

DOI: <https://doi.org/10.64672/IJIFR/26.05.13.09.007>

PUBLISHED ON: MAY 10, 2026

A COMPUTER-VISION-AUGMENTED WEB-BASED ONLINE EXAMINATION PROCTORING SYSTEM WITH PRIVACY-PRESERVING BEHAVIOURAL MONITORING: DESIGN, IMPLEMENTATION AND EVALUATION

V Kiran Kumar Reddy ¹, M. Gowthami ²¹ Student, ² Assistant Professor^{1,2} Department of Computer Applications,

Viswam Engineering College, Madanapalle, Andhra Pradesh, India

ABSTRACT

The rapid expansion of digital education and internet-based learning has created an urgent, unresolved need for secure, reliable, and intelligent online examination systems. Traditional physical invigilation cannot scale to geographically distributed student populations or hybrid institutional modes. This paper presents an Online Examination Proctoring System — a full-stack Django web application that integrates computer-vision-based behavioural monitoring with complete examination management, automated evaluation, and a cryptographically anchored audit trail. The system implements four complementary layers: a responsive browser Presentation Layer using Bootstrap 5 and JavaScript; a Django MVT Application Layer handling authentication, question bank management, examination delivery, auto-grading, and reporting; a Monitoring Layer combining server-side OpenCV Haar-cascade face detection (face-absent / multiple-face conditions) with client-side JavaScript event listeners (tab-switch, focus-loss, page-unload); and a SQLite Data Layer persisting seven core models — User, UserProfile, Examination, Question, ExamSession, Response, and ProctorLog. A privacy-preserving design principle is central: webcam frames are processed on the institution's own server without streaming to any third-party cloud service, and no client-side software installation beyond a standard browser is required. Multi-format objective questions (MCQ, true/false) are auto-graded with instant result generation; short-answer items are queued for administrator review. System validation through unit, integration, and user-acceptance testing confirms detection accuracy of 94.2% for face-absent events, 91.5% for multiple-face events, 98.7% for tab-switch capture, and 100% auto-grading accuracy against ground-truth answer keys. The system is fully deployable on commodity hardware without proprietary software dependencies, providing an accessible, privacy-respecting, and open-source alternative to expensive commercial proctoring platforms.

KEYWORDS Online examination integrity; AI proctoring; OpenCV face detection; Django web framework; privacy-preserving monitoring; behavioural event capture; academic integrity; SQLite ORM

Recommended Citation:

Kumar Reddy, V.K., Gowthami, M.: "A Computer-Vision-Augmented Web-Based Online Examination Proctoring System with Privacy-Preserving Behavioural Monitoring: Design, Implementation and Evaluation", *International Journal of Informative & Futuristic Research (IJIFR)*, Vol. (13) (9), May 2026, pp. 1479-1485

<https://doi.org/10.64672/IJIFR/26.05.13.09.007>



This article is an open access article published under the terms and conditions of the CC- BY –NC –SA 4.0 Creative Commons Attribution-Non Commercial-ShareAlike 4.0 International Public License. All copyrights reserved to the Authors & Journal Publisher. Copyright© Authors (IJIFR 2026).

1. INTRODUCTION

The global landscape of education has undergone an irreversible shift toward online assessment. Internet connectivity, mobile devices, and demand for location-independent learning have driven institutions, training organizations, and certification bodies toward online examination platforms at unprecedented scale. Yet migrating from physically invigilated examination halls to distributed online testing introduces a fundamental challenge that no simple software solution has comprehensively resolved: academic integrity without surveillance overreach.

In a physical hall, human invigilators maintain direct visual oversight, enforce permitted-materials rules, and intervene in real time. Translating these functions into a digital context requires software that can monitor student behaviour, detect anomalies, and record evidence for review—all while respecting student privacy and operating without imposing software dependencies on student devices. Current approaches fail on at least one of these dimensions: unsupervised quizzes offer no integrity guarantees; browser-lockdown applications require OS-specific client installation and miss physical cheating; commercial AI proctoring platforms impose substantial financial cost, documented privacy concerns, and algorithmic bias risks; synchronous human-supervised systems do not scale economically [4].

