whether a cheating occurred. Respondus categorizes flagged behaviors by different levels. For example, minor issues like brief eye movements may be marked as low risk, while more serious violations, such as repeated head turns or external conversations, may be marked as high-risk. The system offers an analysis of student behaviors throughout the exam, helping administrators identify patterns that may be cheating.

2.7.3 Proctorio

Proctorio [7] is an AI-driven online proctoring service that allows subscribing institutions to monitor students during online exams without the need for live human proctors but at the same time it still opens the option for the institutions whether to include human proctor in the online examination. Unlike traditional proctoring services, Proctorio relies entirely on machine learning and artificial intelligence to monitor test-takers' behavior throughout the exam. By removing the need for human proctors, Proctorio offers a scalable solution that can be integrated directly with learning management systems (LMS). Its goal is to provide real-time, automated monitoring to ensure the integrity and security of online assessments. One another good thing with the Proctorio is that it offers limited function free usage for the public unlike the others proctor platform all using subscription method made by the institution. This means that small organizations or normal individuals can use the service up to certain extent.

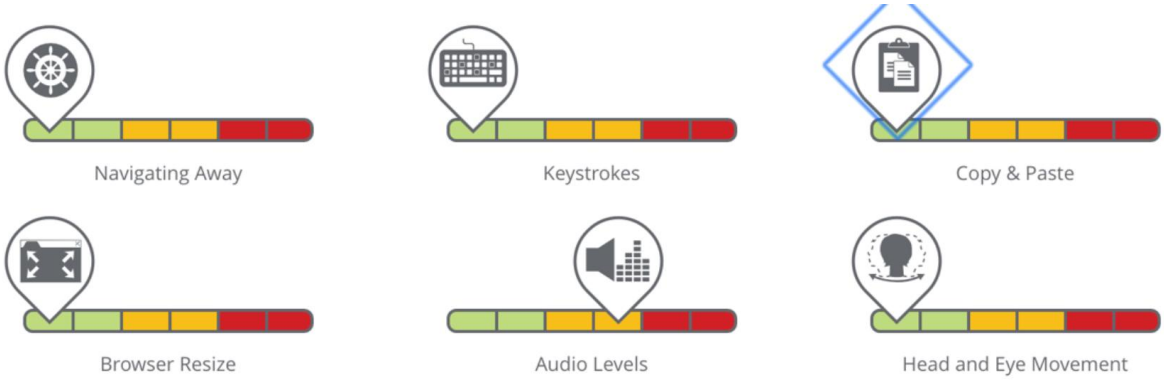


Figure 2.13: Proctorio Allow Limited Configuration to the User

Proctorio Exam Results

	Name	Submission Time	Attempt	Score	Annotations	Abnormalities	Suspicion Level
	Student, Andreav...	03/02/2020 04:27:...	1	1	2	0	69%
	Student, Vuk De...	03/02/2020 09:23:...	1	3	0	1	18%
	Student, Ana De...	03/02/2020 09:18:...	1	3	0	1	16%
	Student, Alekhya ...	02/27/2020 10:36:...	2	3	0	0	12%
	Student, Alekhya ...	02/27/2020 10:31:...	1	3	0	0	9%
	Student, Javan D...	02/28/2020 04:39:...	1	3	0	0	5%

Figure 2.14: Examination Result of Proctorio

Students taking an exam through Proctorio undergo a setup process similar to other online proctoring services. Before the exam, students are required to clear their workspace of unauthorized materials, including additional monitors, mobile phones, books, and other unauthorized items. They are also asked to ensure their exam environment is free from disturbances, meaning no other people should be present in the room.

Proctorio requires students to install a browser extension, typically compatible with Chrome, to run the proctoring software. Once the extension is installed, the system performs a series of checks to ensure the student's computer meets all technical requirements, such as verifying the presence of a webcam, microphone, and stable internet connection. Additionally, students are not allowed from leaving their seat during the exam, and any attempt to do so may be flagged by the system as suspicious behavior.

When the exam begins, Proctorio continuously monitors the test-taker using AI. Students are required to share their screen, activate their webcam, and allow access to their audio feed. The system captures these data streams and monitors them throughout the test. One key component of Proctorio's system is facial recognition, which verifies that the individual taking the exam matches the original student registered for the test. The software also tracks eye movements, head posture, and body language, flagging behaviors that may indicate

cheating. Proctorio additionally monitors the student's screen activity, ensuring that no unauthorized websites, applications, or other tools are being used during the exam. The system even monitors audio for sounds that may indicate that there is collaboration with someone else in the room.

Proctorio automated system also records video and audio for post-exam reviews. All data collected during the exam, including screen recordings and any flagged incidents, are compiled into a report that is sent to the institution's administrator once the exam is completed. This report includes a suspicion score, which indicates the likelihood of cheating based on the behaviors and anomalies detected during the test. Each flagged behavior is accompanied by timestamps, and administrators can review the footage to determine whether further action is needed.

2.8 Critical Analysis of Existing System

As a short conclusion for the previous section, to create a robust and reliable test monitoring solution requires multiple technology and functionality work together. For example, the application ProctorU which is believed to be one of the biggest proctoring systems worldwide, proposes that effective online test monitoring is more than software. The company is not only a software provider but also provides trained human monitors which are known as proctors who directly monitor exam sessions in real-time. This enables institutions to leave the entire exam monitoring task to ProctorU, saving the internal manpower and also ensure the integrity of the examination.

On the other hand, Respondus introduces innovation through the use of multiple solutions in the application such as lockdown browser, background audio monitoring, and identity verification protocols. This further enhances exam security by limiting access to unauthorized resources and detecting suspicious cheating behavior besides that using eye gaze and head posture monitoring. Lastly, Proctorio further innovation in having a system that is fully automated without third party monitoring, the system itself will monitor the whole examination and create a report afterwards, offering a scalable and cost-effective option.

They all have their pros and cons in the way that the characteristics may bring a competitive advantage to them but in the same time also bring some disadvantages alongside. This section will have a critical analysis on these platforms which ProctorU, Respondus, Examity, and Proctorio [6][7][8][9] then explains how the finding able to integrate into this project. These findings provide the foundation for proposing a more balanced and effective

solution through the "Detecting Online Test Cheating Through User Behavior Monitoring" system.

2.8.1 ProctorU

ProctorU has strength that presents a very reliable online test proctoring system. It is a well-known largest network of certified remote proctoring and support staff; the nature of the application is a combination of both live and artificial intelligence indicates that there are layers of monitoring in place. When ProctorU's proctor notices cheating, they will make a decision including informing the institutions and remark the incident too, the system will also generate a report to the institutions regarding the cheating behavior. Proctor will be there to monitor the entire examination process and because the existence of humans enables real-time intervention, the proctors can intercept the examination in the name of the institution when there is a case of cheating is observed to be ongoing.

However, this kind of business model doesn't really work for every institution. This is due to the exam content either question or answer are confidential for universities and companies, they are sensitive stuff. The business model of ProctorU indicate that institutions are basically handing all those confidential materials over to a third party. This make the confidential materials hard to handled and controlled confidentially. There might happened some case like the proctor accidentally leaks questions. Additionally, there is also no guarantee every proctor will treat their job responsibly. Some might be very strict, while others could miss obvious cheating even the suspicious case raised by the system, or in the worst case, proctor maybe offered some benefits by the students. At the end of the day, institutions will loss lot of control towards the examination when it is outsourced to ProctorU. Even though ProctorU will have SOP to prevent these cases from happening but there is still risky to do so, this is one the concern when the institutions choose ProctorU to host their examination. Additionally, involvement of the proctor that outside the organization also an extra expense for the institution.

In addition, ProctorU pre-exam setup cause inconvenience for the test- taker. In most the time, students may not have the full control of their exam environment such as the arrangement of the desk; the conditions in the room are factors that the students cannot control of. The strict rules that require them to adhere will cause so many complications and unnecessary stress to the test takers because the complicated setup is also a waste of time action for examinees.

Besides, ID card approval process of ProctorU is by manually checked by the manpower, even though this will be more reliable, but it slows down the process and in the case that the number of test takers was too many and the number of proctors is limited, the examination maybe couldn't be started on time for all the students. Lastly, the subscription-based system is making ProctorU primarily available to larger institutions that can afford the subscription price. This creates a disadvantage for smaller institutions and individuals, limiting the adoption of online examinations and the digitalization in society.

Advantages	Disadvantages
- One stop service provider as online examination monitoring outsourcing provider - Combines both live proctoring and AI-based monitoring - Real-time intervention allows proctors to step in if suspicious behavior - Detailed reporting with recorded video and flagged behaviors - Provides its own trained proctors	- Leakage of confidential data, examination questions and questions in this case - Complex setup process causes stress and inconvenience for students - High cost due to external party involvement - Manual ID verification can delay the start of exams when proctors have to review IDs for large groups of students. - Subscription-based of the system be extra burden for institutions.

Table 2.1 Advantages and Disadvantages of ProctorU

2.8.2 Respondus

Respondus offers a solid online proctoring solution by integrating AI-driven monitoring with lockdown browser. This combination ensures that the testing environment remains secure while detecting suspicious behaviors like abnormal eye movements or head gestures. Respondus Monitor generates detailed reports with video recordings, flagged incidents, and behavior analysis, allowing administrators to have a comprehensive overview of potential cheating cases. By categorizing incidents based on different levels, institutions can focus their efforts on serious violations that are highlighted in reports, making the post-exam review process more efficient.

However, Respondus Monitor has some limitations. One major weakness is its limited real-time intervention during proctored exams. In these cases, the system does not alert students

during the test if they are doing suspicious activities but only flag them for future report generating for the institutions only, which could allow cheating to continue unnoticed until the review after the exam. This is lack of transparency for both the student and proctor. Instead, the system could be further improved by addressing this issue by incorporating a more dynamic system that alerts students when they engage in suspicious behavior, providing them with a chance to correct their actions during the exam. Also, in the case it is a false positive, the students also have a chance to sound out and clarify their innocence. Allowing real-time notification in automated proctoring would make the system more transparent for both parties and help prevent cheating in real time, and in the same time prevent false positive as the test is important that it will affect the future of the students, students have a right to protect themselves when it is really a false positive case.

In addition, it relies heavily on AI for detecting suspicious behavior, which may result in false positives as mentioned, where normal student actions, such as adjusting their seating or glancing away from the screen but not trying to cheat, may be flagged as suspicious. This not only increases stress for students but also adds more work for administrators who must review these incidents. A solution that only relies on the AI on an online test monitoring work is not really an ideal approach, because AI is not 100% perfect, the involvement of human proctor is still important to have. Additionally, the system does not offer real-time human intervention, meaning any potential cheating is only flagged after the exam is over. This could allow cheating to go undetected during the examination.

Another issue is the system's rigid exam environment requirements. Students in shared environments like coffee shops or shared rooms with others may face unnecessary stress if they cannot comply with strict rules that the room can only have one person with simple setup, resulting in false flags. Moreover, Respondus system is highly dependent on its integration with specific LMS platforms, like Canvas, Blackboard, Moodle and so on those LMS that only specify in their official website as their partners. This can limit its flexibility and may restrict its usability for some institutions. Although integration into LMS may be a good choice and bring convenience but it also limits the scalability of the system.

Advantages	Disadvantages
- Integrated with lockdown browser - Comprehensive reports with video recordings and behavior analysis	- Over-reliance on AI - No real-time human intervention - Complex setup process causes stress and inconvenience for students - The system's dependence on integration with specific LMS platforms - Lack of real-time alerts in proctored exams means proctor may not be aware of suspicious behavior until after the exam. - Transparency issues, as the system does not provide students with real-time alerts

Table 2.2 Advantages and Disadvantages of Respondus

2.8.3 Proctorio

One another online test proctoring system, Proctorio is an AI-enabled proctoring system that does not require actual people to manage and monitor the process, thus it is cost-efficient and can work at a large scale. This flexibility that can be adjusted depending on the specifics of the exam makes Proctorio a perfect solution for large scale institutions since it majors in AI monitoring hence cutting down on human resource input and costs. However, Proctorio has an extra benefit that the public can use their services with restrictions as a free subscriber. This is good since other proctor platforms do not have these services, most of the services of online monitoring application are subscription based that can only be accessed by institutions that subscribed to their services only.

However, Proctorio has limitations. Because the system is fully dependent on artificial intelligence, the system notifies the cheating case after the examination if it detects any suspicious behavior during the exam, and hence cheating can actually go unnoticed during the live exam session. Then, in the end of the examination the supervisor finds it hard to trace back the cheating case, increasing the workload to aliasing with the students on this case. Another disadvantage is that AI-based identifies many normal behavioral patterns of students such as a student changing their seat position or looking away from the screen but not trying to cheat as malicious, thus it has high false positive results. This is due to AI not being 100% accurate. Moreover, the constant capturing of the video and audio alongside the screen activity is a major concern of privacy, especially to the students, especially when in their own space. Proctorio

also has a problem in which minor forms of cheating and misbehavior are strictly monitored, and accidental movements and sounds are considered violations and thus cause stress to students.

Advantages	Disadvantages
- Cost-effective and scalable, making it a suitable solution for large institutions with many students. - Customizable exam settings, allowing institutions to configure parameters and decide which actions should be flagged as suspicious. - Limited function access for the free users	- No real-time intervention, allowing cheating to go undetected during the live exam and only flagged post-exam. - Fully automated AI monitoring caused over-reliance on AI

Table 2.3 Advantages and Disadvantages of Proctorio

2.9 Proposed Solution

The analysis of these commercial systems highlights a clear gap in the current market. Existing solutions often force a choice between high-cost, human-proctored services that can introduce data confidentiality risks, and fully automated systems that may lack real-time feedback and institutional control. This suggests the need for a more balanced approach.

Therefore, an ideal solution should be low-cost, flexible enough to integrate with existing testing platforms of the institutions and designed to keep sensitive exam data under the institution's control. Furthermore, such a system would benefit from incorporating real-time alerts to both suspicious students that caught violent actions and inform administrators, addressing the transparency limitations of systems that rely only on post-exam reports.

In conclusion, this project proposes a solution that combines real-time monitoring and alert to provide immediate feedback. These core functionalities, together with other features detailed in the following chapters, are designed to create a effective eye-gaze tracking system that successfully meets the objectives of this project

CHAPTER 3 SYSTEM DESIGN

This chapter details the high-level design of the "EyeGuard" proctoring system, presenting its technical blueprint through a series of standardized diagrams. The following sections will present the system's three-tier architecture, detail the functional requirements using a Use Case Diagram and descriptions, and illustrate the system lifecycle with an Activity Diagram.

3.1 System Architecture

Figure below is the system architecture for the "EyeGuard" proctoring system. The architecture is divided into several layers, designed using a three-tier client-server model that integrates with cloud services. This design effectively separates the user-facing components, the client from the backend, the server, which is a standard practice for building maintainable applications.

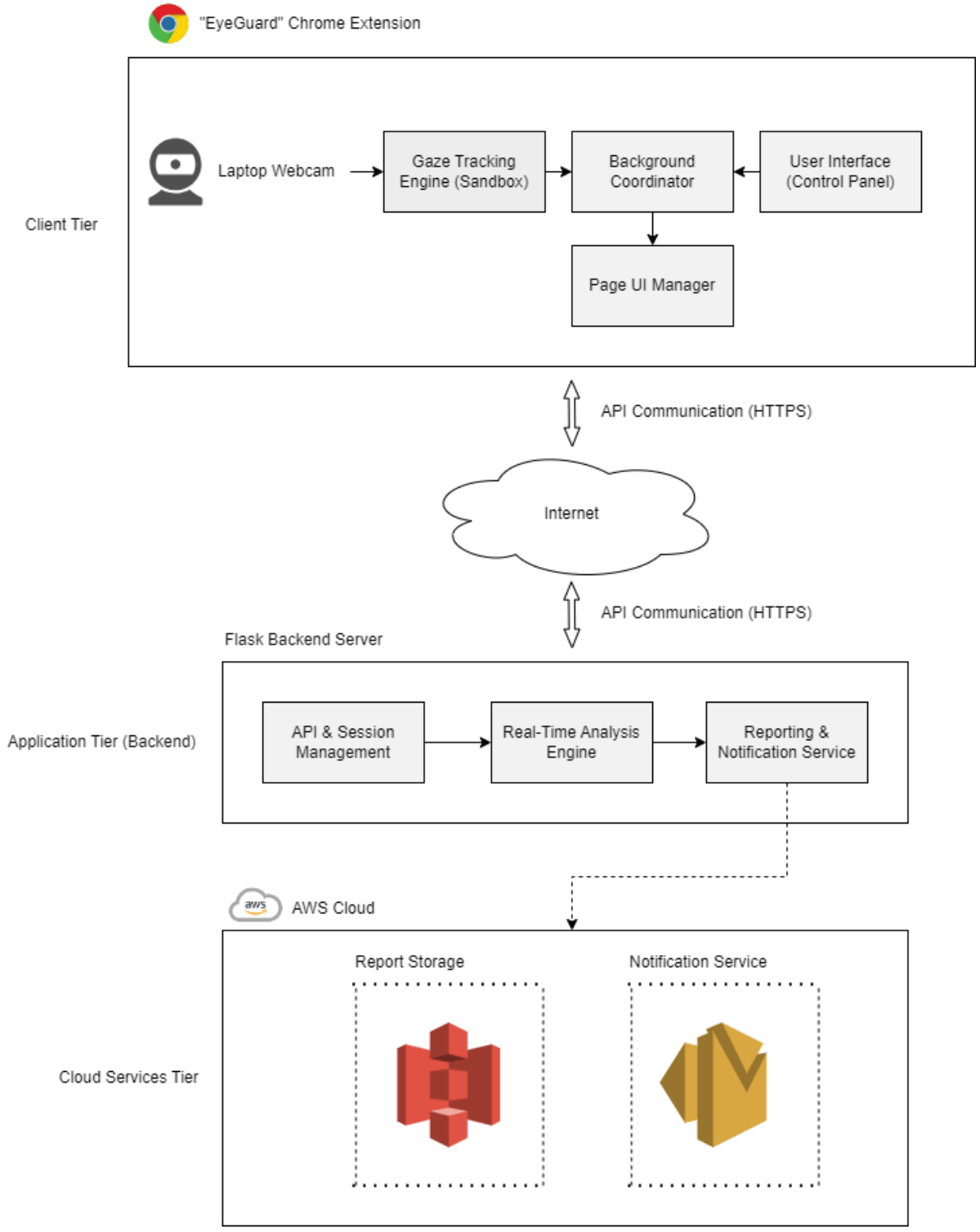


Figure 3.1: System Architecture of the EyeGuard Proctoring System.

Beginning with the topmost tier, the Client Tier represents all the components that run directly within the student web browser. This tier is encapsulated within the "EyeGuard" Chrome Extension. The extension is a collection of specialized components working together. The User Interface (Control Panel) works to allow the student to log in and interact with the proctoring session. Commands from the user are sent to the Background Coordinator, which acts as the central hub of the extension, manages the overall state of the session and acts as a listener for browser activities, such as tab switching. To perform the eye-tracking, the system relies on the Gaze Tracking Engine, which runs in a secure sandbox to safely access the laptop webcam and predict the user gaze coordinates. Finally, the Page UI Manager is responsible for all visual elements that are displayed on the student screen, such as the calibration dots and the real-time violation alerts when there are suspicious activities like gaze look outside the screen boundary or new tab open caught.

Communication between the client and the server happens over the Internet. As shown in the diagram, all data is exchanged via API Communication (HTTPS) channel. This ensures that all data, including gaze coordinates and student information, is protected during transmission.

For the purposes of development, the backend server is run locally on the same laptop. However, the architecture is designed to be ready for a production environment, as the client communicates with the server via standard HTTPS protocols.

Moving on to the middle tier, the Application Tier (Backend) is the central brain of the system. It is a Python Flask Server that waits for and responds to requests from the client. Inside the server, the API & Session Management module handling incoming requests and keeping track of active proctoring session. The data is then passed to the Real-Time Analysis Engine, which contains the custom algorithm designed. This is where the core logic of analyzing user behavior for suspicious patterns occurs. If a violation is confirmed, the analysis engine passes the result to the Reporting & Notification Service, which is responsible for generating and triggering alerts.

Lastly, the Cloud Services Tier handles tasks that require high reliability and scalability. Our backend communicates with Amazon Web Services (AWS) for these functions. The Notification Service uses AWS SES (Simple Email Service) to programmatically send email alerts to the administrator when a critical violation is detected. The Report Storage uses AWS

S3 (Simple Storage Service) to securely archive for the final HTML reports generated at the end of each session.

This three-tier architecture provides significant advantages for “EyeGuard” system. A key benefit is the complete separation of the backend logic from the user machine. This means that a single, dedicated machine can support a large number of users. All the intensive data analysis is performed on the server, ensuring that the student's computer only needs enough power to run the lightweight Chrome Extension.

3.2 Use Case Diagram

Figure 3.2: Use Case Diagram

Figure above is the use case diagram that shows the tasks that users can perform on the application and how the system acts. The use case descriptions for the tasks are discussed in the following.

3.2.1 Use Case Descriptions

Use Case ID	UC-01	Use Case Name	Login to System
Primary Actor	Student		
Brief Description	This use case allows the student to authenticate with the system using their credentials. Successful login is a must for all other proctoring functions.		
Trigger	The student attempts an action that requires authentication which “Calibrate Gaze" for the first time in a session.		
Precondition	The system requires the student to be authenticated to proceed.		
Scenario Name	Step	Action	
Main Flow	1	The system checks if the page is a valid page to run the system or not	
	2	The system shows the guidelines to assist the students get familiar with the system.	
	3	The system automatically opens the login window after students confirm the guideline.	
	4	The student enters their Student ID and Password.	
	5	The student clicks the "Login" button.	
	6	The system sends the credentials to the backend /login endpoint for verification.	
	7	The system validates the credentials and returns a successful message with the student's profile information.	
	8	The system stores the student information, and the login window closes automatically, allowing the student to proceed.	
Sub-Flow A - Invalid Credentials	7a.1	At step 7, if the backend determines the credentials are incorrect, it returns an error message.	
	7a.2	The system displays an "Invalid credentials" message to the student on the login page.	
	7a.3	The system returns to step 2 for the student to try again.	

Sub-Flow B - Invalid Page for Calibration	1a.1	At step 1, if the system detects the current tab is a browser system page like chrome://extensions and first page of chrome.
	1a.2	The system displays the error message: "Cannot run on browser system pages." and use case terminate
	1a.3	The student requires to try again in a valid page

Table 3.1: Login to System Use Case Description

Use Case ID	UC-02	Use Case Name	Perform Gaze Calibration
Primary Actor	Student		
Brief Description	The student follows an interactive process to train the eye-tracking model, personalizing it to their environment to ensure monitoring accuracy.		
Trigger	The student clicks the "Calibrate Gaze" button in the extension control panel		
Precondition	The student must be successfully logged into the system.		
Scenario Name	Step	Action	
Main Flow	1	The system verifies that the current browser tab is a valid webpage.	
	2	The system checks if the browser has permission to open the camera or not	
	3	The system injects the calibration UI overlay onto the webpage	
	4	The system displays a sequence of dots on the screen.	
	5	The student looks at and clicks each dot as instructed.	
	6	The system trains the WebGazer model with each click	
	7	The system removes the calibration of UI upon successful completion	
	8	The system internal state is updated to 'ready' state.	
Sub-Flow B: Invalid Page for Calibration	1a.1	At step 2, if the system detects the current tab is a browser system page like chrome://extensions and first page of chrome.	

	1a.2	The system displays the error message: "Cannot run on browser system pages." and use case terminate
	1a.3	The student requires to try again in a valid page
Sub-Flow B: Camera Permission Not Granted	2b.1	At step 2, if the system detects it does not have camera permission.
	2b.2	The system (via the browser) prompts the student to grant camera access.
	2b.3	If the student grants permission, the main flow continues from step 4.
	2b.4	If the student denies permission, the system displays an error message "Camera access is required for calibration." and the use case terminates.

Table 3.2: Perform gaze calibration description

Use Case ID	UC-03	Use Case Name	Start Proctoring Session
Primary Actor	Student		
Brief Description	Allow the student to begin the monitored exam session after all prerequisites are met. This action triggers the system to start actively monitoring the student.		
Trigger	The student clicks the "Start Proctoring" button in the extension control panel.		
Precondition	The system's internal state must be 'ready', indicating the student is logged in and calibration is complete.		
Scenario Name	Step	Action	
Main Flow	1	The student clicks the "Start Proctoring" button.	
	2	The system sends a start_test request to the backend server.	
	3	The system receives a unique session_id from the backend.	
	4	The system changes its internal state to 'running' state and begins monitoring events.	
	5	The system UI updates to show the session is in progress.	

Table 3.3: Start Proctoring Session Use Case Description

Use Case ID	UC-04	Use Case Name	End Proctoring Session
Primary Actor	Test-taker (Student)		
Brief Description	Allows the student to formally stop the monitoring session, which triggers the finalization and generation of the session report.		
Trigger	The student clicks the "End Proctoring" button in the extension control panel.		
Precondition	A proctoring session must be active and in the 'running' state		
Scenario Name	Step	Action	
Main Flow	1	The student clicks the "End Proctoring" button.	
	2	The system sends an end_test request to the backend server	
	3	The system executes the Generate Final Report use case	
	4	The system internal state is updated to 'ended' state.	
	5	The student has done with his/her examination	

Table 3.4: End Proctoring Session Use Case Description

Use Case ID	UC-05	Use Case Name	View Session Report
Primary Actor	Administrator		
Brief Description	Allows the administrator to access and review the detailed analysis of a completed proctoring session by logging into the designated AWS S3 bucket where reports are stored.		
Trigger	The administrator accesses the AWS Management Console login portal to begin the process of viewing a report.		
Precondition	The administrator has been provided with AWS IAM credentials. The End Proctoring Session use case must have successfully uploaded the report to the S3 bucket.		
Scenario Name	Step	Action	
Main Flow	1	The administrator performs the login to AWS Console use case	
	2	The administrator navigates to the S3 (Simple Storage Service) dashboard.	
	3	The administrator locates and opens the specific S3 bucket designated for EyeGuard reports, which eproctor-reports	

	4	The administrator browses the bucket and locates the report file for the desired session_id
	5	The administrator opens the file to review the integrity score, event timeline, and gaze plot.
	6	The administrator may conduct post exam investigation towards students who are caught suspicious

Table 3.5: View session report use case description

Use Case ID	UC-06	Use Case Name	Display On-Screen Warning
Primary Actor	System		
Brief Description	When the system detects a minor violation, it displays a real-time warning to the student. This is an "extend" of the Monitor User Behavior use case.		
Trigger	The Monitor User Behavior use case detects violation or suspecious behavior of student during test.		
Precondition	A proctoring session is active. Status is "running"		
Scenario Name	Step	Action	
Main Flow	1	The system analysis engine identifies a violation or suspecious behavior of student during test and the API response includes an alert message.	
	2	The system displays a pop-up on the student screen, saying that what the actions they done is violent letting them to stay focus on the test	
	3	The system logs the event and will be display in the report	

Table 3.6: Display on-screen warning use case description

Use Case ID	UC-07	Use Case Name	Handle Critical Violation
Primary Actor	System		
Brief Description	When the system detects a major violation, it executes a two-part alert to notify both the student and the administrator. This is an "extend" of the Monitor User Behavior use case.		
Trigger	The Monitor User Behavior use case detects a critical violation event which the students receive alert more than 3 times.		
Precondition	A proctoring session is active. Status is "running"		
Scenario Name	Step	Action	
Main Flow	1	The system confirms a critical violation	
	2	The system display on-screen critical alert	
	3	The system displays a pop-up on the student's screen, saying that their actions are violent, further investigation will be conducted to them.	
	4	The system sends email critical alerts to administrator	
	5	The system logs the event and will be displayed in the report	

Table 3.7: Handle Critical Violation Warning Use Case Description

3.3 Activity Diagram

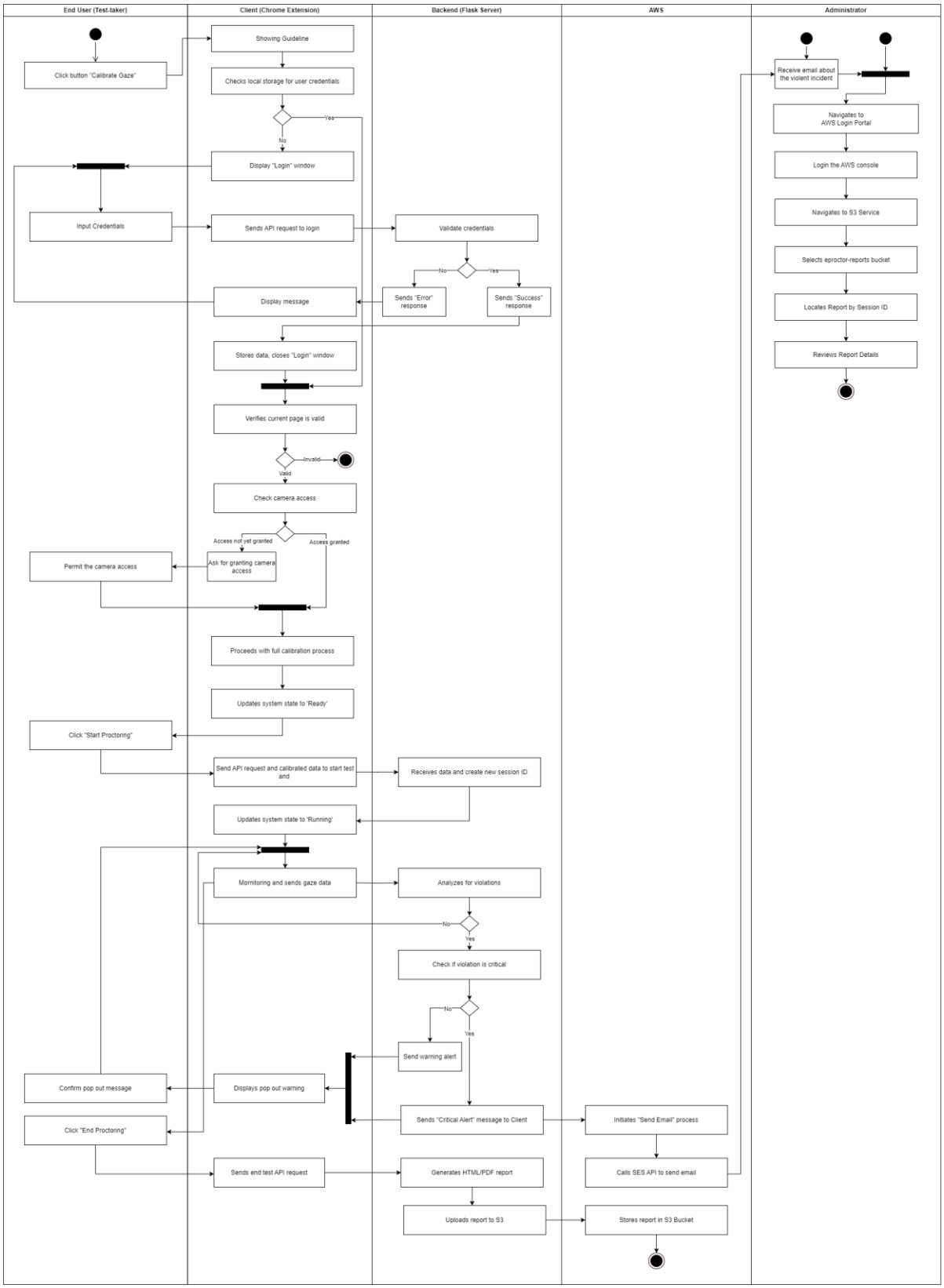


Figure 3.3: Activity Diagram

The activity diagram in Figure 3.3 provides a comprehensive, end-to-end illustration of the entire EyeGuard proctoring lifecycle, from initial student setup to final administrator review. The diagram utilizes swimlanes to clearly delineate the responsibilities and interactions between the Test-taker (Student), the Client (Chrome Extension), the Backend (Flask Server), integrated AWS cloud services, and the Administrator.

The entire workflow is initiated by the Student attempting to prepare for the exam by clicking the "Calibrate Gaze" button. At this point, the Client (Chrome Extension), which manages all browser-side logic, first show the guideline to the user to guide them using the system, then checks its local storage to determine if the user is already authenticated upon confirmation. If the user is not logged in, the Client displays a login window. The student interaction starts with providing their credentials. The Client sends these credentials to the Backend (Flask Server) for validation. If the credentials be invalid, the Backend returns an error, and the Client prompts the student to try again, creating a loop until a successful login occurs. Once authenticated, or if the student was already logged in before, the flow proceeds to the calibration phase, which is handled entirely by the Client as well. This involves verifying the webpage is valid, checking for and requesting camera access from the Student, and guiding the user through the interactive calibration process of clicking dots on the screen. The outcome of this phase is the Client's internal state being updated to 'Ready', indicate that the system is prepared to begin monitoring.

The monitoring starts when the Student clicks "Start Proctoring." This action prompts the Client to send an API request to the Backend, which to creates a unique session ID for the session. The Client's user interface updates to a 'Running' state, signaling to the student that monitoring is now active. This initiates the core monitoring of the system, the Client continuously captures and sends a stream of gaze and browser event data to the Backend for real-time analysis.

The Backend is responsible for all violation detection and handling logic. As it receives data, it first determines if any activity is a violation. If no violation is found, the monitoring simply just continues. If a suspicious activity is detected, decision is made regarding its severity. For minor violations, the Backend sends a standard warning message back to the Client, which is then displayed to the Student as a warning on-screen pop-up. For critical violations, the backend simultaneously sends a critical alert message to the Client for display to the Student, saying that they will be investigated after the test, while also making an API

call to the AWS swimlane. This call triggers the Amazon SES (Simple Email Service) to send a detailed notification email directly to the Administrator with registered email.

The session concludes when the Student clicks the "End Proctoring" button. The Client sends a final API request to the Backend, which then generates the final report files in both HTML and PDF formats. In the final interaction with the cloud, the Backend uploads these reports to the AWS swimlane, where they are securely stored in the designated S3 bucket. The administrator who prompted by an email alert, then takes action that entirely outside the EyeGuard system, as they log into the external AWS Console to locate and review the session report, to take necessary action.

CHAPTER 4 SYSTEM METHODOLOGY/APPROACH

This chapter details the methodology, technical requirements, and core algorithms that guided the development of the "EyeGuard" system. The following sections will explain the rationale for adopting an Agile development process, specify the hardware and software requirements for the project, present the detailed project timeline, and provide a technical breakdown of the key detection and scoring algorithms.

4.1 Agile Development Methodology

For this project, the agile methodology was chosen as the main development process instead of more traditional methods like Waterfall. The Waterfall approach is very rigid and requires each step to be finished before the next can start, which is suitable for large, complex systems where the requirements are already perfectly clear. For a Final Year Project like this, which is smaller and involves a lot of experimentation, a waterfall approach would likely create unnecessary delays and make it hard to adapt when things change.

Agile encourages flexibility and fast iteration, which was very useful for this “Detecting Online Test Cheating Through User Behaviour Monitoring” project. The nature of this project involves designing and implementing AI models, such as the real-time eye-gaze tracking system, which requires a lot of trial and error to get working correctly. Agile allows for this kind of work by letting you build a small part of the model, try it out, see how it works, and if it is not good enough, repeat the cycle with a different strategy without needing a heavy project plan from the start.

Figure 4.1: Agile Methodology in System Development

In Agile, development is divided into series of sprints and as referred from the diagram, listed in planning (requirement analysis), designing, development, testing, deployment and review. Each sprint produces at least a partial prototype, allowing the work to be able to evaluate again if required and further development. [9][13] The development process looked something like this:

1. **Initial Sprints:**

   The first few sprints were focused on building the project's foundation. This meant setting up the basic Flask server on the backend and creating the core Chrome Extension framework with its popup and background scripts. The goal was simply to get the client and server connect and communicate with each other.

2. **Mid-Project Sprints:**

   Once the foundation was stable, the next few sprints tackled the most important feature of eye-tracking. WebGazer is implementing to secure sandbox environment and set up the on-

screen calibration interface. By the end of these sprints, system could successfully track a user gaze.

3. **Later Sprints:**

With gaze data being collected, the next sprints focused on the backend logic. This is where the the gaze analyzes and violation algorithms, “SmartGazeMonitor” was developed to analyze the data and intelligently filter out false positives. After that, dual-channel real-time alerting system built, integrating the on-screen alert for students and the AWS SNS notifications for administrators.

4. **Final Sprints:**

The last sprints brought everything together by creating the final reporting features, including the automatic generation of HTML and PDF reports and their storage in AWS S3. Also, others feature like the login module and guideline module are built upon these.

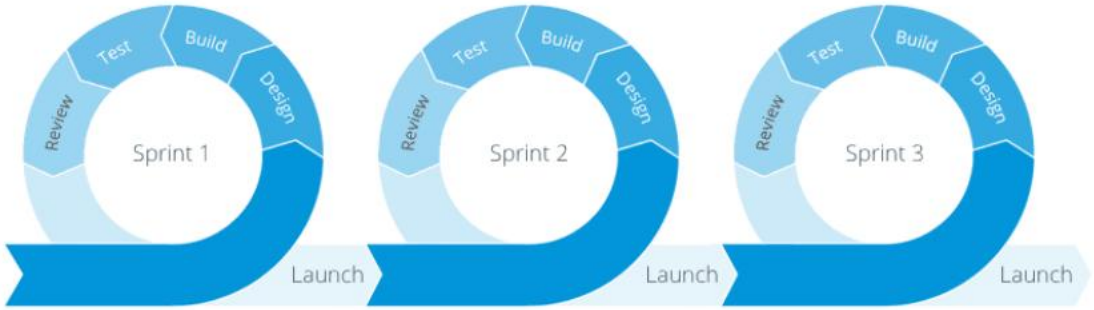


Figure 4.2: Agile Development Lifecycle with Iterative Nature

This incremental approach also made the project very responsive to change. For instance, during testing, we might find that the gaze-tracking algorithm is producing too many false alerts. Because of the Agile process, it was easy to dedicate the next sprint to refining the module parameters and improving its logic, using the foundation we had already built. In short, the agile methodology was a great fit for this project, as it enabled quick prototyping and experimentation.

4.2 System Requirement

This section details the necessary hardware and software specifications required for the development, testing, and operation of the "EyeGuard" system.

4.2.1 Hardware Requirements

The project was developed and tested on a single machine that served as both the client and the server. The specifications for this development environment are listed below, followed by the minimum requirements for an end-user.

Component	Specification
Model	Lenovo IdeaPad 5
Processor	12th Gen Intel(R) Core (TM) i7-1255U @ 1.70Ghz
Memory (RAM)	16.0 GB
Graphics	NVIDIA GeForce MX550
Storage	474 GB SSD
Note	This machine served as both the client (running the Chrome Extension) and the server (running the Flask backend) for the project. Communication between the two components was handled locally via HTTPS.

Table 4.1 Development and Testing Environment

Component	Minimum Specification	Purpose
Computer	A standard laptop or desktop	To run the Chrome browser and the exam interface.
Webcam	A functional, integrated or external webcam	Essential for capturing the video feed for gaze tracking.
Internet	A stable connection	For reliable communication with the backend server

Table 4.2 End-User Minimum Requirements

4.2.2 Software Requirements

The following software components and tools were used to build and test the "EyeGuard" system.

Category	Component / Tool	Purpose
Operating System	Windows 11 Home (64-bit)	The primary OS for development and testing.
Development Tools	Visual Studio Code	The primary code editor for all project files.
	Google Chrome	For developing, debugging, and running the extension.
	Git & GitHub	For version control and source code management.
Backend	Python	The programming language for the server.
	Flask	The web framework for creating the API.
	Boto3	AWS SDK for Python, used for S3 and SES integration.
Frontend	JavaScript	The programming language for the Chrome Extension.
	WebGazer.js	Core library for real-time gaze tracking.
	Chart.js	Library used for data visualization in the final report.
Cloud Services	Amazon S3	For secure storage of generated session reports.
	Amazon SES	For sending automated critical violation email alerts.

Table 4.3 Software Components and Tools

4.2.2.1 Development Platform and Tools

4.2.2.1.1 Visual Studio Code

Visual Studio Code is a lightweight but powerful source code editor developed by Microsoft. It supports a wide array of programming languages, including Python and JavaScript, through its extensive library of extensions. It was used as the primary integrated development environment (IDE) for writing, managing, and debugging all frontend and backend code for the EyeGuard system.

4.2.2.1.2 Google Chrome & Extensions API

Google Chrome serves as the runtime environment for the client-side of the EyeGuard system. The project is built as a Chrome Extension using the Manifest V3 platform. This framework provides the necessary Application Programming Interfaces (APIs) to inject scripts into web pages, run persistent background processes, and monitor browser-level events like tab switching and window focus changes, which are essential for proctoring.

4.2.2.1.3 Python & Flask

Python is a high-level, interpreted programming language chosen for the backend due to its simplicity and powerful data processing libraries. Flask, a lightweight and flexible web framework for Python, was used to build the RESTful API for the EyeGuard system. This API is responsible for handling all communication from the client, processing proctoring data, and managing user sessions.

4.2.2.1.4 WebGazer.js

WebGazer.js is an open-source JavaScript library that enables real-time eye tracking directly in the browser without needing specialized hardware. It is the core component for the system's primary cheating detection feature. In this project, it is run within a secure sandbox to access the user's webcam and train a personalized regression model that predicts the user's gaze location on the screen.

4.2.2.1.5 Amazon Web Services (AWS) & Boto3

Amazon Web Services is a comprehensive cloud computing platform. To interact with its services programmatically, the backend uses Boto3, the official AWS SDK (Software Development Kit) for Python. Boto3 was used to integrate two key AWS services: Amazon S3 for the secure and scalable storage of final session reports, and Amazon SES (Simple Email Service) for sending automated critical violation email alerts to administrators.

4.2.2.1.6 Chart.js

Chart.js is a popular open-source JavaScript library for creating responsive and animated charts. It was used in the report.html frontend to create the 2D scatter plot that visually represents the student's gaze data, providing administrators with an intuitive way to analyze where the student was looking during the session.

4.2 Timeline

4.2.1 Timeline of FYP1

Week	Duration (week)	Task
1	1	Requirement Gathering
2	1	Project feasibility testing
3	7	Develop prototype of system
9	3	Report writing
10	1	Poster creation
11	2	Report finalization
13	1	Presentation preparation

Table 4.4 Timeline of FYP1

As a recap that this "Detecting Online Test Cheating Through User Behaviour Monitoring" project is practicing Agile approach, which indicates that flexibility and iterative development is emphasized. Thus, planning and implementation will change from time to time throughout the project timeline with frequent updates or modifications. The timeline showing is according to major phases, with the respective task and duration.

The main goal of FYP1 is to develop an initial prototype and serve as foundation for this whole project. First phase is starting with requirement collection, where all the materials like the dataset for eye gaze tracking model and algorithms are collected and verified. The dataset will then be labelled so that it able to be trained with YOLO then, which the eye positions are labelled with bounding work for training the model. Also, the feasibility analysis is carried out to make sure the dataset, algorithms, and methodologies for YOLO model training are suitable for the model training for eye gaze tracking.

Moving forward after the dataset is prepared with labelling, model training begins with using Google Colab as tool due to the limited computing power of laptop without a powerful GPU, thus Google Colab is used to provide more computing power. The model selection then conducted due to different versions of YOLOv4, YOLOv5, YOLOv11 and so on each having different performance in the way that the newest model does not indicate that it is the best and thus testing and verification is required to find out which performs best in terms of accuracy and speed for the real time eye gaze detection.

Also, the prototype will contain a simple alerting system in which the system will notify when there is suspicious behaviour, such as when a student has unusual eye gaze. This is important to serve as foundation for the alerting system that is going to be developed in FYP2.

4.2.2 Timeline of FYP2

Week / Progress	1	2	3	4	5	6	7	8	9	10	11	12	13
Project Preparation & Environment Setup													
Develop Client-Side Gaze Calibration													
Build Basic Backend API Endpoints													
Integrate Client with Backend (Session Start/End)													
Develop Backend Violation Logic (SmartGazeMonitor)													
Implement Real-time Alerting System													
Integrate AWS & Reporting Feature													
End-to-End System Testing & Debugging													
Final Report Writing & Presentation													

Table 4.5 Timeline of FYP2

As shown in the timeline in Table 4.5, the 13-week schedule for FYP 2 is structured to build the "EyeGuard" system. The process begins in Week 1 with Project Preparation & Environment Setup, which includes configuring the Flask backend and the Chrome Extension framework, ensuring they are able to communicate with each other. Following this, a two-week period (Weeks 2-3) is allocated to developing the core client-side feature which the Gaze

Calibration UI, in Week 4, is to start the construction of the basic backend API endpoints. Moving on in Week 5 is to focus on connecting the client and backend for gaze point receiving. A significant two-week block (Weeks 6-7) is dedicated to developing the backend analyze and violation logic which the “SmartGazeMonitor” algorithm. This is followed by the implementation of the real-time alerting system in Week 8. The final development push occurs over Weeks 9 and 10, where the AWS cloud services (S3 and SES) are integrated and the final reporting feature with Chart.js visualization is built. Week 11 is reserved for comprehensive end-to-end system testing and debugging. The project concludes with the final two weeks (12-13) dedicated to writing the final report and preparing for the presentation.