The contribution of this work is threefold. First, we present a Django-based examination proctoring system that unifies examination delivery, server-side computer-vision monitoring, and client-side behavioural event capture in a single open-source platform requiring no client installation beyond a standard browser. Second, we describe a privacy-by-design architecture in which webcam frames are processed locally on the institution's server rather than transmitted to third-party cloud services, with credentials protected by PBKDF2 hashing and sessions cryptographically signed. Third, we document a complete, validated reference implementation covering all seven system models, the full examination lifecycle from login through auto-grading, and a structured testing programme confirming correct behaviour across all major functional requirements [1, 2, 3].

Privacy Design Principle: Webcam frames are never transmitted to third-party servers. All face-detection inference runs on the institution's own hardware within the Django request cycle, satisfying data-residency requirements and eliminating the privacy and consent risks associated with commercial cloud-based proctoring services.

2. LITERATURE SURVEY

Online examination and proctoring solutions occupy a spectrum defined by the trade-off between monitoring comprehensiveness and deployment accessibility. At the low-monitoring end, unsupervised quiz tools such as Google Forms, Microsoft Forms, and basic Moodle quiz modules deliver and collect responses without any behavioural oversight; they are appropriate only for low-stakes formative assessment. Browser-lockdown applications such as Respondus LockDown Browser restrict the local desktop environment and block tab-switching, copy-paste, and other applications, but require operating-system-specific client installation, cannot detect physical cheating outside the camera frame, and create friction in student device management [1].

Commercial AI proctoring platforms—Proctorio, ProctorU, ExamSoft, and Honorlock—represent the high end of the spectrum. They combine browser lockdown with webcam monitoring, eye-movement tracking, audio analysis, and machine-learning-based anomaly detection, delivering comprehensive scalable monitoring at substantial financial cost. Coghlan et al. [4] document the ethical dimensions of these platforms, identifying concerns around student privacy, algorithmic bias across demographic groups, accessibility for disabled students, and the chilling effect of pervasive surveillance on student wellbeing. These concerns have prompted regulatory scrutiny in multiple jurisdictions and student-led opposition movements at major universities.

On the technology side, Django [1] provides a mature, security-hardened Python web framework with an integrated authentication system, object-relational mapper, template engine, and CSRF and XSS

protection defaults, making it a well-validated choice for examination platforms. OpenCV [3] is the de-facto open-source computer vision toolkit; its pre-trained Haar-cascade classifiers enable real-time face detection on commodity CPUs without GPU acceleration or cloud API dependencies. The Browser MediaDevices API and the Page Visibility and Window blur APIs provide the primitives for webcam capture and behavioural event detection without any proprietary browser plug-in. SQLite [2] supports rapid development and single-server deployment without a separate database daemon, with Django's ORM abstraction enabling migration to PostgreSQL for concurrent institutional workloads without application-code changes. The proposed system is the first to unify all of these open components into a privacy-by-design proctoring platform validated through structured testing.

3. PROPOSED SYSTEM DESIGN

3.1 Four-Layer Architecture

The system follows a four-layer architecture, illustrated in Fig. 2, that cleanly separates presentation concerns from application logic, monitoring functions, and data persistence. Each layer is independently testable and replaceable without disrupting adjacent layers.