4.3 Core Algorithms and Detection Logic

The effectiveness of the EyeGuard system is derived not from a single technology, the WebGazer, but also from a collection of interconnected algorithms designed to intelligently interpret raw user data, the gaze point. This section details the key custom algorithms that form the analytical engine for the violation eye gaze of the backend server, the “SmartGazeMonitor” Temporal Filter, and the Integrity Scoring model. Together, these algorithms transform the system from a simple monitor into a proctoring tool that prioritizes accuracy and fairness**.**

4.3.1 The Gaze Boundary Polygon Algorithm

The Gaze Boundary Polygon algorithm creates from the screen calibration process and produce a personalized and accurate "safe zone" for on-screen gazing by using the calibration function built into WebGazer. Its purpose is to define the precise boundaries of the user's screen for the duration of the monitoring session.

The process works in three distinct steps:

1. **Client-Side Calibration (Data Collection):**

   The process begins when the user is prompted to look at and click a series of dots on their screen from the chrome extension. For each click, the system record the screen position function. This action trains WebGazer's internal regression model, creating a personalized map between the user's specific eye features and the (x, y) coordinates of their screen.

2. **Data Transfer to Backend:**

   Once calibration is complete and the proctoring session begins, this list of collected gaze coordinates is sent to the backend server.

3. **Backend Polygon Creation:**

The backend algorithm receives the array of gaze coordinates. It then works to find the minimum and maximum values for both the x-axis and the y-axis. These four values which min_x, max_x, min_y, max_y are used to define the corners of a rectangle. This rectangle is then stored for that specific session.

This algorithm is important for the system accuracy. By using the data from the user own calibration, it creates a boundary that specific screen size. All subsequent gaze data is then checked against this custom boundary.

4.3.1.1 Comparison with Object Detection (YOLO)

This method is fundamentally different from proctoring approaches that might use an object detection model like YOLO. The table below outlines the key distinctions:

Feature	EyeGuard's Regression Approach	Alternative Classification Approach (YOLO)
Task Type	Localization & Regression. It calculates the precise (x, y) coordinates of the gaze and checks if the point is inside a defined area.	Classification. It classifies the student's eye orientation into categories like "looking left," "center," or "looking up".
Output	A continuous stream of precise (x, y) gaze coordinates.	A discrete class label and a confidence score for that prediction.
Precision	High. It can detect precise points (x, y) coordinates for fine-tuned analysis.	Medium. It cannot distinguish between looking at the edge of the screen versus looking at the point beyond the screen.
Personalization	High. The boundary is custom defined for every user and session, adapting to their specific screen size and posture.	Low. A pre-trained model is generic and does not adapt to user setups.

Table 4.6: Comparison of Regression Approach and Classification Approach

4.3.2 The Temporal Filtering Algorithm

This algorithm is designed to against the false positives, adding a layer of analysis to the raw gaze data.

This is due to raw gaze data is actually "noisy" data. Innocent actions like blinking or quick eye glance can cause the gaze point to briefly fall outside the boundary. Thus, the eye gaze analyze and violation algorithm, “SmartGazeMonitor” solves this by using a temporal filter. It maintains a sliding window of the last 2 seconds of gaze data in a fixed-size queue. A violation is only confirmed if a high density of "outside" gaze points accumulates within that short time frame, which is around 2 seconds, with 7 points captured. This logic effectively ignores isolated, outlier data points which are believed to be only natural glances while correctly identifying patterns of off-screen gazing that are suspicious for cheating. Without this temporal filter, the system would be functionally unusable, as it would generate an lots of false positives alerts triggered by innocent behavior.

Figure 4.3: The “SmartGazeMonitor” Temporal Window Illustration

It is important to acknowledge that this filtering mechanism introduces a trade-off. By increasing the threshold for what constitutes a violation, the system may increase the risk of false negatives, where genuine cheating attempt might not be flagged. However, this is a calculated decision that aligns with the core objective of the project: to reduce the administrator's workload by providing high-confidence alerts. An unfiltered system that constantly flags innocent actions would overwhelm administrators with meaningless data to review and cause significant frustration for students. Therefore, the “SmartGazeMonitor” algorithm as discussed is designed to find a crucial balance. It prioritizes the drastic reduction of false positives, accepting a minimal risk of false negatives in return. This approach ensures that when an alert is generated, it represents a sustained, credible event worthy of review, thereby respecting the time of the administrator and the integrity of the student's experience. In the meantime, ensure the examination integrity as well.

4.3.3 The Integrity Scoring and Risk Assessment Algorithm

This algorithm combines all confirmed violations into a single, easy-to-understand metric, providing administrators with a high-level summary of the session's integrity.

Each proctoring session begins with an Integrity Score of 100. When the "SmartGazeMonitor" algorithms or the browser event algorithm confirms a violation, a predefined number of points are deducted. The penalty varies by the severity of the action:

Violation Event	Penalty
tab_switch	15 points
new_tab_opened	15 points
window_blur	10 points
Confirmed Gaze Violation	10 points
proctoring_tab_closed	25 points

Table 4.7: Integrity Scoring for each Event

At the end of the session, the final Integrity Score is mapped to a qualitative Risk Level to guide the administrator's review process, to have a direct assessment on how the session overall severity be like.

Integrity Score	Risk Level
90 - 100	MINIMAL
75 - 89	LOW
60 - 74	MODERATE
40 - 59	HIGH
0 - 39	CRITICAL

Table 4.8: Integrity Scoring with Corresponding Risk Level

CHAPTER 5 SYSTEM IMPLEMENTATION

This chapter provides a detailed technical walkthrough of the "EyeGuard" system implementation, demonstrating how the architectural designs from the previous chapter were translated into a functional application. The following sections will detail the construction of both the Python Flask backend, the frontend Chrome Extension, as well as the AWS integration

5.1 Backend Setup (Flask Server)

The entire E-Proctor Advanced system is supported by a robust backend server, developed using the Flask micro-framework, functioning as the central nervous system for the application. Its primary responsibilities include handling incoming data from the frontend, processing user authentication, managing sessions in real-time, analyzing the collected data upon session completion as well as connecting SES and S3 services with desire functions. The setup of this backend is detailed as below.

5.1.1 Environment File

The AWS configuration is managed through a .env file, separating configuration from code is a crucial practice for enhancing security and maintainability. This is particularly important when dealing with sensitive information such as API keys and credentials. This file stores key-value pairs that are loaded into the application's environment at runtime, ensuring that confidential data is not exposed directly in source code.

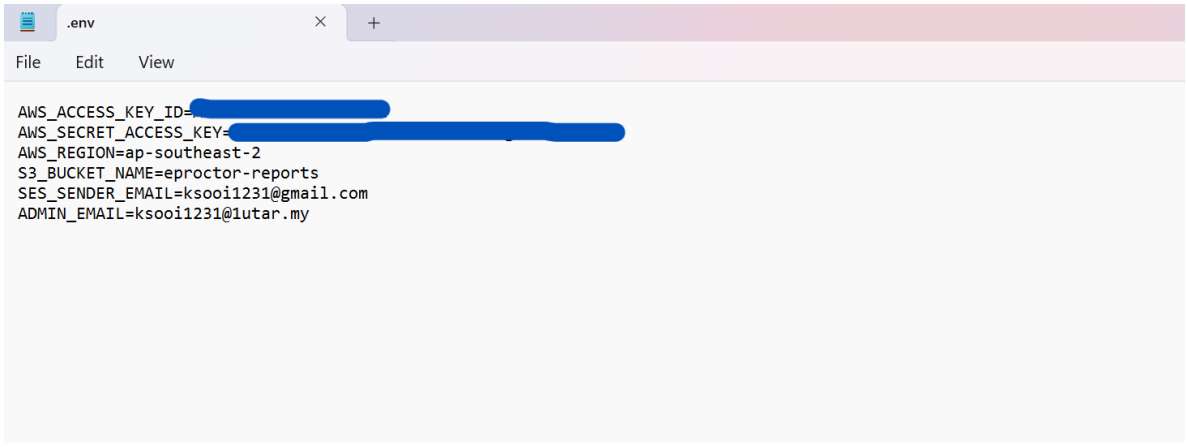


Figure 5.1: env. File

Remark: The access key and secret key which are mostly confidential are erased

5.1.2 Dependency Management

A Python project often relies on a set of external libraries to function as intended. Managing these dependencies is essential for ensuring that the application can be reliably deployed without errors. All external Python library dependencies are explicitly defined in a requirements.txt file. This includes Flask for the core web framework, boto3 as the official AWS SDK for Python, and WeasyPrint for PDF generation. This practice facilitates seamless project setup for developers, which can be installed efficiently using the command pip install -r requirements.txt

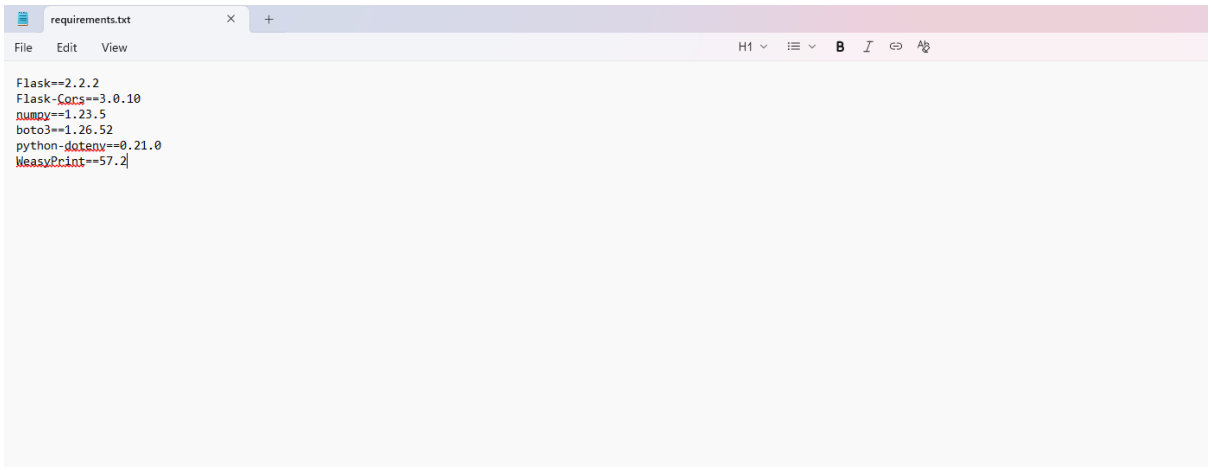


Figure 5.2: requirement.txt File

5.1.3 Backend Execution

The application is started by running the main_app.py script with command python main_app.py. The server then begins listening for incoming HTTP requests on the local host at port 5000, making it ready to communicate with the frontend Chrome extension

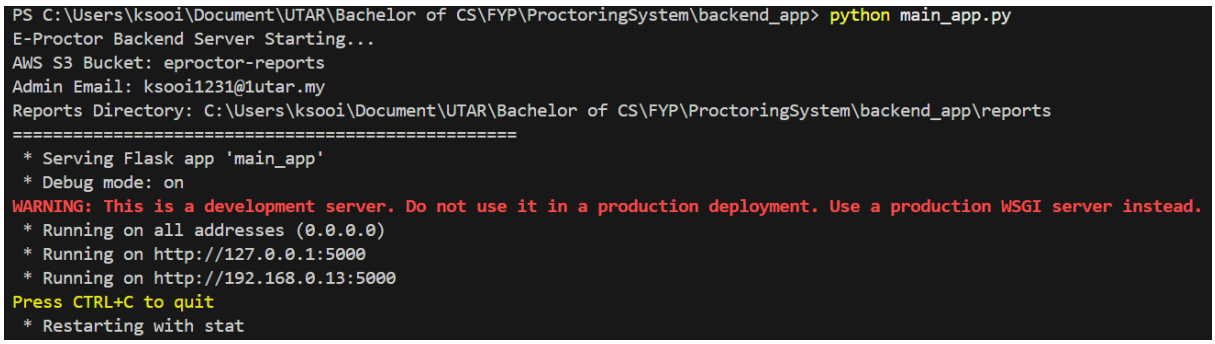


Figure 5.3: Running a Backend with Command “python main_app.py”

5.2 Frontend Implementation

The frontend of the E-Proctor Advanced system is a Google Chrome extension, which serves as the user-facing client. It is responsible for capturing user gaze, and browser activities, monitoring the browser environment, and communicating with the backend server.

5.2.1 Loading the Extension for Development

During the development phase, the extension was loaded into the browser in an 'unpacked' state. This is achieved by enabling 'Developer mode' on Chrome extension management page (chrome://extensions), selecting the designated folder and confirming the action. This allows for immediate testing of code modifications where any changes made to the files can be applied simply by reloading the extension.

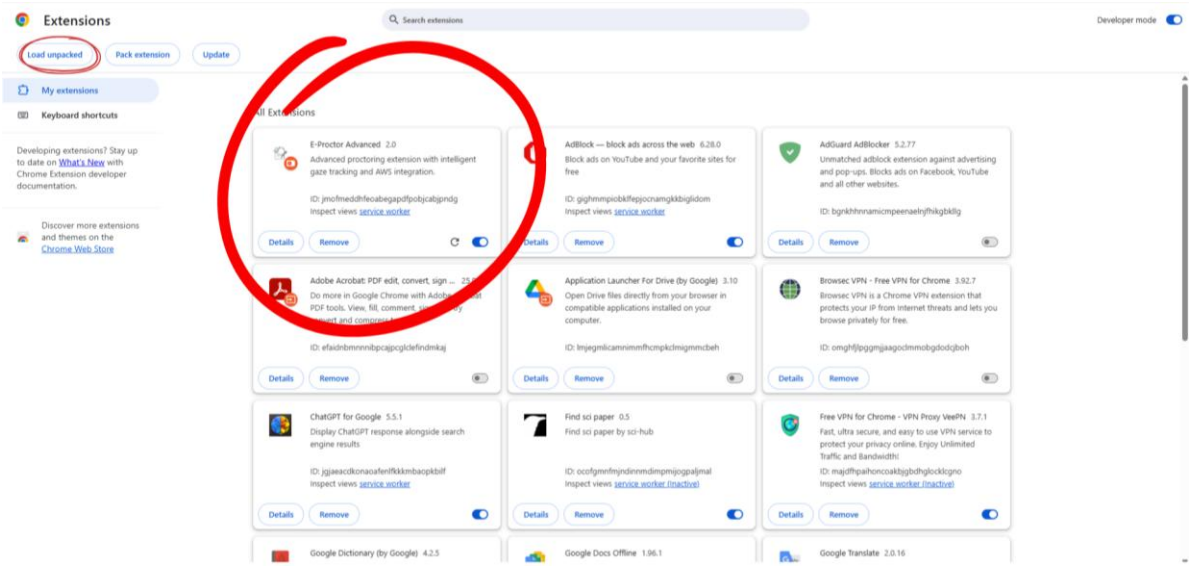


Figure 5.4: Files are Loaded to Chrome Extension through Developer Mode

5.3 Backend Implementation (Python Flask Server)

5.3.1 User Authentication Module

The system starts with the student authentication process, managed by the /login endpoint. The primary purpose of this module is to simulate a real-world login and create a unique user identity for each proctoring session. This allows the system to correctly associate all collected gaze coordinates, tab switches, and other events with a specific student.

The current implementation is a mock login, where credentials are validated against a pre-defined dictionary within the server code (main_app.py). This approach was chosen to simplify the development process, allowing the project to focus on the core functionalities rather than on complex user management systems. Upon successful validation, the server sends a success token and students were permitted to proceed.

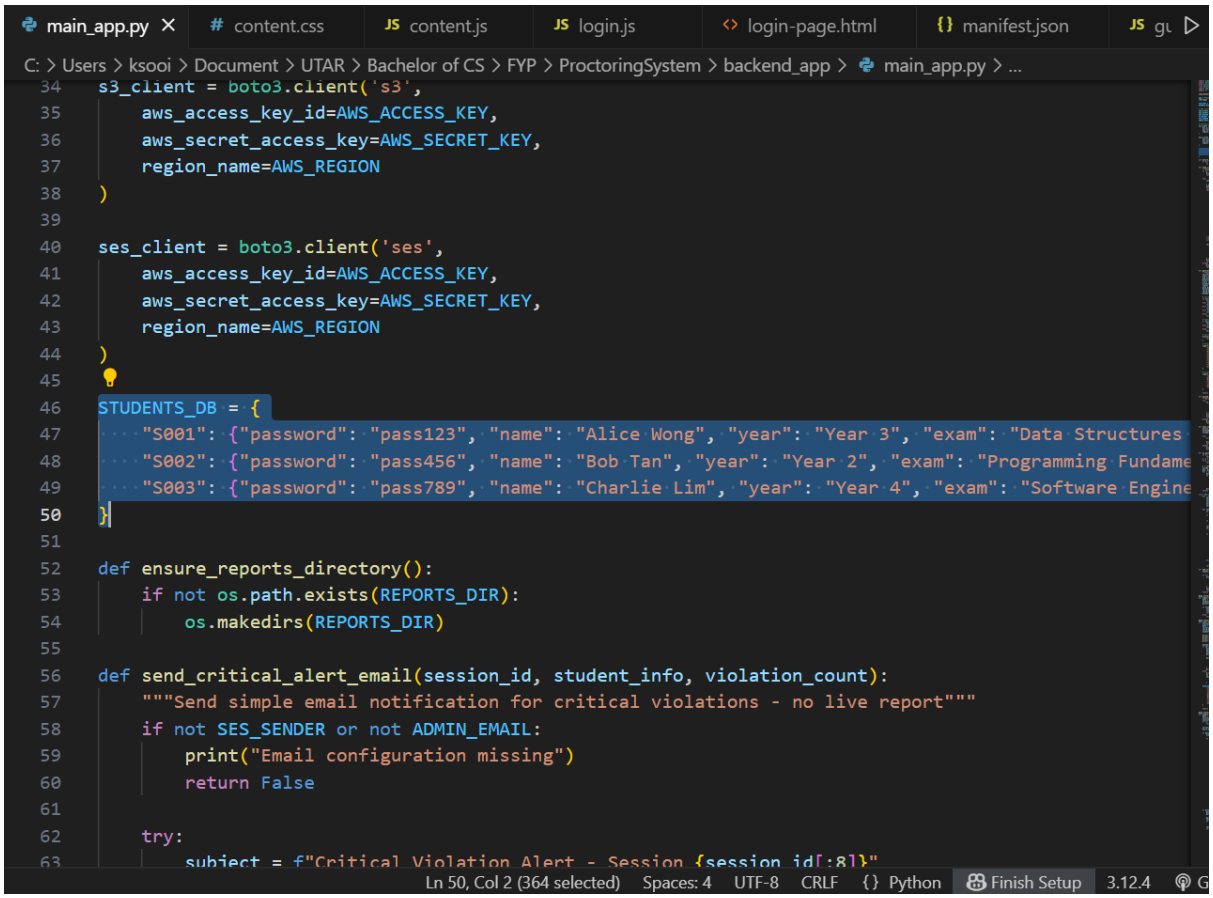


Figure 5.5: Mock Student Info that Stored in Backend

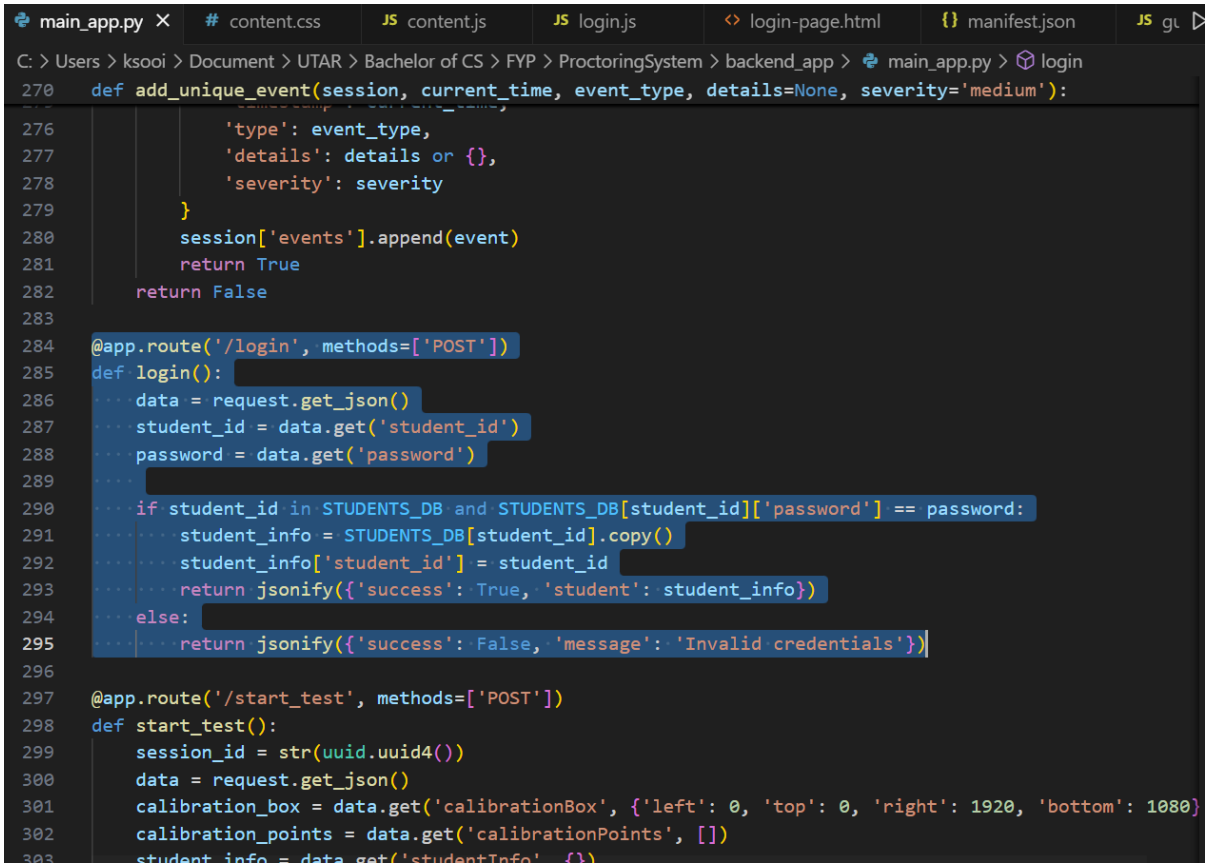


Figure 5.6: Login Module

5.3.2 Session and Event Processing Modules

The backend is handled by the session and event modules, which work to manage the entire system lifecycle from initiation to final reporting. The process begins when a student clicks the "Start Proctoring" button in the extension, triggering a call to the /start_test endpoint. This action creates a new, isolated data container for the session, with a unique session ID, which is then returned to the client, the chrome extension. Upon this, the /submit_data endpoint functions as a real-time data logger, continuously receiving stream of JSON data from the extension that includes event like gaze coordinates and browser activities which are violent. This data collection phase will be continued until the student clicks "End Proctoring," which system status will be change again from “running” to “ended” and sends a final request to the /end_test endpoint.