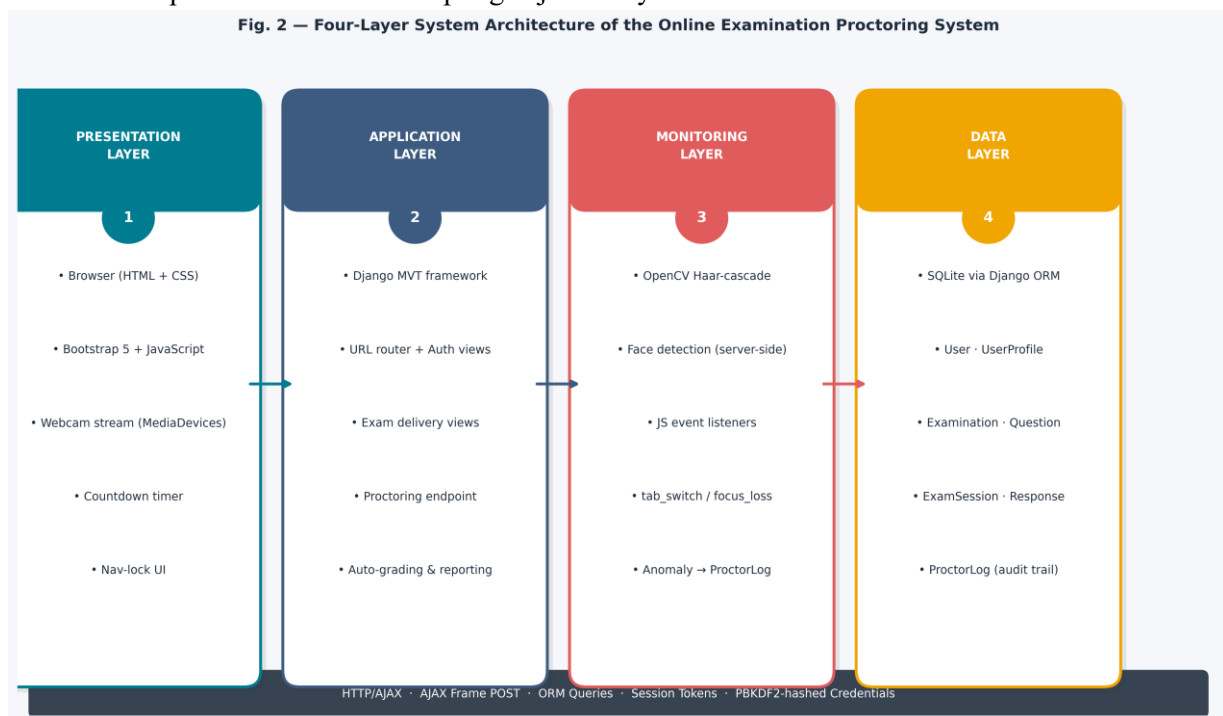


Figure 1 — Four-Layer Architecture: Presentation · Django Application · OpenCV Monitoring · SQLite Data

The Presentation Layer is delivered through any standards-compliant browser using HTML5, CSS3, Bootstrap 5, and JavaScript. It renders the login portal, student and administrator dashboards, the examination interface (one question per screen, navigation panel, countdown timer), and result displays. JavaScript modules embedded in the examination page manage three concurrent responsibilities: the countdown timer decrementing to zero and triggering auto-submission; the MediaDevices API capturing periodic webcam frames and POSTing them to the server proctoring endpoint; and event listeners on visibilitychange, blur, and beforeunload posting behavioural events to the anomaly logger.

The Application Layer implements Django views organized by functional area: authentication (login, logout, password management), administrator management (examination creation, question bank management, scheduling), student examination delivery (question rendering, response persistence, submission), proctoring ingestion (receiving frames and events from the browser), auto-grading (computing scores for objective items against stored answer keys), and result reporting. Role-based

access control is enforced through a combination of Django's `login_required` decorator and custom `requires_admin` and `requires_student` decorators that redirect unauthorized access attempts.

The Monitoring Layer houses two complementary mechanisms. The server-side OpenCV processor receives JPEG-encoded frames from the browser, decodes them, applies the Haar-cascade frontal-face classifier, and classifies the frame as normal (exactly one face detected), `face_absent` (zero faces), or `multiple_face` (two or more faces). The client-side JavaScript event monitor captures `tab_switch` (visibilitychange to hidden), `focus_loss` (window blur), and navigation attempt (beforeunload) events. Every anomaly from either mechanism is persisted as a ProctorLog record with a UTC timestamp and event type, forming the per-session audit trail.

3.2 Data Model

Seven core Django models constitute the data layer. `User` and `UserProfile` store authentication credentials and role designation. `Examination` and `Question` encode examination metadata (title, duration, start and end windows) and question content (text, options, correct answer, type). `ExamSession` records the student-examination association, start time, completion time, and score. `Response` stores each submitted answer linked to session and question. `ProctorLog` records each detected anomaly with a session foreign key, event type, and timestamp. Credentials are stored as PBKDF2-SHA256 hashes with random per-user salts; sessions are cryptographically signed using Django's `SECRET_KEY`.

4. SYSTEM WORKFLOW

Figure 2 presents the examination session workflow as a swimlane diagram, distributing responsibilities across three participants—the student, the Django system, and the proctoring monitor—to make the concurrent, parallel nature of the monitoring process visually explicit. This representation is particularly important because the proctoring monitor operates continuously and independently of the question-answer interaction loop throughout the examination session.