The /end_test triggers the analysis and thanks to the WebGazer, the computationally heavy machine learning for gaze prediction is performed in the browser by WebGazer, so the backend only receives lightweight coordinate data rather than processing raw video streams. Upon session completion, the analysis algorithm processes the raw data, together with all

flagged browser events, and formulate a cumulative Risk Score. Once this analysis is complete, the final structured report is generated with relative student information, this will then stored in the S3 bucket that we will discuss later.

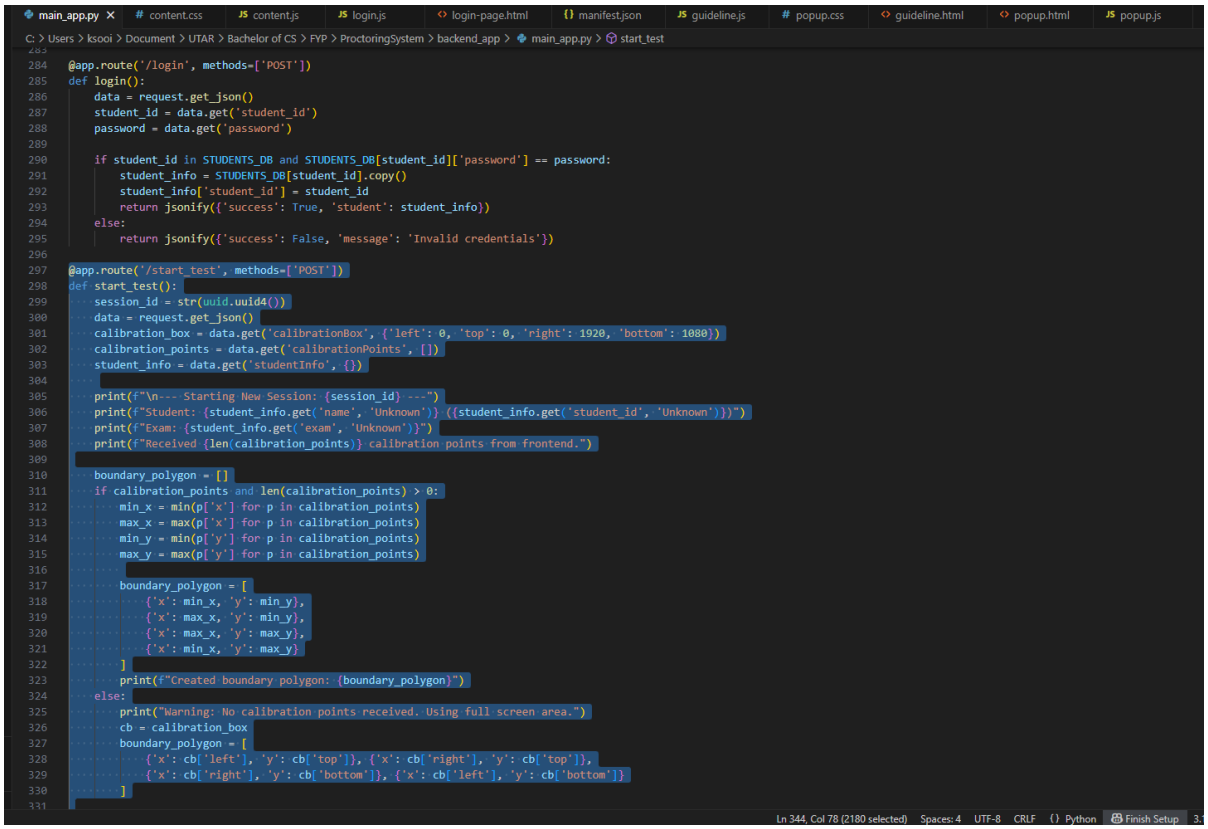
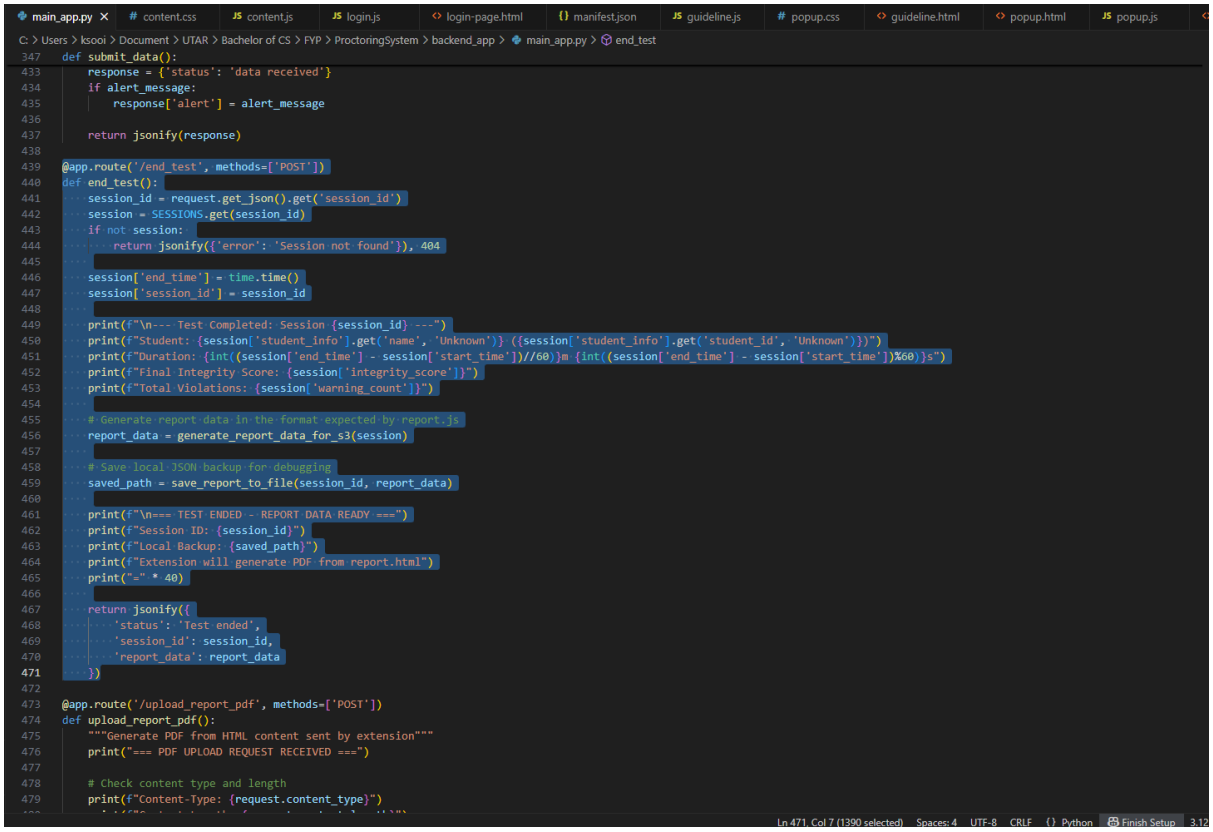
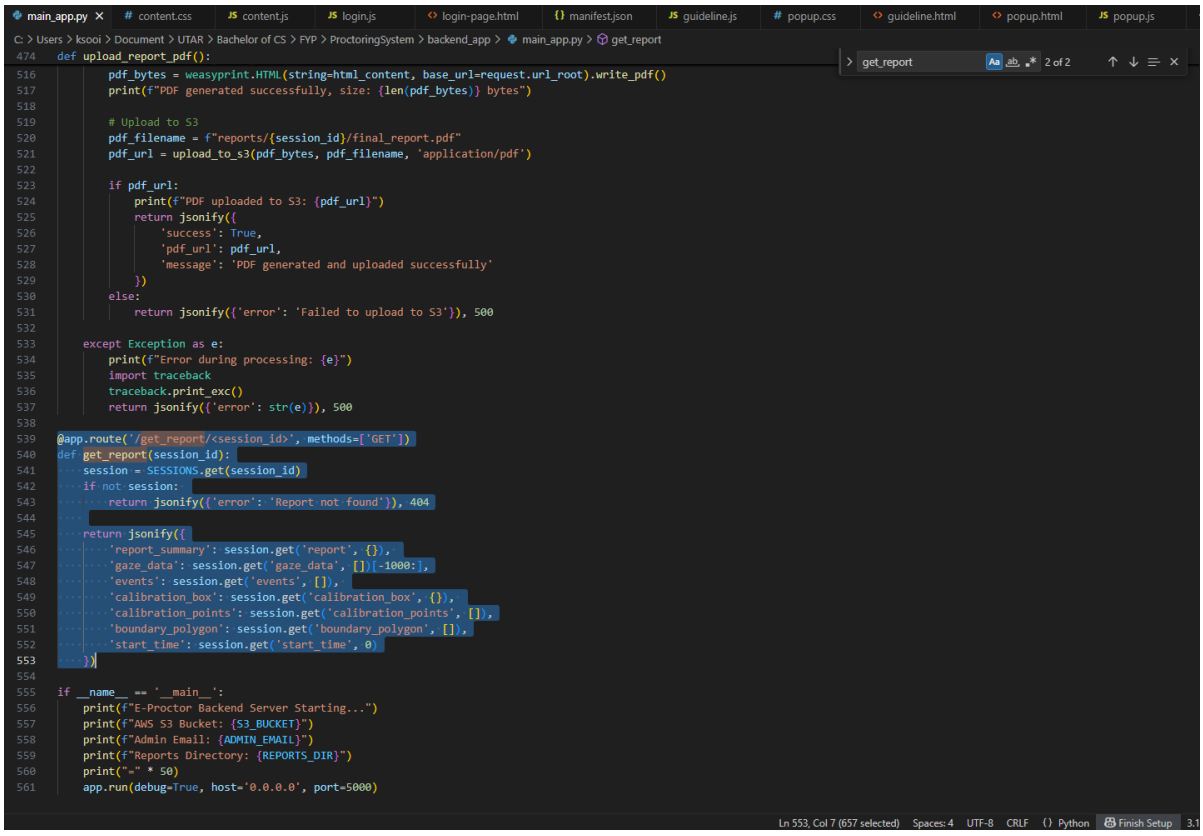


Figure 5.7: start_test() Session, Triggered when User Start the Test



```python
def submit_data():
    response = {'status': 'data received'}
    if alert_message:
        response['alert'] = alert_message

    return jsonify(response)

@app.route('/end_test', methods=['POST'])
def end_test():
    session_id = request.get_json().get('session_id')
    session = SESSIONS.get(session_id)
    if not session:
        return jsonify({'error': 'Session not found'}), 404

    session['end_time'] = time.time()
    session['session_id'] = session_id

    print(f"\n--- Test Completed: Session {session_id} ---")
    print(f"Student: {session['student_info'].get('name', 'Unknown')} ({session['student_info'].get('student_id', 'Unknown')})")
    print(f"Duration: {int((session['end_time'] - session['start_time'])//60)}m {int((session['end_time'] - session['start_time'])%60)}s")
    print(f"Final Integrity Score: {session['integrity_score']}")
    print(f"Total Violations: {session['warning_count']}")

    # Generate report data in the format expected by report.js
    report_data = generate_report_data_for_s3(session)

    # Save local JSON backup for debugging
    saved_path = save_report_to_file(session_id, report_data)

    print(f"\n=== TEST ENDED - REPORT DATA READY ===")
    print(f"Session ID: {session_id}")
    print(f"Local Backup: {saved_path}")
    print(f"Extension will generate PDF from report.html")
    print("=" * 40)

    return jsonify({
        'status': 'Test ended',
        'session_id': session_id,
        'report_data': report_data
    })

@app.route('/upload_report_pdf', methods=['POST'])
def upload_report_pdf():
    """Generate PDF from HTML content sent by extension"""
    print("=== PDF UPLOAD REQUEST RECEIVED ===")

    # Check content type and length
    print(f"Content-Type: {request.content_type}")
```

Figure 5.8: end_test session, Triggered when User End the Test



```python
def upload_report_pdf():
        pdf_bytes = weasyprint.HTML(string=html_content, base_url=request.url_root).write_pdf()
        print(f"PDF generated successfully, size: {len(pdf_bytes)} bytes")

        # Upload to S3
        pdf_filename = f"reports/{session_id}/final_report.pdf"
        pdf_url = upload_to_s3(pdf_bytes, pdf_filename, 'application/pdf')

        if pdf_url:
            print(f"PDF uploaded to S3: {pdf_url}")
            return jsonify({
                'success': True,
                'pdf_url': pdf_url,
                'message': 'PDF generated and uploaded successfully'
            })
        else:
            return jsonify({'error': 'Failed to upload to S3'}), 500

    except Exception as e:
        print(f"Error during processing: {e}")
        import traceback
        traceback.print_exc()
        return jsonify({'error': str(e)}), 500

@app.route('/get_report/<session_id>', methods=['GET'])
def get_report(session_id):
    session = SESSIONS.get(session_id)
    if not session:
        return jsonify({'error': 'Report not found'}), 404

    return jsonify({
        'report_summary': session.get('report', {}),
        'gaze_data': session.get('gaze_data', [])[-1000:],
        'events': session.get('events', []),
        'calibration_box': session.get('calibration_box', {}),
        'calibration_points': session.get('calibration_points', []),
        'boundary_polygon': session.get('boundary_polygon', []),
        'start_time': session.get('start_time', 0)
    })

if __name__ == '__main__':
    print("E-Proctor Backend Server Starting...")
    print(f"AWS S3 Bucket: {S3_BUCKET}")
    print(f"Admin Email: {ADMIN_EMAIL}")
    print(f"Reports Directory: {REPORTS_DIR}")
    print("=" * 50)
    app.run(debug=True, host='0.0.0.0', port=5000)
```

Figure 5.9: get_report Session, Send Final Report Data as JSON to Frontend

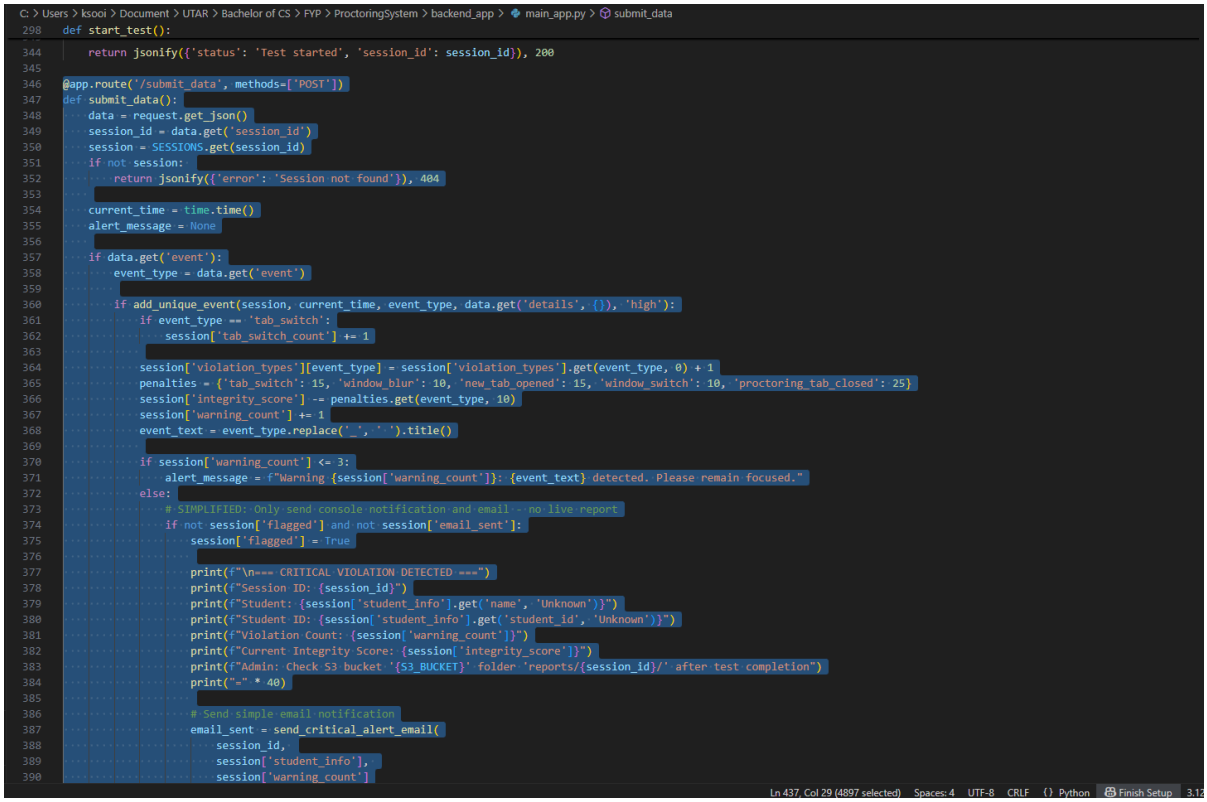


Figure 5.10: submit_data() Session, Work as Listener, Receiving JSON Data from the Frontend

5.3.3 Analysis and Reporting Module

The Analysis and Reporting Module in this system is implemented using two-stage process to get real-time alert using AWS notification possible. Instead of waiting until the test is over, the system performs real-time analysis with each event, and then a final report generated upon completion.

The first stage occurs within the /submit_data endpoint. This function acts as a live analysis engine. Each time it receives a browser activity such as a tab_switch, indicate user has switch their browser tab, it immediately calculates the impact on the student's integrity score and updates the total warning_count. This real-time processing allows the system able react towards critical violations, such as by flagging the session and triggering an email alert via AWS SES after a certain threshold of warnings is passed, which is 3 in this system.

The second stage is handled by the generate_report_data_for_s3() function, which is called by the /end_test endpoint when the student finishes their exam. The purpose of this is to format all the live-calculated data into a final, structured report. It takes the final integrity_score, determines a "Risk Level", and packages all the data including the summary,

the full gaze history, and event logs into the JSON format to the report.html page for report generation.

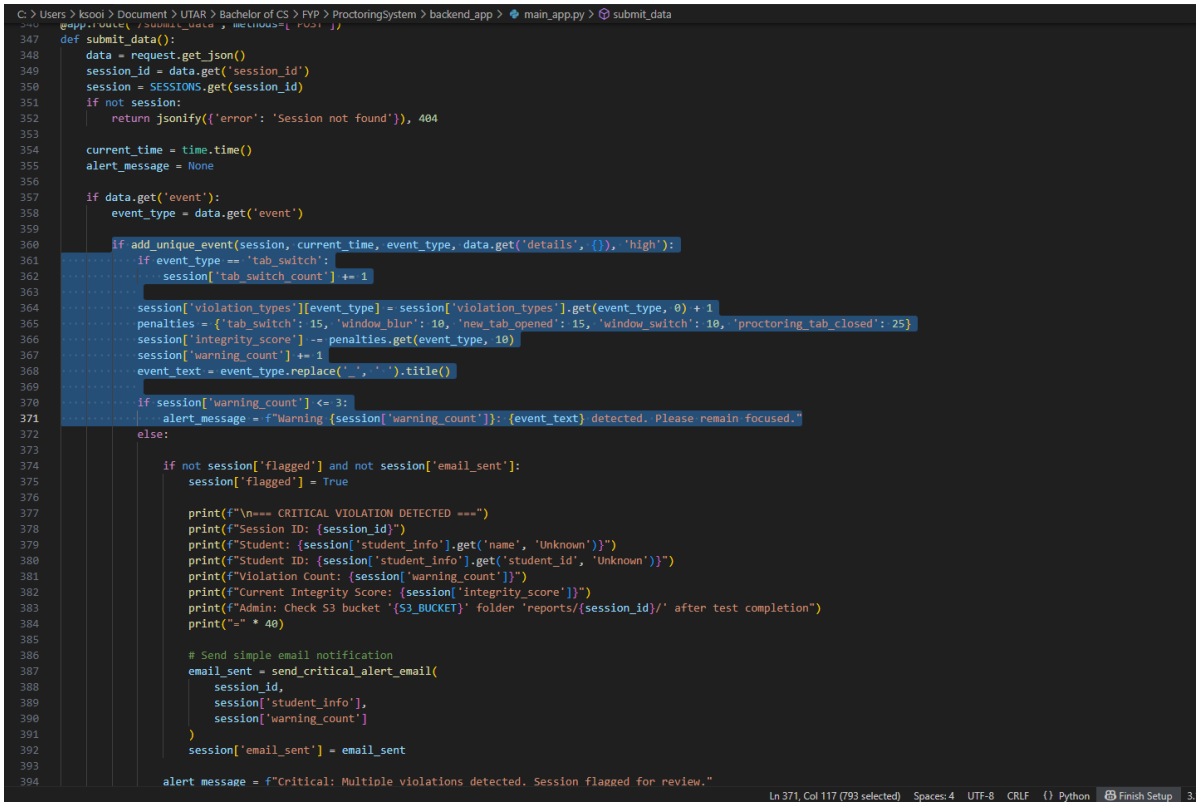


Figure 5.11: /submit_data, Showing How the System Analyzes Event and Send to Frontend

5.4 Frontend Implementation (Chrome Extension)

5.4.1 Developing Core Extension Architecture and Control

5.4.1.1 manifest.json

The manifest.json file is the blueprint of the Chrome Extension. It is a mandatory, file name sensitive, strictly formatted configuration file that serves like the table of contents, letting the Chrome browser know the structure of the file, defining everything such as the name and version. The file paths and names are case-sensitive and must precisely match the local file structure, as any dismatch will prevent the extension from loading or functioning correctly.

A key rule set by the manifest is keeping things secure. This is due to Chrome practice strict Content Security Policy (CSP), an extension is treated as a self-contained package.[15] This policy is strict in the way that all resources must be local. This is a security measure set to prevent remote code execution vulnerabilities, where an extension might fetch and run malicious scripts from external server.

The manifest enforces this by defining key properties like the background service worker (background.js) and the popup window (popup.html), which must be local files. Furthermore, the web_accessible_resources key is used to declare local resources are permitted to be accessed by web pages.

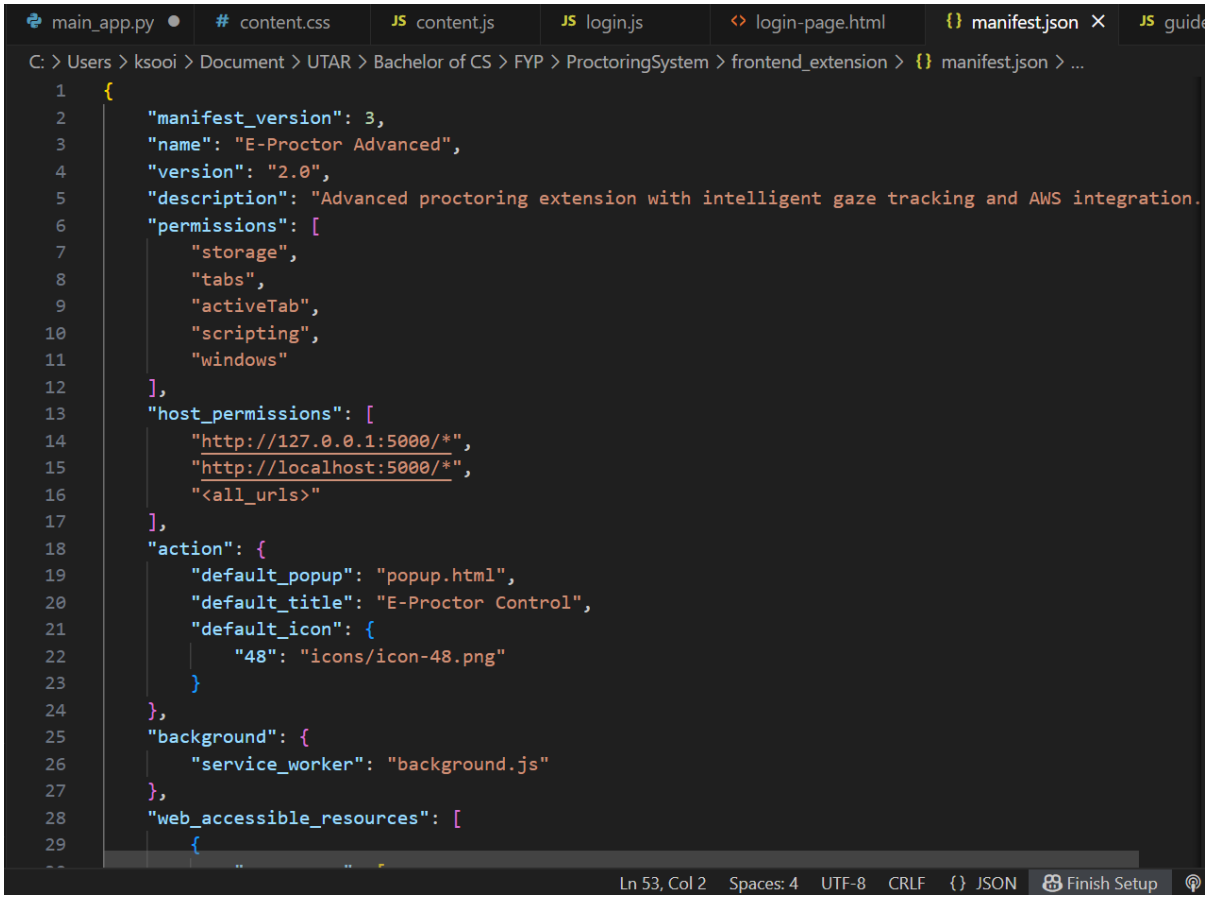


Figure 5.12: manifest.json

5.4.1.2 background.js

This script is the central nervous system of the extension, it is the communication hub with the backend server. The architectural design mandated by Google Chrome for extensions. As mentioned, it is registered in the manifest.json as a service worker. This allowing it to run persistently in the background. This persistence is critical if the communication logic were placed in other file like popup file, the proctoring would stop until the user closed the popup window. This robust design prevents such issue and is essential for the reliability of system.

popup.html | popup.js | report.html | report.js | sandbox.html | sandbox.js | background.js

C: > Users > ksooi > Document > UTAR > Bachelor of CS > FYP > ProctoringSystem > frontend_extension > background.js > endProctorin

```
let previousActiveTabId = null;
let lastEventTime = {}; // Track last event times to prevent duplicates

async function setState(newState, errorInfo = null) {
    state = newState;
    await chrome.storage.local.set({ proctoringState: newState });
    try {
        await chrome.runtime.sendMessage({ type: 'STATE_UPDATE', state: newState, error: errorInfo })
    } catch (e) { /* Popup may be closed */ }
    console.log(`State changed to: ${newState}`);
}

async function apiCall(endpoint, options = {}) {
    try {
        const response = await fetch(`${API_URL}${endpoint}`, {
            headers: { 'Content-Type': 'application/json' },
            ...options
        });
        if (!response.ok) throw new Error(`API Error: ${response.statusText}`);
        return await response.json();
    } catch (error) {
        console.error(`API call to ${endpoint} failed:`, error);
        throw error;
    }
}

async function startCalibration() {
    try {
        // Get student info from storage
```

Ln 130, Col 33 Spaces: 4 UTF-8 CRLF {} JavaScript Finish Setup

Figure 5.13: background.js

5.4.2 Developing the Student Authentication Interface

Before any monitoring can begin, the system must verify the student's identity. This is handled by a dedicated authentication interface. When the student first initiates the proctoring process, a new window is launched displaying a secure login form. This interface is to capture the student ID and password. An API call then send these credentials to the backend's /login endpoint. The interface provides immediate feedback to the user, displaying a "Logging in..." status and showing an error message if the backend returns an authentication failure. Upon successful validation, the script notifies the background service, and the window closes automatically.

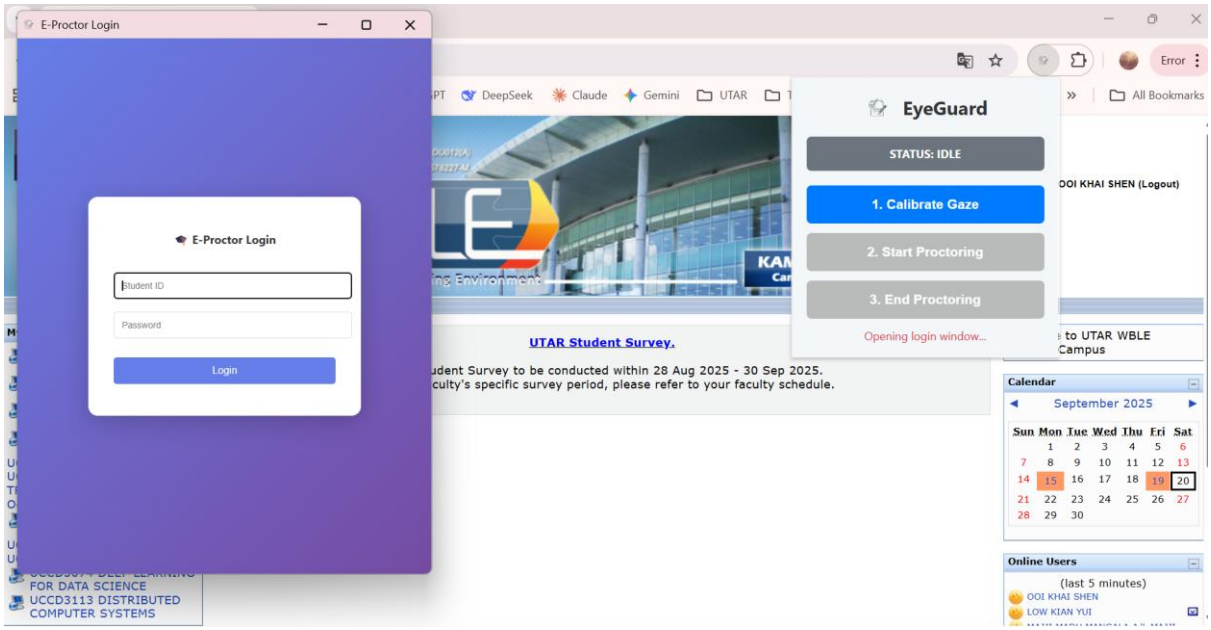
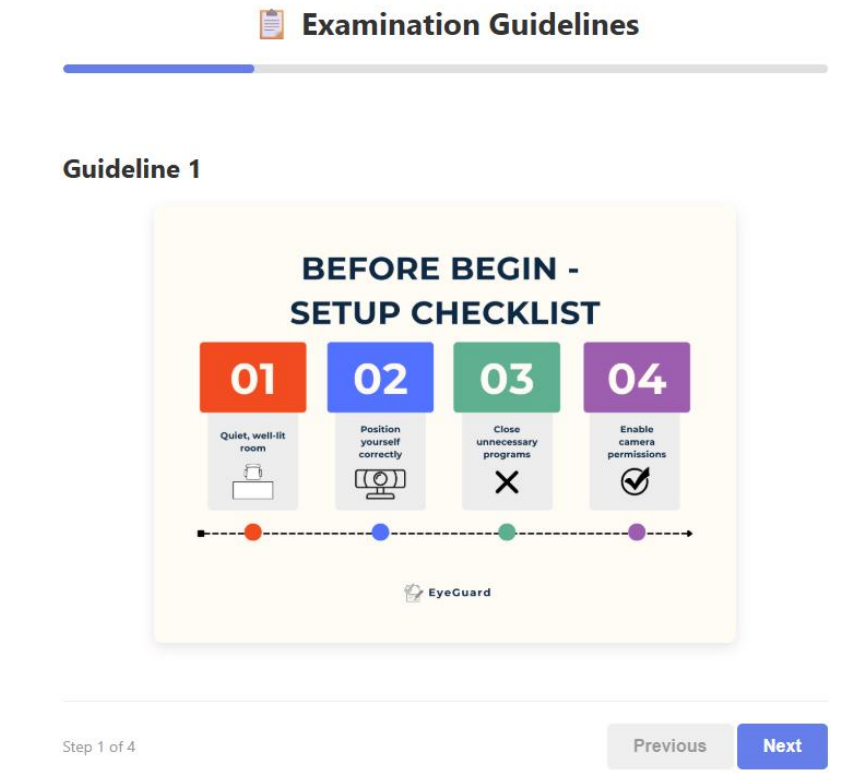
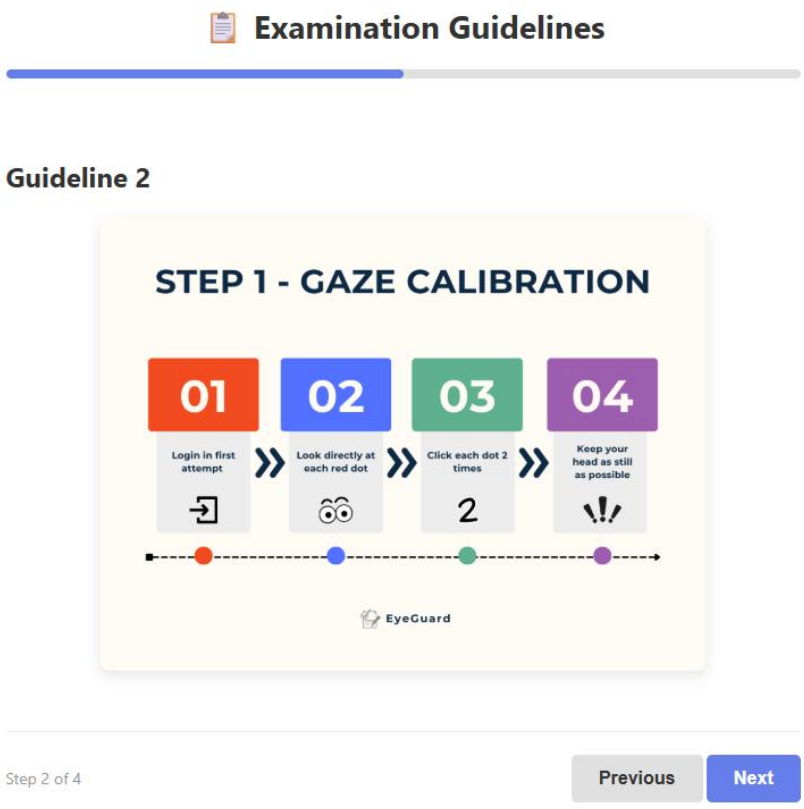


Figure 5.14: Student Authentication Interface.

5.4.3 Creating the Pre-Proctoring Guideline Module

To ensure academic integrity and properly inform the user, a mandatory guideline module is presented before successful login. This module functions to display a series of examination rules and flows that the student must acknowledge. The interface uses a progress bar to show the student's progression through the steps. Once all guidelines have been agreed to, the module sends a completion message to the background service worker and closes.


Examination Guidelines
Guideline 1
BEFORE BEGIN -
SETUP CHECKLIST
01
Quiet, well-lit room
02
Position yourself correctly
03
Close unnecessary programs
04
Enable camera permissions
EyeGuard
Step 1 of 4
Previous
Next


Examination Guidelines
Guideline 2
STEP 1 - GAZE CALIBRATION
01
Login in first attempt
02
Look directly at each red dot
03
Click each dot 2 times
2
04
Keep your head as still as possible
EyeGuard
Step 2 of 4
Previous
Next

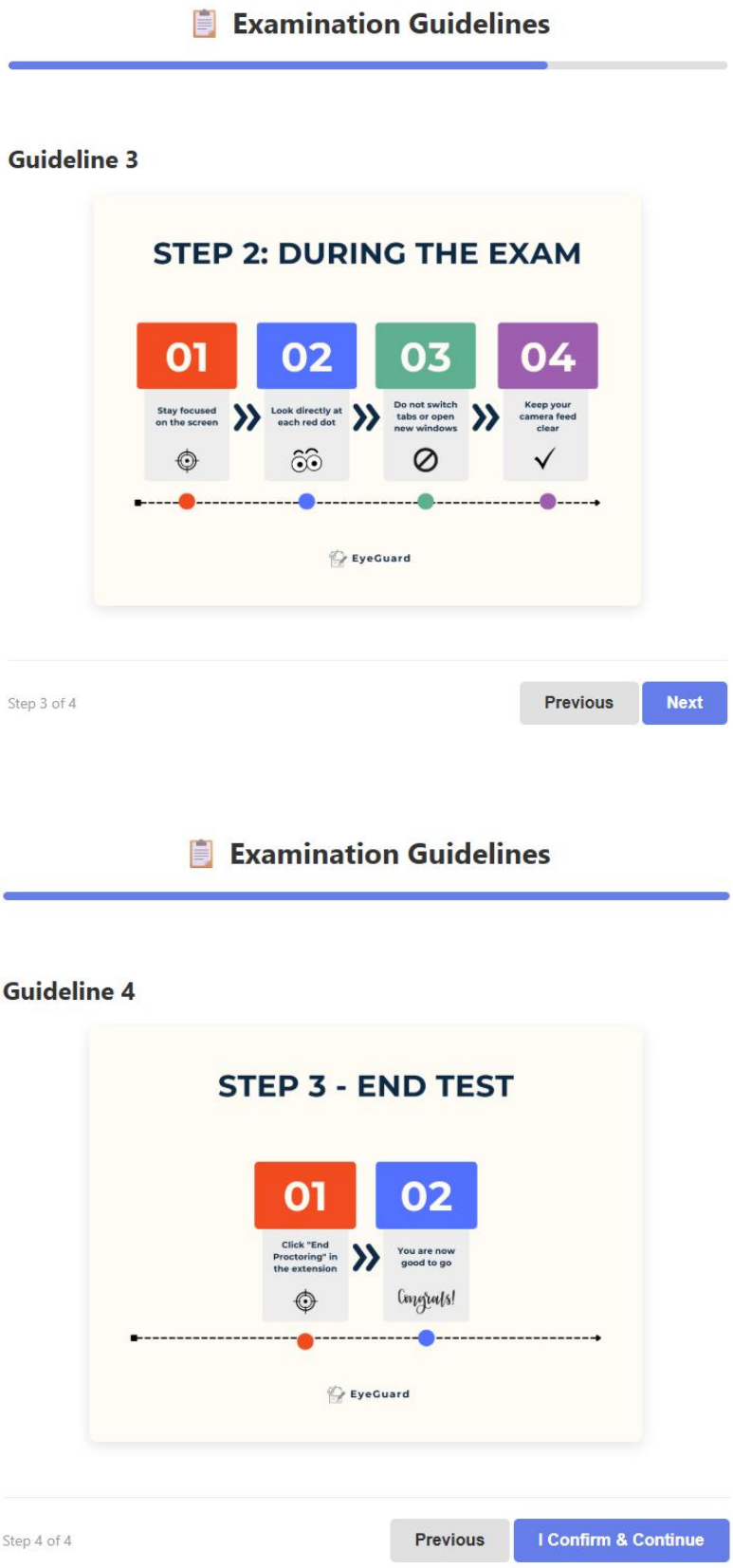


Figure 5.15: Examination Guideline Agreement Module.

5.4.4 Constructing the Main Extension Control Panel

The interface features the status of the system, display with different color-coded for immediate recognition which grey for idle, blue for ready, green for running and red for ended. The control buttons ("Calibrate," "Start," "End") are logically enabled or disabled to guide the user through the correct sequence. For instance, the "Start Proctoring" button remains disabled until calibration is successfully completed. This UI design prevents user error and provides clear guidelines throughout the examination process.

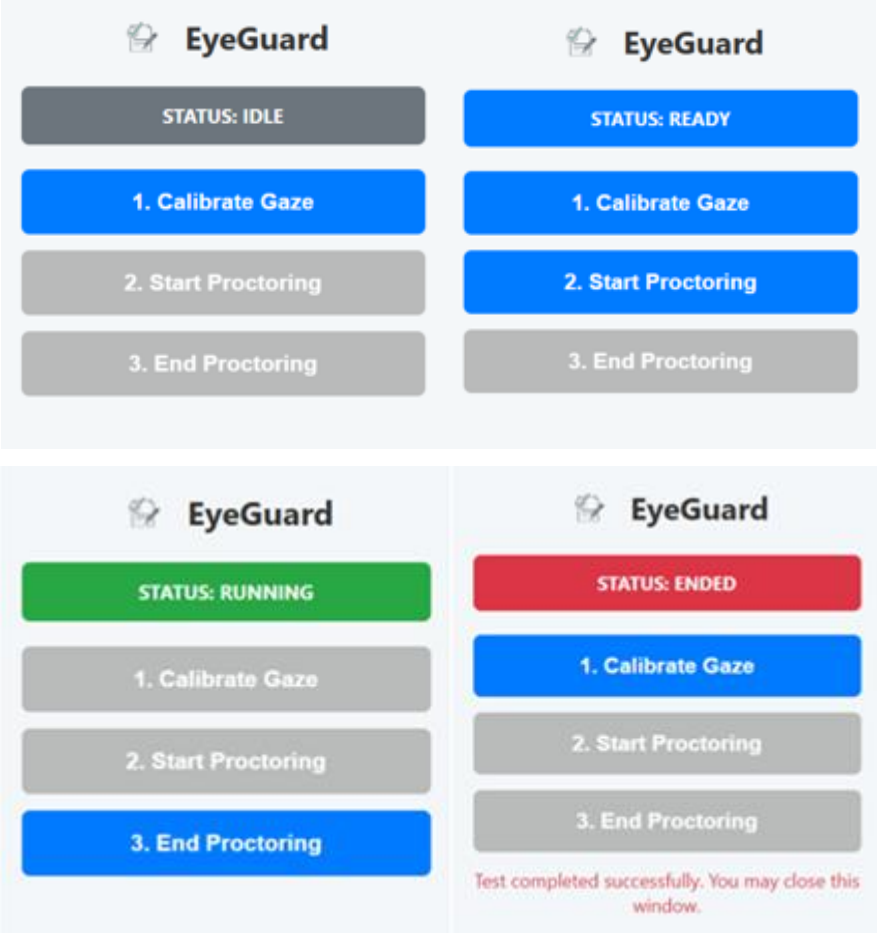


Figure 5.16: Extension Control Panel in Various States (Idle, Ready, Running).

5.4.5 Developing the Sandboxed Gaze Tracking Module

The core eye-tracking functionality is powered by the WebGazer library, an open-source tracker that uses machine learning to predict gaze location from a webcam feed. To implement this securely, all WebGazer operations are confined within a sandboxed environment.

The sandbox contains all the commands for the WebGazer instance. It handles initializing the library, configuring the prediction model for calibrating and managing the

webcam feed. During calibration, it receives messages containing the coordinates of the user clicks, which it uses to train the WebGazer model. Once monitoring begins, it continuously gets the latest gaze prediction and sends this data back to the background service.

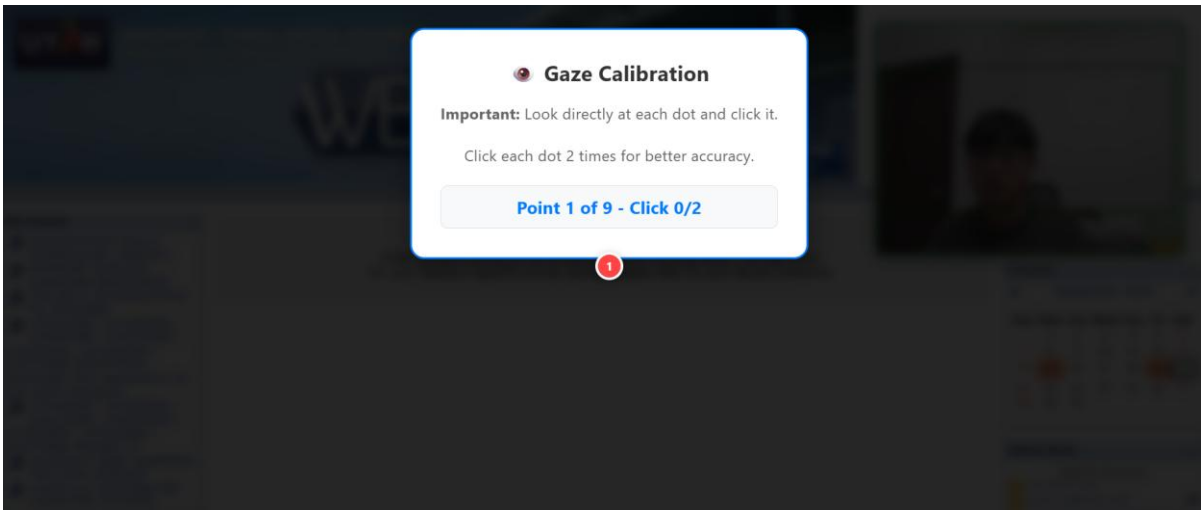


Figure 5.17: The Gaze Tracking Calibration Interface in Action.