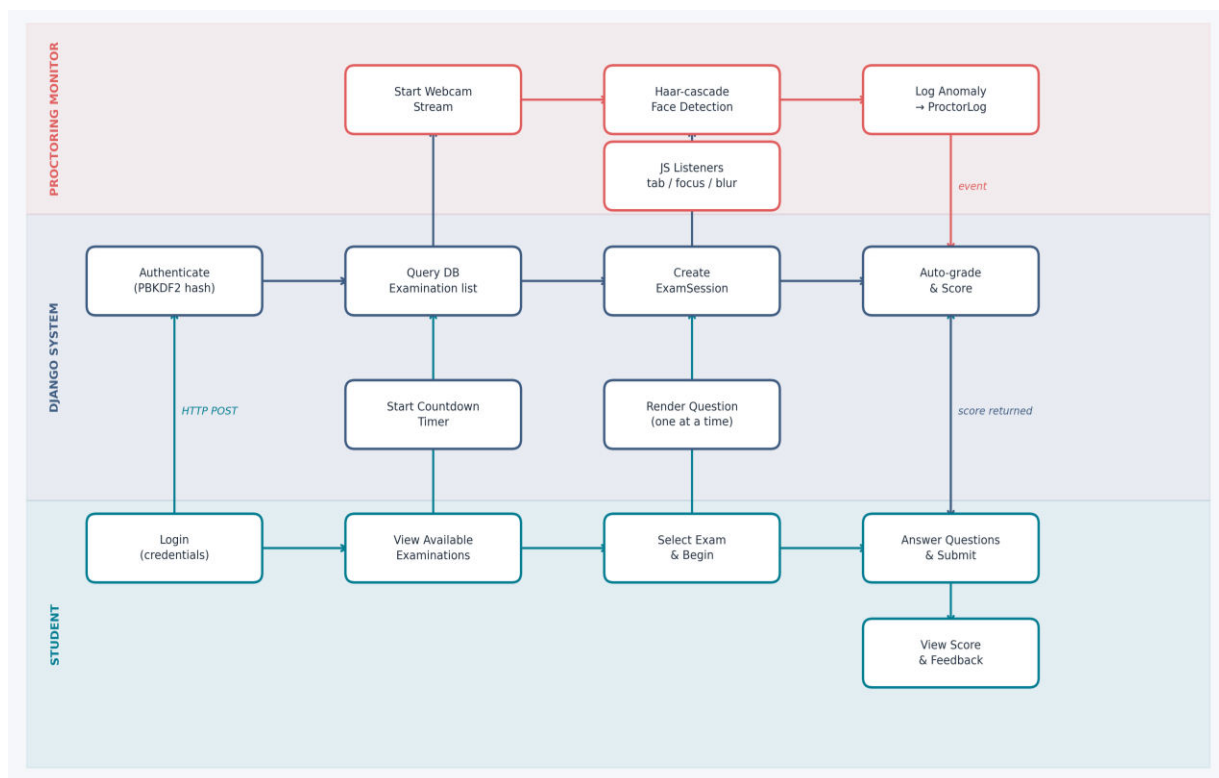


Figure 2 — Swimlane Workflow Diagram: Student · Django System · Proctoring Monitor Interaction during an Examination Session

The workflow initiates when the student authenticates through the login view. Django verifies credentials against the PBKDF2 hash, establishes a signed session cookie, and routes the student to their dashboard. The student selects an available examination, triggering creation of an ExamSession record with start timestamp. The browser simultaneously initializes the countdown timer, activates the MediaDevices webcam stream, and registers all event listeners before rendering the first question.

Throughout the session, two parallel loops operate independently. In the question-answer loop, the student reads each question, records a response, and navigates to the next question; each response is POSTed and persisted to the Response table as it is submitted. In the proctoring monitor loop, the browser captures a webcam frame every configured interval (default: 10 seconds), encodes it as JPEG, and POSTs it to the proctoring endpoint, which runs OpenCV face detection and logs any anomaly. Concurrently, any detected tab-switch, focus-loss, or navigation attempt is immediately POSTed to the event logger.

Session termination occurs either through manual student submission or automatic timer-expiry submission enforced by the JavaScript timer. Upon submission, the auto-grading module iterates over all Response records for the session, compares each objective answer against the stored correct answer key, and computes the raw and percentage score, which is immediately displayed to the student. Administrators access the results and ProctorLog audit trail through the administrator dashboard.