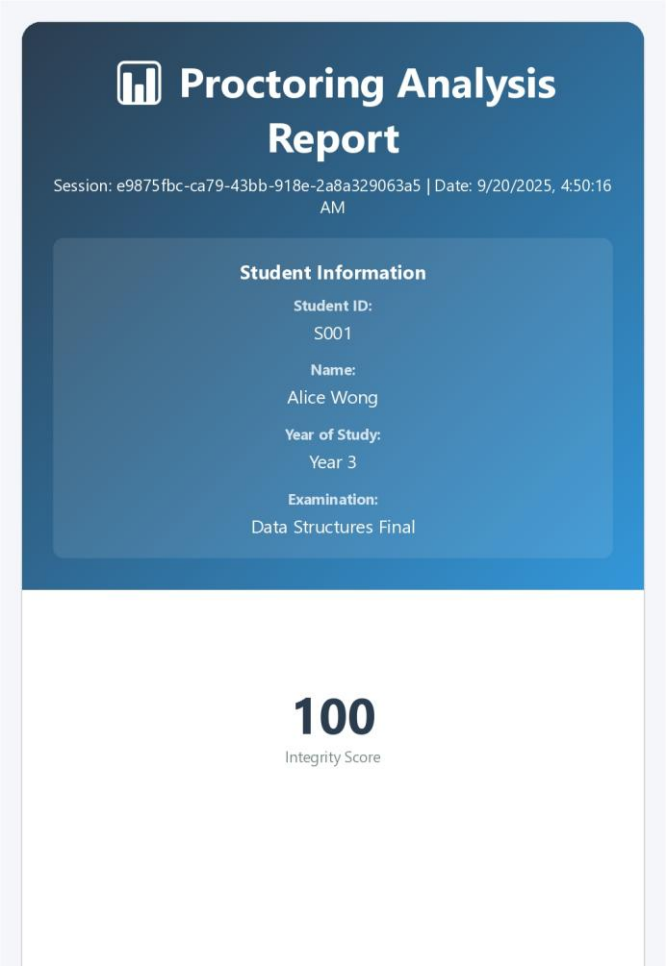
5.4.6 Developing the Post-Session Visualization Report

This final module of the frontend is dedicated to presenting the results of a completed proctoring session to an administrator or moderator. Its purpose is to transform the data logged by the backend into a clear dashboard-like report.

When the report.html page is loaded, its corresponding script, report.js, immediately takes control. The script extracts the unique session_id and receiving the comprehensive JSON object containing the full session data populates the dashboard.

Key metrics from the report_summary object, such as the final integrity score and total warning count, are injected into their respective summary cards. The student's gaze pattern is achieved using Chart.js, a powerful open-source charting library. The report script processes the large array of gaze_data coordinates and renders them as a 2D scatter plot. Finally, the script gets all the events array to construct a chronological timeline, logging every flagged incident with a timestamp and description.

This combination of statistical summaries, graphical plots, and detailed logs provides the examiner with a comprehensive overview of the particular test session.


Proctoring Analysis Report
Session: e9875fbc-ca79-43bb-918e-2a8a329063a5 | Date: 9/20/2025, 4:50:16 AM
Student Information
Student ID:
S001
Name:
Alice Wong
Year of Study:
Year 3
Examination:
Data Structures Final
100
Integrity Score

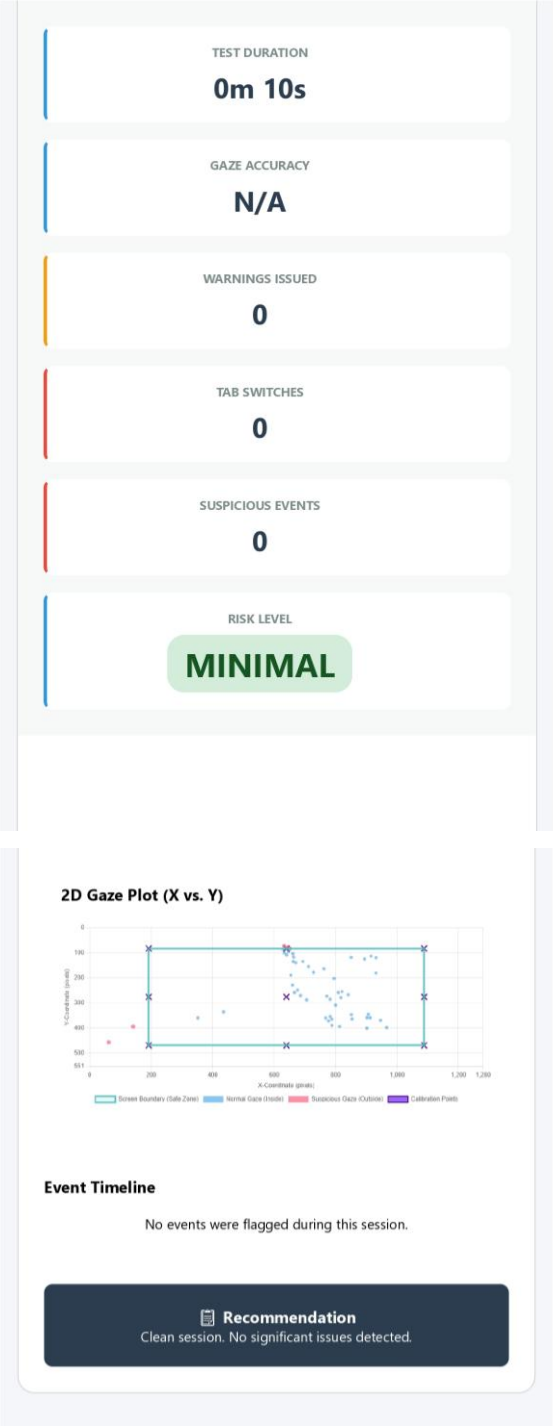


Figure 5.18: The Final Proctoring Analysis Report Dashboard

5.4.7 Implementing Real-time Violation Alerts

To maintain the integrity of the examination environment, the system provides immediate feedback to the student when a potential violation is detected. This is handled by a real-time alert module. The implementation of this feature is a collaborative effort between the backend and the frontend. When the backend's determines a warning like a tab switch or a confirmed gaze violation, it includes an alert message in its JSON response to the frontend.

The background service receives this response and immediately relays a alert message on the exam page. This modal displays the specific warning message to the student like "Warning: Tab Switch detected. Please remain focused." and closed upon confirmation.

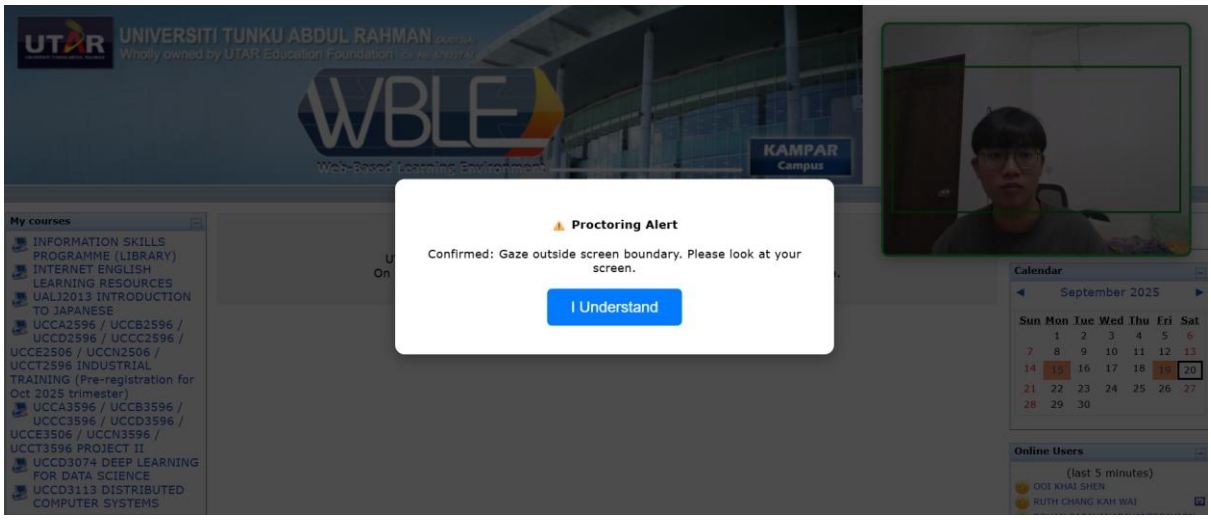


Figure 5.19: The Alert triggered when User Caught Off-screen Glace

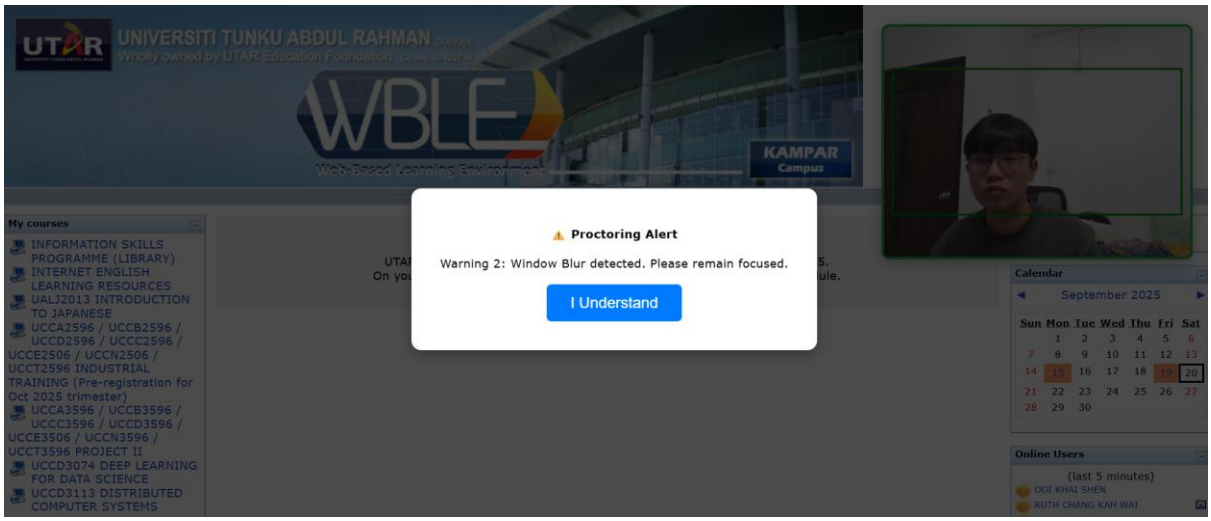


Figure 5.20: The Alert triggered when User Caught Minimize Screen

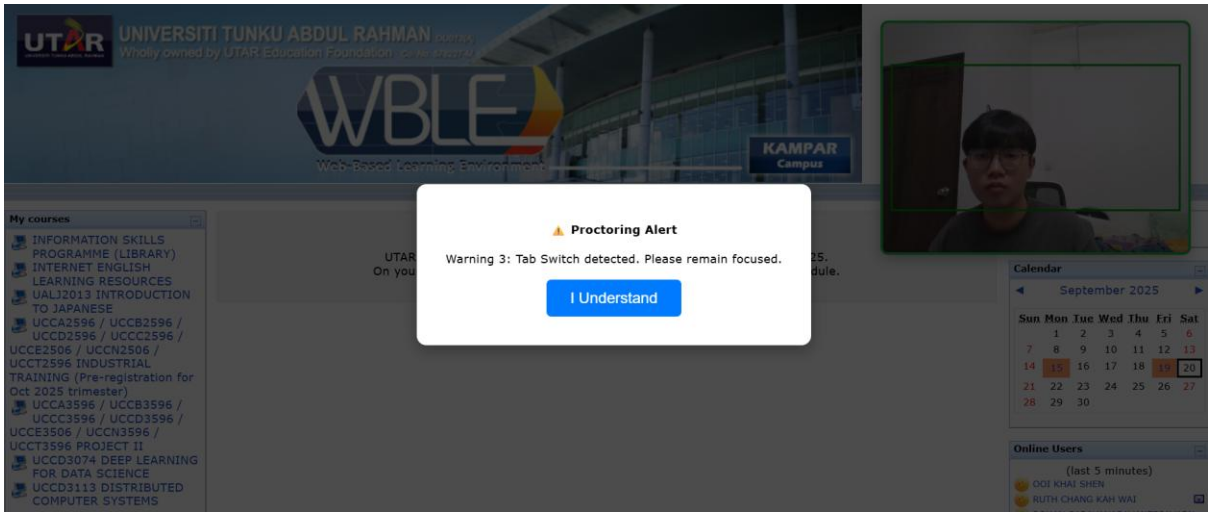


Figure 5.21: The Alert triggered when User Caught Switch Between the Tab

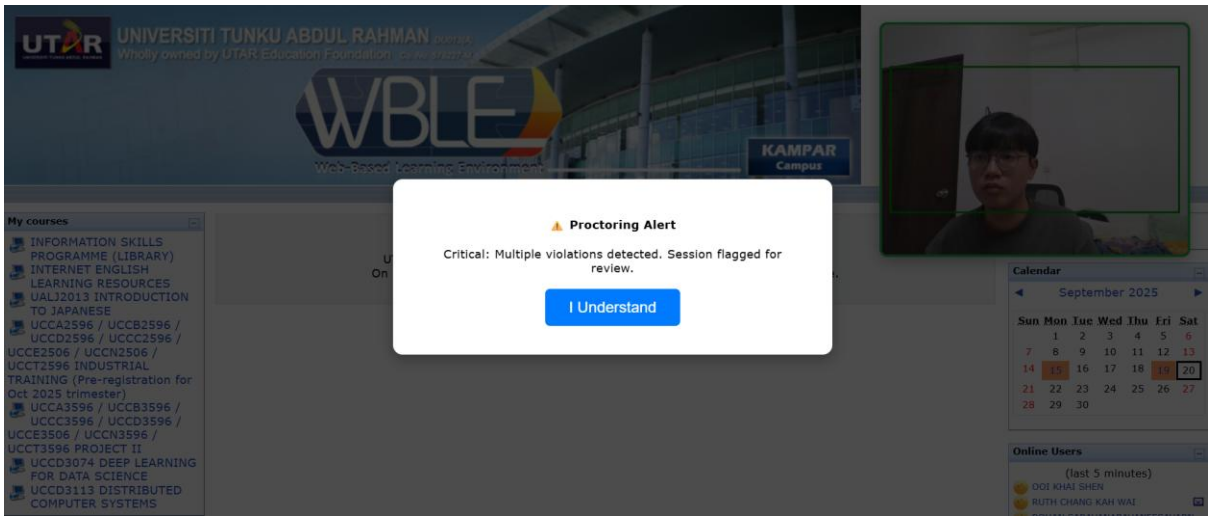


Figure 5.22: The Critical Alert triggered when User Multiple Violations

5.4.8 Integration with Cloud Services (AWS)

A key feature of the E-Proctor Advanced system is its integration with Amazon Web Services (AWS) for robust notifications and report storage. The frontend extension does not communicate with AWS directly. Instead, it acts as the trigger for the backend server

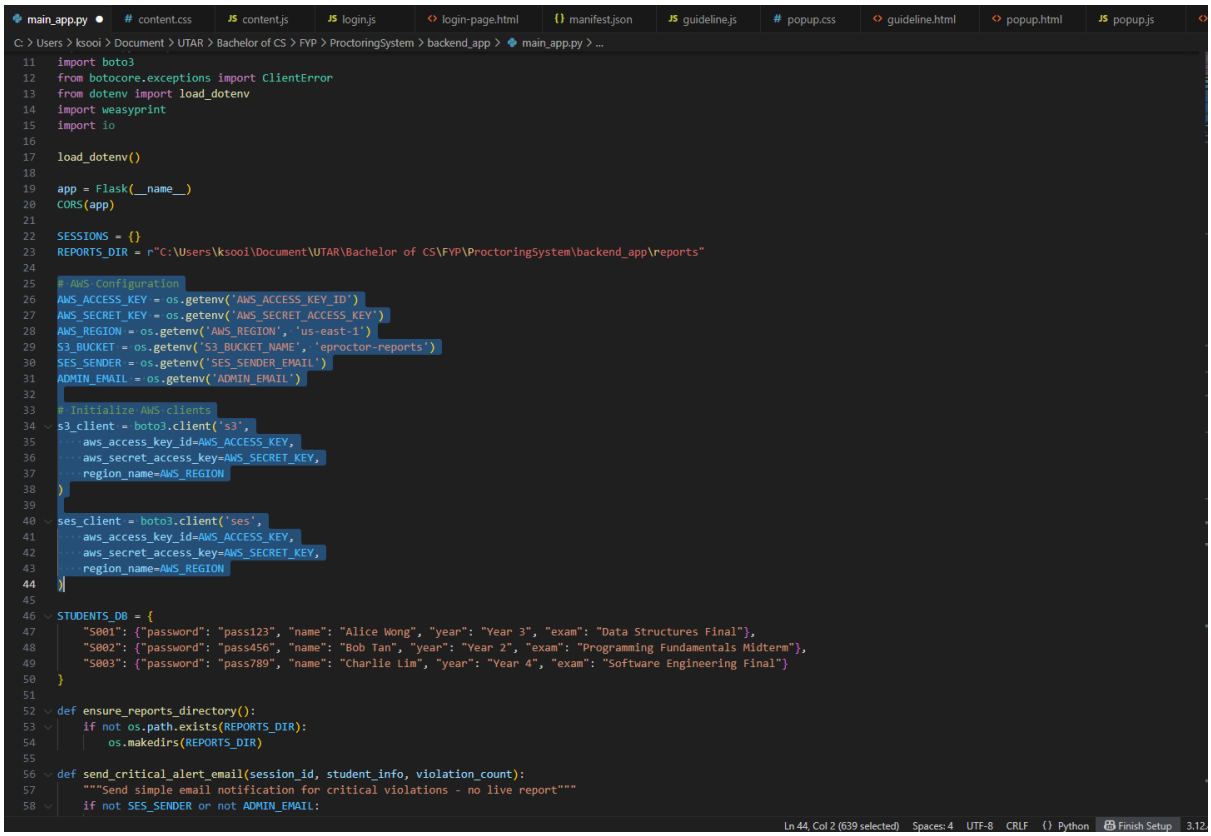


Figure 5.23: Boto3 as the Coordinator for AWS Services.

5.4.8.1 Trigger Email Alerts (AWS SES)

The system is designed to notify administrators of critical violations automatically. When the real-time analysis in the backend's /submit_data endpoint determines that a student has accumulated a critical number of warnings, the system will then uses the AWS SDK for Python (Boto3) to connect to Amazon Simple Email Service (SES) and send a pre-formatted HTML email to a designated administrator.

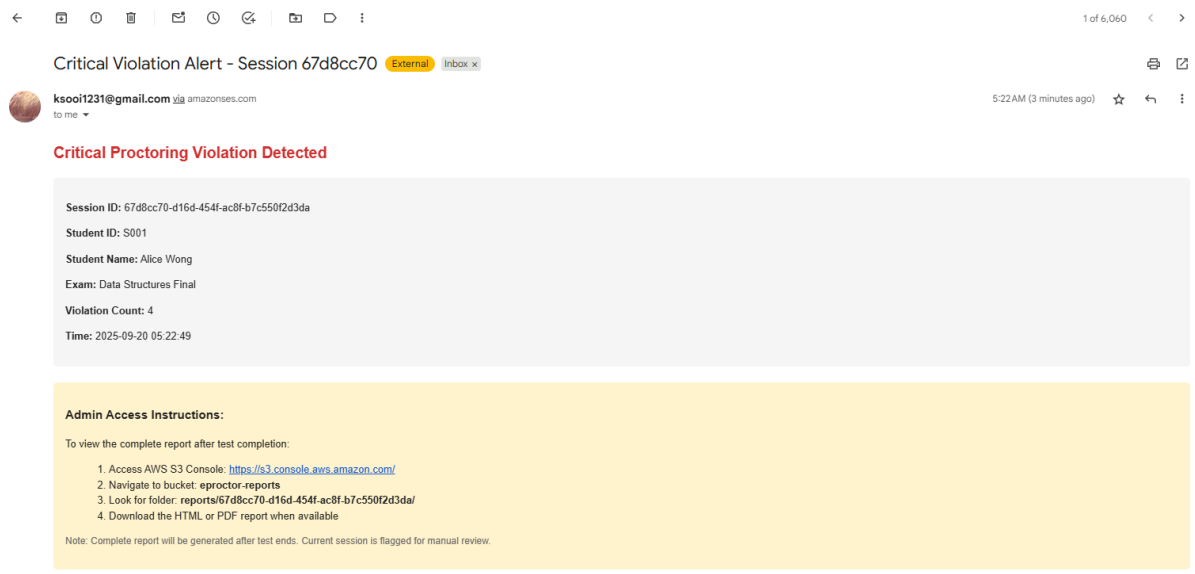
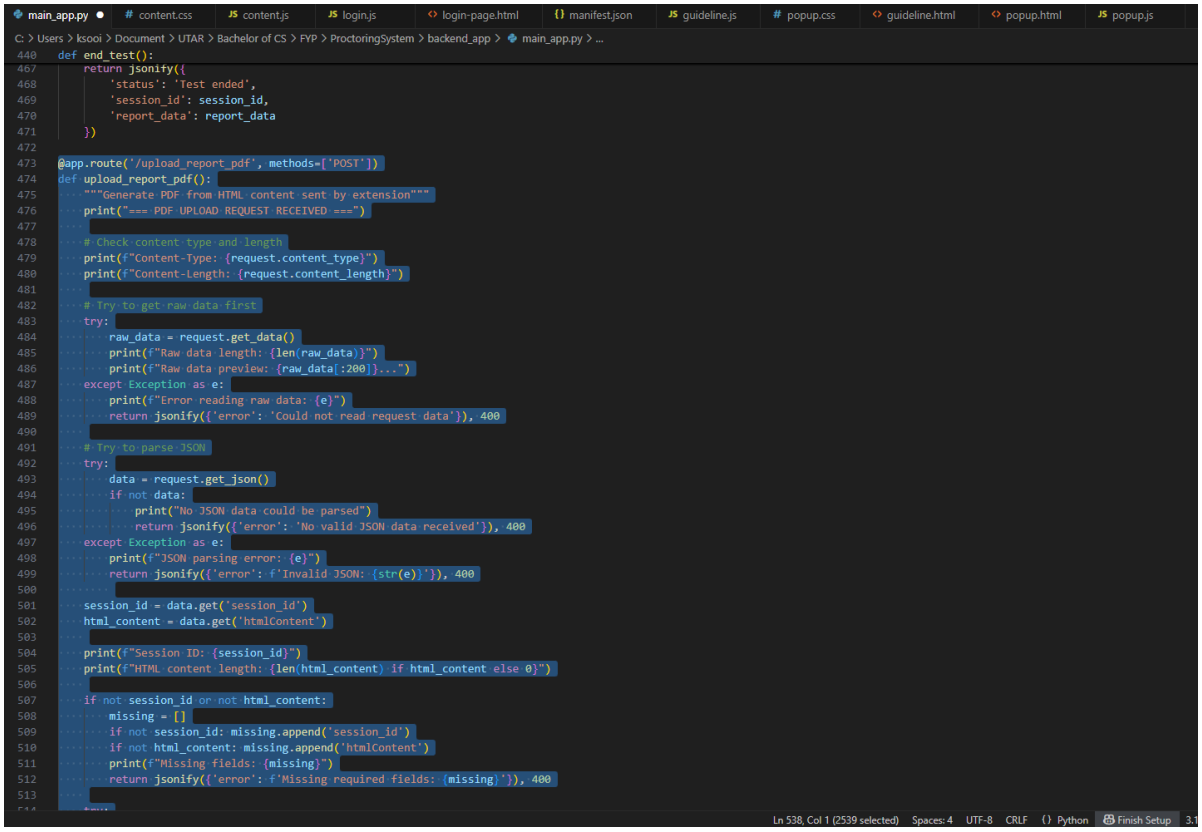


Figure 5.24: Example Critical Violation Email Alert Sent via AWS SES.

5.4.8.2 PDF Report Uploads (AWS S3)

For permanent archival, final reports will be stored in Amazon S3 (Simple Storage Service), a highly durable cloud object storage service. The process is initiated from the frontend but executed by the backend to ensure security.

The implementation begins on the report.html page, as has been mentioned in previous module, to generated a report with chart.js library. The backend server will then take over. It receives HTML content, uses the WeasyPrint library to convert it into a PDF document, and then uses the AWS SDK (Boto3) to upload this PDF file to the designated S3 bucket. The file is stored under a folder structure named after the session ID for easy organization (e.g., reports/session-id-123/final_report.pdf). This then allows the proctoring session report to be accessed by authorized administrators in S3 bucket only



```
def end_test():
    return jsonify({
        'status': 'Test ended',
        'session_id': session_id,
        'report_data': report_data
    })

@app.route('/upload_report_pdf', methods=['POST'])
def upload_report_pdf():
    """Generate PDF from HTML content sent by extension"""
    print("=== PDF UPLOAD REQUEST RECEIVED ===")

    # Check content type and length
    print(f"Content-Type: {request.content_type}")
    print(f"Content-Length: {request.content_length}")

    # Try to get raw data first
    try:
        raw_data = request.get_data()
        print(f"Raw data length: {len(raw_data)}")
        print(f"Raw data preview: {raw_data[:200]}...")
    except Exception as e:
        print(f"Error reading raw data: {e}")
        return jsonify({'error': 'Could not read request data'}), 400

    # Try to parse JSON
    try:
        data = request.get_json()
        if not data:
            print("No JSON data could be parsed")
            return jsonify({'error': 'No valid JSON data received'}), 400
    except Exception as e:
        print(f"JSON parsing error: {e}")
        return jsonify({'error': f'Invalid JSON: {str(e)}'}), 400

    session_id = data.get('session_id')
    html_content = data.get('htmlContent')

    print(f"Session ID: {session_id}")
    print(f"HTML content length: {len(html_content) if html_content else 0}")

    if not session_id or not html_content:
        missing = []
        if not session_id: missing.append('session_id')
        if not html_content: missing.append('htmlContent')
        print(f"Missing fields: {missing}")
        return jsonify({'error': f'Missing required fields: {missing}'}), 400
```

Figure 5.25: upload_report_pdf(), to Upload the PDF Report to S3 Bucket

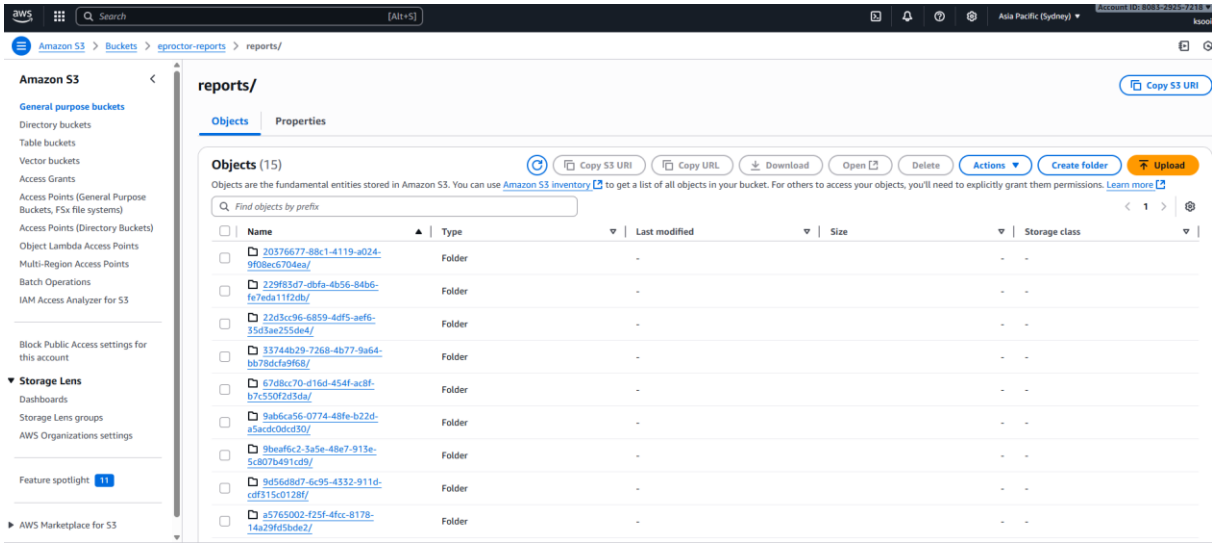


Figure 5.26: Administrator able to View the Reports Stored inside S3 Bucket

CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION

This chapter presents a comprehensive evaluation of the completed "EyeGuard" system, detailing the methodology and results of the tests conducted to validate its functionality and performance. The following sections will outline the testing setup, present the results of functional verification through a series of structured test cases, analyze the system's effectiveness and resource impact, and provide a comparative analysis against the initial prototype. The chapter concludes by discussing the project's challenges and evaluating its success against the original objectives.

6.1 Functional Verification

These test cases are to confirm the operational of the system. The following test cases document the successful execution of the system primary user flows and backend processes, with all tests achieving a "PASS" status.

Test Case	Description	Test Data	Result
TC-01	Guideline showing	A new user attempts calibration	The system correctly redirects to the guideline.html page, which must be confimed before proceeding
TC-02	User Authentication	A first-time login user clicks "Calibrate".	The login-page.html window appears for the first time login and successfully authenticates with valid credentials.
TC-03	Gaze Calibration	User clicks all 9 on-screen points twice.	The calibration UI is removed, and the extension's internal state updates to "Ready".

Table 6.1: Test Cases for User Onboarding and Setup

Test Case	Description	Test Data	Result
TC-04	Session Lifecycle	User clicks "Start Proctoring" and later "End Proctoring".	A session is correctly initiated with the backend, and upon ending, the session is terminated and the report generation process is triggered.
TC-05	Browser Violation Detection	User switches tabs during an active session.	A "tab_switch" event is logged, an on-screen alert is displayed, and the Integrity Score is correctly reduced

			by 15 points as well as minimize action
TC-06	Critical Email Alert	Users trigger more than 3 violations.	An alert is sent to the administrator email address via AWS SES.
TC-07	Report Generation & Storage	A proctoring session is ended.	A PDF report are generated and successfully uploaded to the correct folder within the AWS S3 bucket.

Table 6.2: Test Cases for Core Proctoring and Cloud Integration

6.2 Technology Justification and Accuracy

As mentioned in literature review, the reliability of EyeGuard is fundamentally based on the robustness of its core gaze-tracking technology. This section validates the selection of WebGazer and further down the accuracy.

WebGazer is not only a landmark detector, it is a in-browser eye-tracking solution. Its primary innovation is an integrated adaptive regression model that is trained in real-time through user interactions. This self-calibrating nature allows it to create a personalized mathematical map of a user facial features to their screen coordinates, making it highly adaptive to real-world variables.

Research has demonstrated that the model used in this project can achieve an average on-screen error of approximately 130 pixels without specialized hardware.[12] This level of accuracy is highly effective for the specific goals of a proctoring system. This is due to when a student looks from their laptop monitor to a note on a nearby wall, their gaze might shift by over 1,000 pixels. In this context, a prediction error of 130 pixels is negligible and does not impact on the system's ability to correctly identify a significant off-screen gaze event. Furthermore, this precision is complemented by the model robustness because WebGazer maps the student face in 3D, it able to detect for natural head movements, ensuring the tracking remains stable during the online examination.

This validated performance confirms that the choice of WebGazer provides a strong technological base for the project.

6.3 Performance and Effectiveness Analysis

This section moves beyond functional checks to analyze how well the system performs, focusing on the intelligence of its custom algorithms and its efficiency in a real-world scenario.

6.3.1 Validation of Gaze Violation Logic

To validate the core proctoring logic, a direct functional test of the gaze analyzes and violation algorithms, “SmartGazeMonitor” was conducted. The test was designed to confirm that the system correctly distinguishes between various gazing behaviors by monitoring the backend server log for alerts. During the test, the frontend extension continuously submitted gaze data to the backend at a rate of five times per second (5 Hz).

The test involved four distinct 10-second scenarios:

1. **On-Screen Focus:**
   The user focused entirely on the screen to establish a baseline and test for false positives.
2. **Sustained Off-Screen Gaze:**
   The user stared at a fixed point off-screen to confirm a violation would be triggered and to measure the detection time.
3. **Natural Off-Screen Glance:**
   The user performed a single, brief glance away from the screen, which under 1 second and immediately returned their focus to the screen. This tests the system's ability to ignore natural, non-suspicious movements.
4. **"Shifty Eyes" Test:**

The user repeatedly alternated between looking on-screen and glancing off-screen. This tests the temporal filter's logic, the gaze analyzes and violation algorithms, to see if multiple glances in quick succession, which maybe an action of user cheating would trigger an alert or not.

The results, summarized in Table 6.3, definitively validate the algorithm's effectiveness.

Test Scenario	Duration	Expected Outcome	Actual Result	Status
On-Screen Focus	10s	No alerts should be triggered.	No alerts were generated. The log showed normal data submission.	Pass
Sustained Off-Screen Gaze	10s	An alert should be triggered.	An alert was consistently triggered after 2.5 seconds of off-screen gazing.	Pass

Natural Off-Screen Glance	10s	No alert should be triggered.	The brief glance was correctly ignored by the temporal filter. No alert was generated.	Pass
"Shifty Eyes" Test	10s	An alert should be triggered.	An alert was triggered after the 3rd off-screen glance, correctly identifying a suspicious pattern	Pass somehow

Table 6.3: Gaze Violation Logic Test Results

6.3.2 Gaze Violation Test Cases Discussion

The results summarized in Table 6.1 provide a clear validation of the gaze analyzes and violation algorithms algorithms effectiveness and nuanced design. The "On-Screen Focus" scenario confirms that the system remains stable and does not generate false positives during normal user interaction. In the "Sustained Off-Screen Gaze" test, the system reliably triggered an alert after approximately 2 seconds. This delay is a due to the temporal filter, which requires a consistent pattern of off-screen data before confirming a violation to prevent false positive and ensure high confidentiality. Critically, the "Natural Off-Screen Glance" test was correctly ignored by the algorithm. This demonstrates the system able to differentiate between a innocent movement and a genuinely suspicious action. Finally, the "Shifty Eyes" test confirmed the system successfully flagging a pattern of repeated intentionally glances after few times failure, the algorithm proved it can detect not only long stares off-glances but also more subtle cheating behaviors.

However, the difficulty in detecting subtle "shifty eyes" patterns is not a system flaw but a algorithms design trade-off. Capturing every brief off-screen glance would require setting the system's sensitivity extremely high, which would greatly increase false positives flagging students for normal, harmless movements as a violent one. To avoid this, the system is tuned to minimizing false positives, even if it means occasionally missing very subtle suspicious behaviors. This approach ensures fairer monitoring by prioritizing accuracy. This also ensures that innocent students are not unfairly flagged for cheating, which would cause unnecessary trouble and frustration.

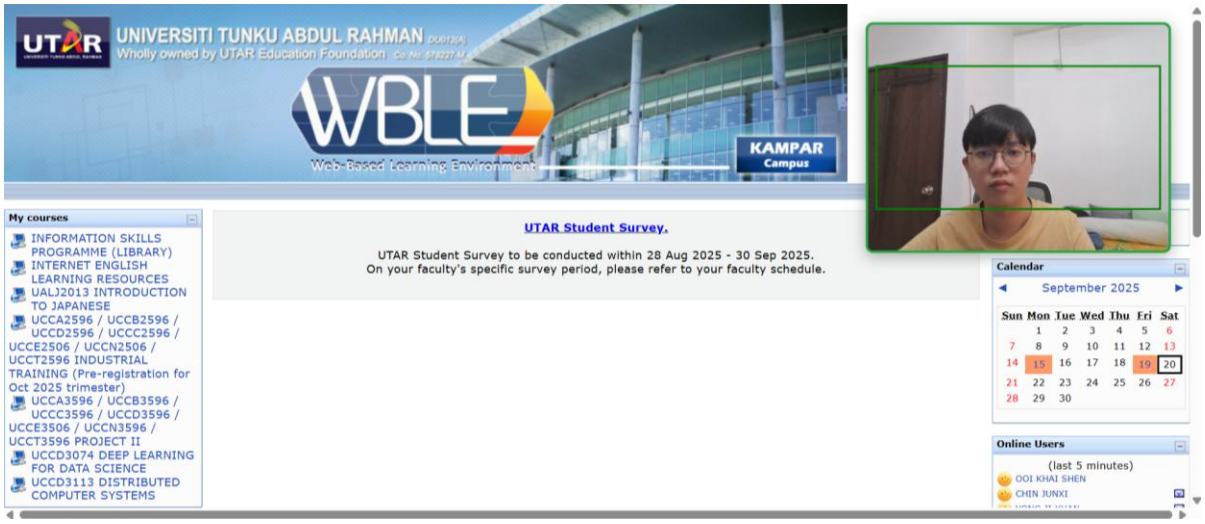


Figure 6.1: Gaze Pattern when User Focused on the Screen

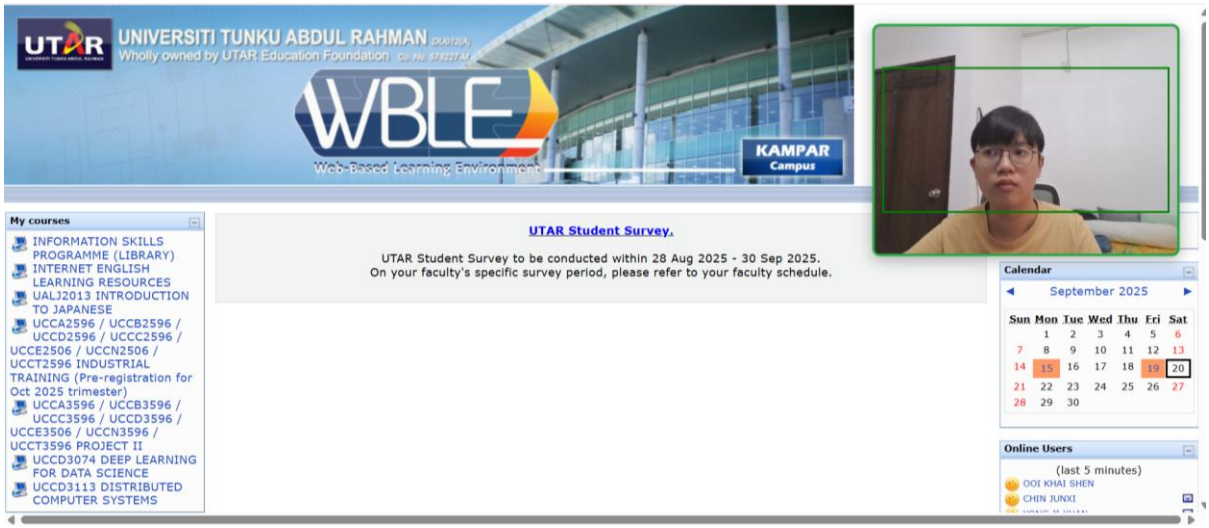


Figure 6.2: Gaze Pattern when User Focused Off-screen

```
127.0.0.1 - - [20/Sep/2025 15:24:08] "POST /submit_data HTTP/1.1" 200 -
127.0.0.1 - - [20/Sep/2025 15:24:09] "POST /submit_data HTTP/1.1" 200 -
127.0.0.1 - - [20/Sep/2025 15:24:09] "POST /submit_data HTTP/1.1" 200 -
127.0.0.1 - - [20/Sep/2025 15:24:09] "POST /submit_data HTTP/1.1" 200 -
127.0.0.1 - - [20/Sep/2025 15:24:09] "POST /submit_data HTTP/1.1" 200 -
127.0.0.1 - - [20/Sep/2025 15:24:09] "POST /submit_data HTTP/1.1" 200 -
127.0.0.1 - - [20/Sep/2025 15:24:10] "POST /submit_data HTTP/1.1" 200 -
127.0.0.1 - - [20/Sep/2025 15:24:10] "POST /submit_data HTTP/1.1" 200 -
127.0.0.1 - - [20/Sep/2025 15:24:10] "POST /submit_data HTTP/1.1" 200 -
127.0.0.1 - - [20/Sep/2025 15:24:10] "POST /submit_data HTTP/1.1" 200 -
127.0.0.1 - - [20/Sep/2025 15:24:10] "POST /submit_data HTTP/1.1" 200 -
127.0.0.1 - - [20/Sep/2025 15:24:11] "POST /submit_data HTTP/1.1" 200 -
127.0.0.1 - - [20/Sep/2025 15:24:11] "POST /submit_data HTTP/1.1" 200 -
127.0.0.1 - - [20/Sep/2025 15:24:11] "POST /submit_data HTTP/1.1" 200 -
127.0.0.1 - - [20/Sep/2025 15:24:11] "POST /submit_data HTTP/1.1" 200 -
127.0.0.1 - - [20/Sep/2025 15:24:11] "POST /submit_data HTTP/1.1" 200 -
127.0.0.1 - - [20/Sep/2025 15:24:12] "POST /submit_data HTTP/1.1" 200 -
127.0.0.1 - - [20/Sep/2025 15:24:12] "POST /submit_data HTTP/1.1" 200 -
127.0.0.1 - - [20/Sep/2025 15:24:12] "POST /submit_data HTTP/1.1" 200 -
```

Figure 6.3: Example System Log when User Focused Entirely on the Screen

```
127.0.0.1 - - [20/Sep/2025 15:24:12] "POST /submit_data HTTP/1.1" 200 -
NAIVE_ALERT: Gaze detected outside boundary.
127.0.0.1 - - [20/Sep/2025 15:24:12] "POST /submit_data HTTP/1.1" 200 -
NAIVE_ALERT: Gaze detected outside boundary.
127.0.0.1 - - [20/Sep/2025 15:24:12] "POST /submit_data HTTP/1.1" 200 -
NAIVE_ALERT: Gaze detected outside boundary.
127.0.0.1 - - [20/Sep/2025 15:24:13] "POST /submit_data HTTP/1.1" 200 -
NAIVE_ALERT: Gaze detected outside boundary.
127.0.0.1 - - [20/Sep/2025 15:24:13] "POST /submit_data HTTP/1.1" 200 -
NAIVE_ALERT: Gaze detected outside boundary.
127.0.0.1 - - [20/Sep/2025 15:24:13] "POST /submit_data HTTP/1.1" 200 -
NAIVE_ALERT: Gaze detected outside boundary.
127.0.0.1 - - [20/Sep/2025 15:24:13] "POST /submit_data HTTP/1.1" 200 -
NAIVE_ALERT: Gaze detected outside boundary.
127.0.0.1 - - [20/Sep/2025 15:24:13] "POST /submit_data HTTP/1.1" 200 -
NAIVE_ALERT: Gaze detected outside boundary.
127.0.0.1 - - [20/Sep/2025 15:24:14] "POST /submit_data HTTP/1.1" 200 -
NAIVE_ALERT: Gaze detected outside boundary.
127.0.0.1 - - [20/Sep/2025 15:24:14] "POST /submit_data HTTP/1.1" 200 -
NAIVE_ALERT: Gaze detected outside boundary.
```

Figure 6.4: Example System Log when User Focused Entirely on the Screen

Figure 6.5: Example System Log when User Make a Random Glance

6.3.3 Analysis of Client-Side and System-Level Resource Impact

An analysis of the system's resource impact was conducted to confirm its efficiency during a live session. The evaluation was performed on two levels:

1. Browser-level impact of the Chrome Extension
2. System-level impact of the local Python backend server.

The Chrome Task Manager results show the extension is efficient, consuming ~22 MB of memory when idle and ~148 MB during monitoring. The most significant finding comes from the Windows Task Manager, which reveals that the backend python.exe process consumed only less than 1.1% of the total CPU during the test.

This is due to WebGazer performing high computational power machine learning for gaze tracking directly in the browser which is called as in-browser machine learning, the backend server is never required to process a heavy video stream. Instead, the local backend only handles lightweight data in a JSON array, the final (x, y) gaze coordinates sent from the extension. This client-side processing approach shows the system high efficiency, proving that it is not computationally demanding on the server and is a scalable solution for monitoring.