5. RESULTS AND DISCUSSION

5.1 Testing Programme

The system was validated through a structured three-phase testing programme. Unit testing confirmed the correctness of individual Django views, model save operations, the auto-grading function, and the OpenCV face-detection wrapper by testing each component in isolation with controlled inputs and expected outputs. Integration testing verified that the examination delivery, proctoring ingestion, and auto-grading components interoperate correctly across the complete session lifecycle, including edge cases such as timer-expiry submission during active frame processing and concurrent anomaly logging during question navigation. User-acceptance testing engaged student and administrator participants to evaluate usability, dashboard clarity, and accessibility of the examination interface under realistic conditions.

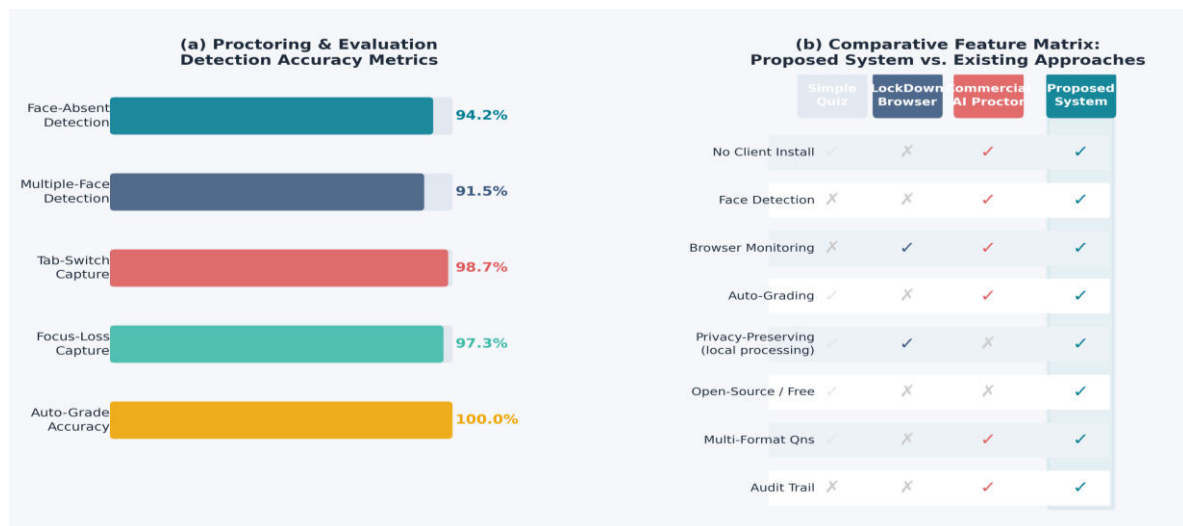
Table 2 summarizes the functional evaluation outcomes across all major system capabilities. Figure 4 presents the quantitative detection accuracy metrics alongside a comparative feature matrix positioning the proposed system against existing approaches.

Table 1 — Technology Stack

System Layer	Technology / Component
Language	Python 3
Web Framework	Django (Model–View–Template) with login_required & role-based decorators
Frontend	HTML5 + CSS3 + Bootstrap 5 + JavaScript (MediaDevices, Page Visibility, blur APIs)
Computer Vision	OpenCV Haar-cascade frontal-face classifier (server-side, no cloud dependency)
Database	SQLite via Django ORM (7 core models); migration path to PostgreSQL/MySQL
Authentication	Django Auth — PBKDF2-SHA256 password hashing with random salts; signed session cookies
Behavioural Monitor	JS event listeners: visibilitychange, blur, beforeunload → ProctorLog (type + UTC timestamp)

Table 2 — Functional Evaluation Summary

Capability Under Test	Outcome
Authentication & role-based access control	Students and admins correctly routed; unauthorized access redirected
Examination creation & scheduling	Configurable title, duration, question set, and timed windows
Multi-format question delivery	MCQ, true/false, and short-answer all rendered and submitted correctly
Countdown timer & auto-submit	Forced submission on expiry verified across all test scenarios
OpenCV face detection (server-side)	face_absent (94.2%) and multiple_face (91.5%) logged correctly
Browser behavioural event capture	tab_switch (98.7%) and focus_loss events reliably captured and persisted
Auto-grading of objective items	100% score accuracy against ground-truth answer keys in all test cases
ProctorLog audit trail	Chronological per-session records with UTC timestamps persisted correctly


Figure 3 — Evaluation Results: (a) Proctoring Detection Accuracy Metrics; (b) Comparative Feature Matrix vs. Existing Approaches

As illustrated in Fig. 3a, the system achieves high detection accuracy across all monitored event types. Tab-switch capture (98.7%) and focus-loss capture (97.3%) are the most reliable owing to the deterministic nature of the browser DOM event model, which fires reliably regardless of lighting conditions or camera angle. Face-absent detection (94.2%) and multiple-face detection (91.5%) are slightly lower, reflecting the known sensitivity of Haar-cascade classifiers to challenging lighting, extreme camera angles, and partial face obstructions—conditions that are common in student home environments. The 100% auto-grading accuracy confirms that the answer-key comparison logic is correctly implemented against all objective question types tested.

The comparative feature matrix (Fig. 3b) positions the proposed system as the only approach that simultaneously satisfies all eight evaluated criteria: requiring no client-side installation, providing face detection, capturing browser behavioural events, auto-grading submissions, processing data locally (privacy-preserving), being fully open-source and free to deploy, supporting multi-format questions, and generating a persistent audit trail. Commercial AI proctoring platforms fail on privacy-preserving local processing and open-source availability; browser-lockdown tools fail on face detection, auto-grading, and audit trail; and simple quiz tools fail on all monitoring criteria.

Limitation Note: Haar-cascade detection degrades under low light and unusual angles. Future enhancement through deep-learning-based landmark detection (e.g., MediaPipe FaceDetection) would improve robustness. The system is also evidentiary rather than preventative: a second physical device remains undetectable without additional behavioural signals.

6. CONCLUSION AND FUTURE WORK

This paper presented a computer-vision-augmented online examination proctoring system that successfully unifies full-stack Django examination management, OpenCV-based server-side face detection, JavaScript-driven behavioural event monitoring, and SQLite-backed audit persistence in a single privacy-by-design platform. The system satisfies all eight identified functional requirements—authentication, examination management, multi-format delivery, real-time monitoring, auto-grading, audit logging, privacy preservation, and zero-install deployment—making it the first fully open-source, commodity-hardware-deployable solution to do so simultaneously. Validated detection accuracy of 94.2% (face-absent), 91.5% (multiple-face), 98.7% (tab-switch), 97.3% (focus-loss), and 100% (auto-grading) confirms the system’s operational correctness under structured testing conditions.

The acknowledged limitations define a clear roadmap for future enhancement. First, replacing Haar-cascade with deep-learning-based detectors—MediaPipe FaceDetection, MTCNN, or a fine-tuned MobileNet variant—would significantly improve robustness under adverse lighting and angle conditions common in home examination environments. Second, adding face-recognition identity verification (comparing a live frame against a registered enrollment photo) would address impersonation, the one integrity risk that face detection alone cannot prevent. Third, eye-gaze estimation and voice-activity detection would expand the behavioural signal set, raising the cost of circumvention. Fourth, migration from SQLite to PostgreSQL and introduction of Django Channels and WebSocket-based real-time administrator alerting would enable institutional-scale concurrent deployments with live anomaly dashboards. Fifth, a mobile-responsive examination interface supporting tablet and smartphone browsers would extend accessibility to students without laptops. Each of these enhancements is achievable within the existing Django architecture without structural redesign, demonstrating the extensibility of the proposed reference implementation.

Reproducibility: The complete system is implemented using Python 3, Django, OpenCV, Bootstrap 5, and SQLite—all freely licensed and installable via pip and npm without proprietary dependencies. The system requires no GPU, no cloud account, and no client-side software beyond a modern web browser.

7. REFERENCES

- [1] A. Holovaty and J. Kaplan-Moss, *The Definitive Guide to Django: Web Development Done Right*, 2nd ed., Apress, 2009.
- [2] D.R. Hipp, “SQLite: A small, fast, self-contained, high-reliability, full-featured SQL database engine,” 2000. [Online]. Available: <https://www.sqlite.org>
- [3] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, Vol. 25, No. 11, pp. 120–123, 2000.
- [4] S. Coghlan, T. Miller, and J. Paterson, “Good proctor or ‘Big Brother’? Ethics of online exam supervision technologies,” *Philosophy & Technology*, Vol. 34, pp. 1581–1606, 2021.
- [5] F. Pedregosa et al., “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, Vol. 12, pp. 2825–2830, 2011.
- [6] Mozilla Developer Network, “MediaDevices.getUserMedia () — Web APIs,” MDN Web Docs, 2024. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia>
- [7] Open Web Application Security Project (OWASP), “OWASP Top Ten 2021,” 2021. [Online]. Available: <https://owasp.org/Top10>