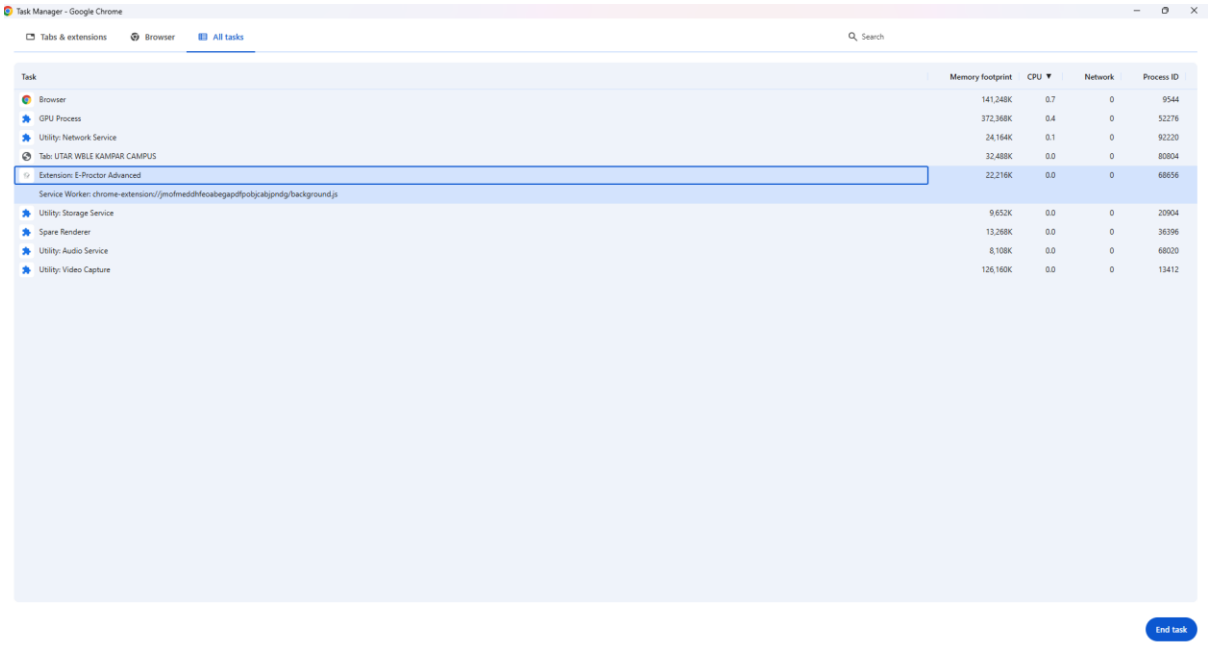


Figure 6.6: Resources Used by the Chrome Before Run the Monitoring

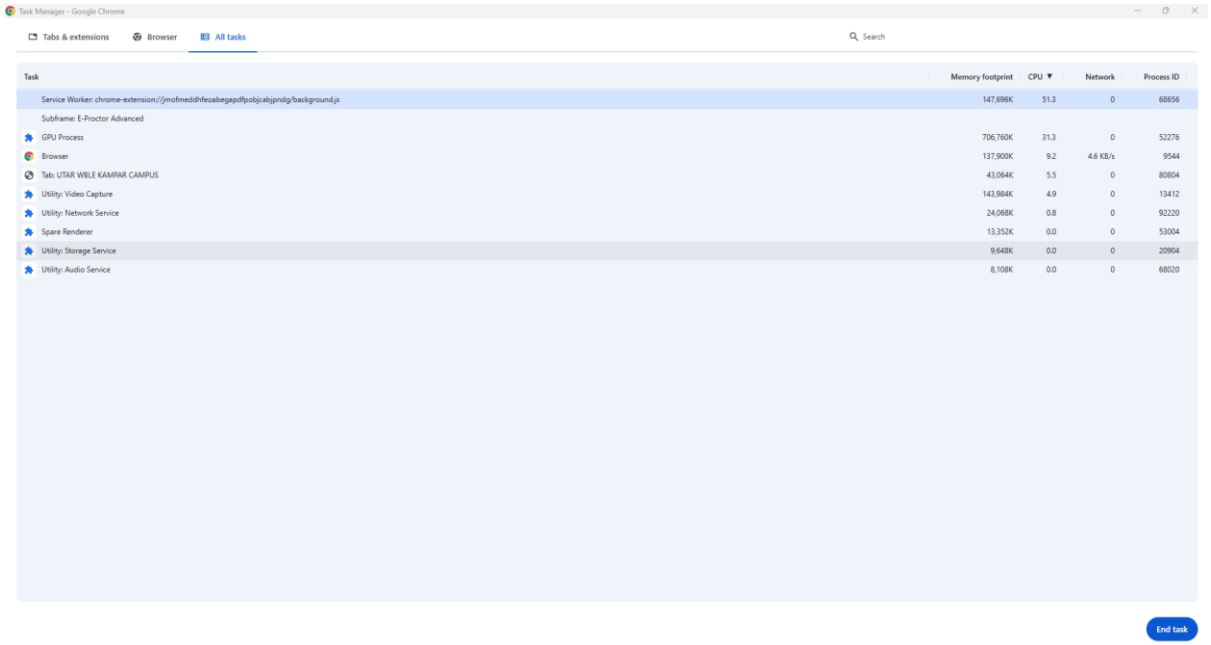


Figure 6.7: Resources Used by the Chrome After Run the Monitoring

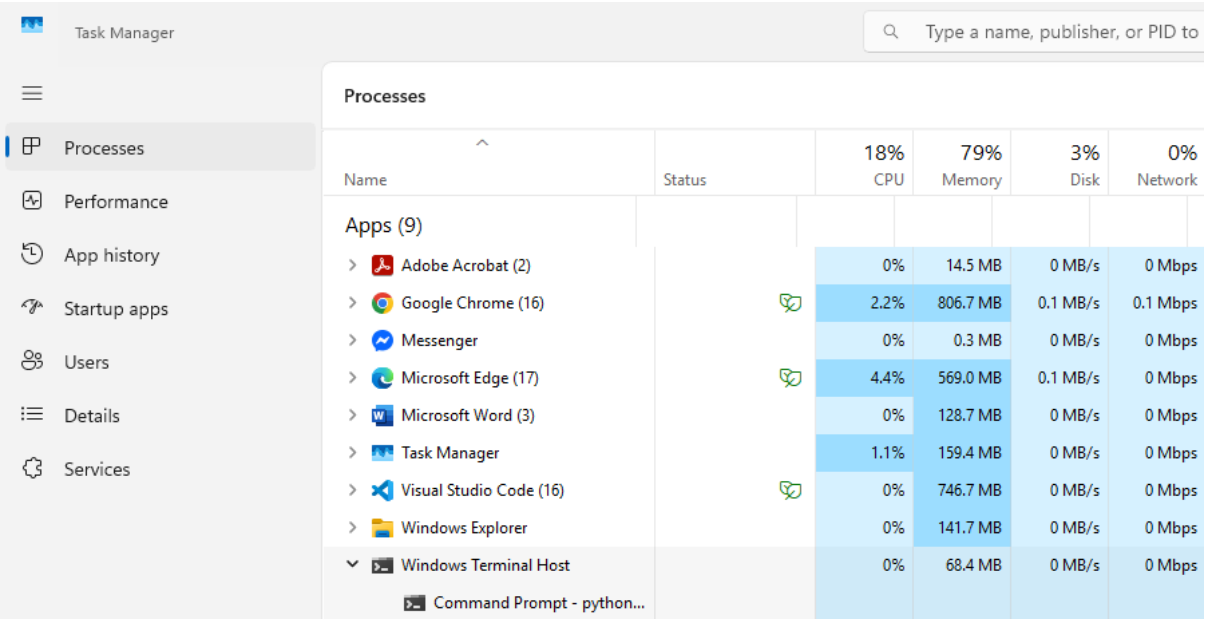


Figure 6.8: Resources Used by the System before Run the Monitoring

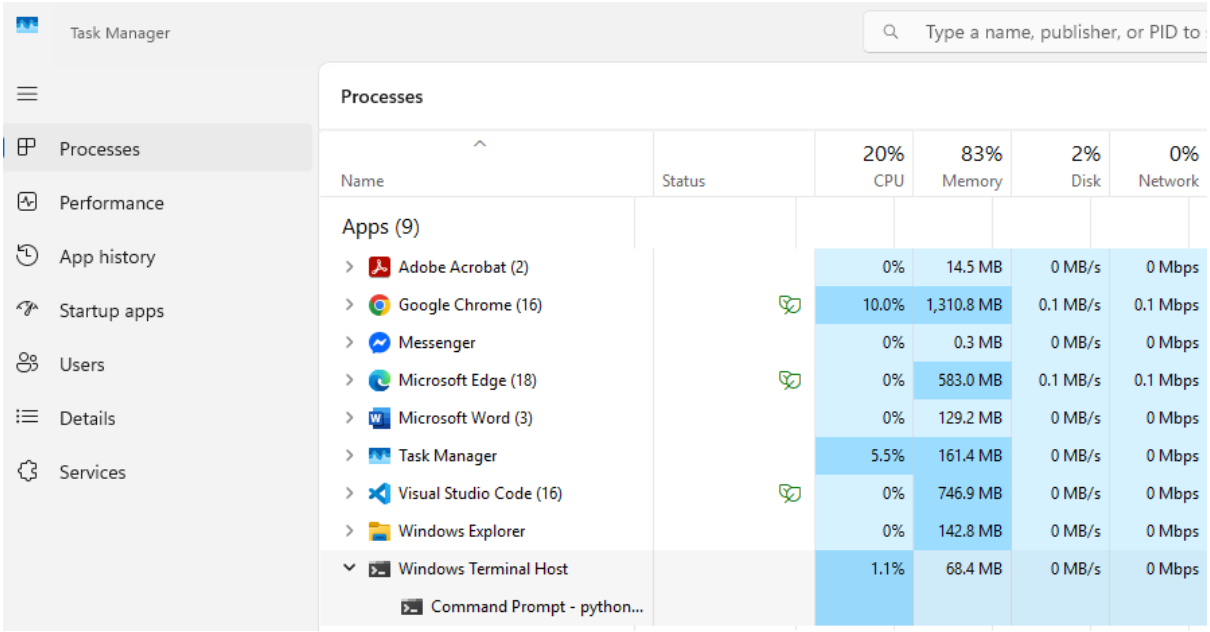


Figure 6.9: Resources Used by the System After Run the Monitoring

6.4 Comparative Analysis against FYP1

This project evaluation includes a direct comparison with the FYP1 prototype to highlight the significant advancements in technology. The evolution from FYP1 to FYP2 was driven by the need to create a more accurate, robust system, as well as addressing key limitations discovered in the initial prototype.

The FYP1 prototype required a manual calibration process that was imprecise. In contrast, the FYP2 system leverages WebGazer built-in calibration and regression engine. This is a critical advancement because it doesn't just record points; it uses them to train a personalized mathematical model that maps the user unique eye features to their screen coordinates.

Also, the FYP1 prototype uses 2D vector logic that will be sensitive to the head movement and causing errors. A slight tilt of head could ruin the system. The FYP2 system uses WebGazer 3D-aware model to solve this problem. This makes the tracking significantly more stable and reliable in a real-world setting where minor user movements are expected during the test.

These technological improvements result in a demonstrably more effective system when tested against common cheating activities.

Scenario	Description	FYP1 Prototype (MediaPipe) Outcome	FYP2 System (WebGazer) Outcome	Finding
Quick Glance	User briefly glances at the corner of the screen for <1 second.	False Positive.	Correctly Ignored.	FYP2 intelligent filtering algorithms, a violation analyze algorithm creates a fairer and less intrusive user experience.
Reading Notes on a Wall	User head remains still while their eyes move to read a	Reliable Detection.	Reliable Detection.	Both projects leveraging calibrated screen boundary for detecting off-screen activity

	note taped next to the monitor.			
Natural Head Movement:	The user tilts their head slightly while reading a question on-screen.	Ineffective.	Effective.	FYP2 3D-aware model is significantly more robust against common, natural head movements, making it far more reliable and fair for real-world proctoring.
Large Positional Change	The user makes a large, obvious movement, such as leaning far back or shifting their entire chair to the side.	Ineffective.	Somehow Ineffective.	Both systems are limited when faced with large. This reveals a fundamental constraint of webcam-based gaze tracking that relies on an initial, static calibration. The FYP2 "EyeGuard" project is somehow better in this case because it leverages 3D-aware model for eye gaze monitoring

Table 6.4: Scenario-Based Effectiveness Testing

6.4.1 Remark on Test Findings

It is important to note that the limitation identified in the second scenario, in which large positional changes are considered an acceptable trade-off for this system. In a formal examination setting, students are typically instructed by rules to remain seated and still in front of their webcam for the duration of the test. Significant movements, such as shifting chairs, are also considered suspicious behavior that maybe a cheating behavior. Therefore, the EyeGuard system is designed to be highly effective under the most common and expected condition where a user who remains relatively on their seat.

CHAPTER 7 CONCLUSION

7.1 Conclusion and Novelty

The project has successfully achieved its development goals by incorporating a comprehensive set of features into the proposed "EyeGuard" system, aimed at ensuring academic integrity during online examinations. These features are designed to provide a robust and automated approach to proctoring and are shown below:

1. **Real-Time Gaze Tracking:**
   The system uses the student's webcam to monitor their eye gaze in real-time. This feature is designed to detect when a student gazes away from the screen for a sustained period, which may indicate suspicious behavior.
2. **Browser Environment Monitoring:**
   To provide an additional layer of security, the system actively monitors the user browser for actions such as switching tabs, opening new windows, or changing focus to another application during the exam.
3. **Dual-Channel Real-Time Alerting System:**
   When a violation is detected, the system provides immediate feedback through two channels: an on-screen warning is displayed to the student, and a critical alert notification is sent via email to the administrator.
4. **Comprehensive Analysis Report:**
   At the end of each session, the system generates a detailed report for administrative review. This report includes a final "Integrity Score," a chronological timeline of all flagged events, and a 2D visual plot of the student's gaze patterns.

7.2 Recommendations

Several recommendations could further enhance the functionality, accuracy, and practical usability of the EyeGuard system. To improve its core detection capabilities, the current eye gaze monitoring could be expanded by incorporating audio analysis to detect suspicious sounds, such as third-party voices in the room. This could be also enhanced with head pose tracking to robustly handle large user movements and by implementing automated evidence capture function, a feature that would take screenshots or short video clips of violations to provide administrators with evidence proof. From an administrative perspective, the system's practicality would be greatly improved by developing a dedicated administrator

dashboard for managing sessions and reports. This would be further strengthened by replacing the current mock login system with a secure connection to institutional databases for authentication and login. Finally, to improve the user experience, a 'temporary leave' function could be implemented, allowing students to request short, logged breaks for emergencies like a toilet break. By implementing these suggestions, the EyeGuard system could evolve into an even more robust, intuitive, and efficient solution for ensuring academic integrity in online test.

REFERENCES

[1] F. Muna, A. Waheeda, F. Shaheeda, and A. Shina, "Challenges in implementing online assessments at Maldivian higher education institutions: Lessons from the COVID-19 pandemic," Environment and Social Psychology, vol. 9, no. 3, Jan. 2024. [Online]. Available: https://doi.org/10.54517/esp.v9i3.1907. [Accessed: April 2, 2025].

[2] J. Kang, S. Tariq, H. Oh, and S. S. Woo, "A survey of deep learning-based object detection methods and datasets for overhead imagery," IEEE Access, vol. 10, pp. 20118–20134, 2022. [Online]. Available: https://doi.org/10.1109/access.2022.3149052. [Accessed: Mar 23, 2025].

[3] R. Kundu, "YOLO: Real-time object detection explained," V7labs, Jan. 17, 2023. [Online]. Available: https://www.v7labs.com/blog/yolo-object-detection. [Accessed: May 2, 2025].

[4] A. Baijal, A. Cannarsi, F. Hoppe, W. Chang, S. Davis, and R. Sipahimalani, "e-Conomy SEA 2021," Bain, Nov. 10, 2021. [Online]. Available: https://www.bain.com/insights/e-conomy-sea-2021/. [Accessed: Mar 25, 2025].

[5] K. Rohit, "YOLO algorithm for object detection explained [+examples]," V7labs, 2024. [Online]. Available: https://www.v7labs.com/blog/yolo-object-detection#:~:text=Using%20a%20more%20complex%20architecture. [Accessed: Apr 12, 2025].

[6] "Remote exam proctoring," Meazure Learning, Jun. 7, 2024. [Online]. Available: https://www.meazurelearning.com/exam-proctoring/remote-exam-proctoring. [Accessed: May 1, 2025].

[7] "A comprehensive learning integrity platform - Proctorio," Proctorio. [Online]. Available: https://proctorio.com/. [Accessed: April 21, 2025].

[8] Respondus, "LockDown Browser - Respondus," 2019. [Online]. Available: https://web.respondus.com/he/lockdownbrowser/. [Accessed: Mar 12, 2025].

[9] Atlassian, "Agile best practices and tutorials," Atlassian, 2019. [Online]. Available: https://www.atlassian.com/agile. [Accessed: May 1, 2025].

[10] Google for Developers, “Face landmark detection guide | Google AI Edge,” Nov. 4, 2024. [Online]. Available: https://ai.google.dev/edge/mediapipe/solutions/vision/face_landmarker [Accessed: Sep 12, 2025].

[11] Y. Kartynnik, A. Ablavatski, I. Grishchenko, and M. Grundmann, “Real-time Facial Surface Geometry from Monocular Video on Mobile GPUs,” arXiv preprint arXiv:1907.06724, Jul. 2019. [Online]. Available: https://arxiv.org/abs/1907.06724 [Accessed: Sep 19, 2025].

[12] A. Papoutsaki, N. Daskalova, P. Sangkloy, J. Huang, J. Laskey, and J. Hays, “WebGazer: Scalable Webcam Eye Tracking Using User Interactions,” [Online]. Available: https://cs.brown.edu/people/apapouts/papers/ijcai2016webgazer.pdf [Accessed: Sep 18, 2025].

[13] S. Gray, “Agile Software Development Life Cycle,” Medium, Aug. 18, 2020. [Online]. Available: https://serenagray2451.medium.com/agile-software-development-life-cycle-b3ed0f0f7212 [Accessed: Sep 20, 2025].

[14] A. Papoutsaki, J. Tompkin, X. Koo, A. Gokaslan, I. De Smet, and J. Huang, “WebGazer.js: Democratizing Webcam Eye Tracking on the Browser,” WebGazer Project. [Online]. Available: https://webgazer.cs.brown.edu/ [Accessed: Sep 19, 2025].

[15] J. Medley, “Content Security Policy,” Chrome for Developers, 2017. [Online]. Available: https://developer.chrome.com/docs/privacy-security/csp [Accessed: Sep 18, 2025].

[16] R. Neupane, “Facial Landmark Detection - Riwaj Neupane - Medium,” Medium, Jan. 14, 2024. [Online]. Available: https://medium.com/@RiwajNeupane/facial-landmark-detection-a6b3e29eac5b [Accessed: Sep 19, 2025].

[17] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, and M. Cifrek, “A brief introduction to OpenCV,” in Proc. 35th Int. Conv. MIPRO, May 2012, pp. 1725-1730. [Online]. Available: https://ieeexplore.ieee.org/document/6240859 [Accessed: Sep 20, 2025].

[18] A. F. Abate, C. Bisogni, A. Castiglione, and M. Nappi, "Head pose estimation: An extensive survey on recent techniques and applications," *Pattern Recognition*, vol. 127, p. 108591, Jul. 2022. doi: https://doi.org/10.1016/j.patcog.2022.108591 [Accessed: Sep 20, 2025].

[19] V. Agarwal, "Real-Time Head Pose Estimation in Python," Medium, Jul. 25, 2020. [Online]. Available: https://medium.com/data-science/real-time-head-pose-estimation-in-python-e52db1bc606a [Accessed: Sep 20, 2025].

[20] A. Asperti and D. Filippini, "Deep Learning for Head Pose Estimation: A Survey," SN Computer Science, vol. 4, no. 4, Apr. 2023. doi: https://doi.org/10.1007/s42979-023-01796- [Accessed: Sep 21, 2025].

[21] A. Al-Rahayfeh and M. Faezipour, "Eye Tracking and Head Movement Detection: A State-of-Art Survey," IEEE J. Transl. Eng. Health Med., vol. 1, pp. 2100212–2100212, 2013. doi: https://doi.org/10.1109/jtehm.2013.2289879 [Accessed: Sep 21, 2025].

[22] A. Haq, "What Is SOLVEPNP and How Does it Work? - Abdul Haq - Medium," Medium, Sep. 24, 2024. [Online]. Available: https://medium.com/@abdulhaq.ah/what-is-solvepnp-and-how-does-it-work-d9ac70823724 [Accessed: Sep 22, 2025].

[23] E. Kochegurova, E. Kochegurova, and R. Zateev, "Hidden Monitoring Based on Keystroke Dynamics in Online Examination System," Automatic Control and Computer Sciences, vol. 48, no. 6, pp. 385–398, 2022. doi: https://doi.org/10.1134/S0361768822060044 [Accessed: Sep 20, 2025].

[24] "chrome.windows," Chrome for Developers, 2025. [Online]. Available: https://developer.chrome.com/docs/extensions/reference/api/windows [Accessed: Sep 20, 2025].

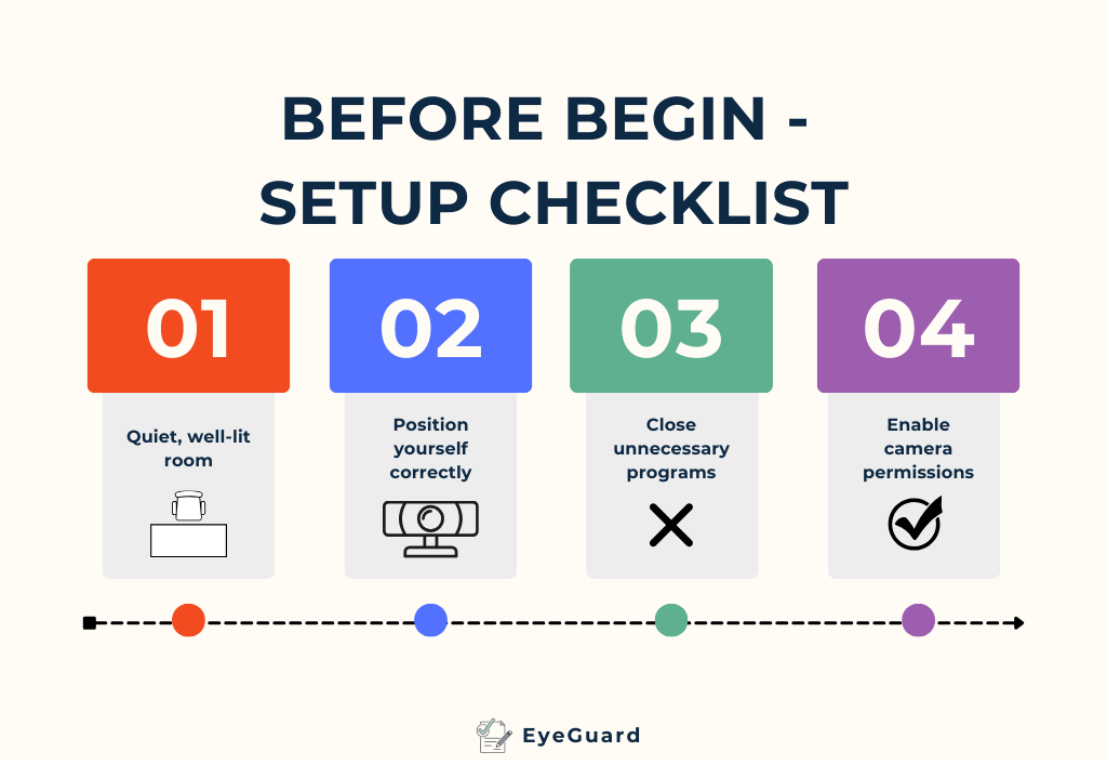
[25] D. G. Balash, D. Kim, D. Shaibekova, R. A. Fainchtein, M. Sherr, and A. J. Aviv, "Examining the Examiners: Students' Privacy and Security Perceptions of Online Proctoring Services," arXiv preprint, arXiv:2106.05917, 2021. [Online]. Available: https://arxiv.org/abs/2106.05917 [Accessed: Sep 22, 2025].

[26] S. Desai, "Comprehensive Survey on Object Detection Taxonomy and Paradigms," Medium, Sep. 24, 2024. [Online]. Available: https://medium.com/@shasvatdesai/comprehensive-survey-on-object-detection-taxonomy-and-paradigms-e83482690ea1

[Accessed: Sep 22, 2025].

APPENDIX A

Guideline




STEP 2: DURING THE EXAM
01
Stay focused on the screen
02
Look directly at each red dot
03
Do not switch tabs or open new windows
04
Keep your camera feed clear
EyeGuard


STEP 3 - END TEST
01
Click "End Proctoring" in the extension
02
You are now good to go
Congrats!
EyeGuard

APPENDIX B

Poster