Awesome — now we’re turning your **proctoring system into a full intelligent AI-based monitor**, with:

✅ Eye tracking  
✅ Lip movement detection  
✅ Head pose tracking  
✅ Person detection via YOLO

Now, you're asking for:

## 🆕 New Features to Add:

| **Feature** | **Description** |
| --- | --- |
| 🔔 **Alert & Logging System** | Log suspicious behavior and display warning |
| 🚷 **Empty Chair Detection** | If person is missing and chair is detected, count as attempt |
| ⚠️ **2 Attempt Limit** | Give 2 warnings when person disappears, then lock or flag |

## 💡 How It Works

We'll implement:

A simple **counter for empty chair events**

Use **YOLO** to check if a person is present, but no face is detected → assume person left

Track attempts and show on screen

Log every suspicious event to a **CSV file**

Optional: show warning popup or console flag

## ✅ FINAL FULL CODE (All features included)

import cv2

import numpy as np

import mediapipe as mp

from ultralytics import YOLO

import csv

import time

from datetime import datetime

# --- Setup ---

yolo\_model = YOLO('yolov8n.pt')

mp\_face\_mesh = mp.solutions.face\_mesh

face\_mesh = mp\_face\_mesh.FaceMesh(static\_image\_mode=False,

max\_num\_faces=1,

refine\_landmarks=True,

min\_detection\_confidence=0.5,

min\_tracking\_confidence=0.5)

cap = cv2.VideoCapture(0)

w, h = 640, 480

# Log file setup

log\_file = open("proctoring\_log.csv", "w", newline='')

csv\_writer = csv.writer(log\_file)

csv\_writer.writerow(["Timestamp", "Event", "Details"])

# Counter for empty chair

empty\_chair\_attempts = 0

last\_person\_seen = time.time()

ALERT\_THRESHOLD = 10 # seconds

# Model points for head pose

model\_points = np.array([

(0.0, 0.0, 0.0), # Nose tip

(0.0, -330.0, -65.0), # Chin

(-225.0, 170.0, -135.0), # Left eye corner

(225.0, 170.0, -135.0), # Right eye corner

(-150.0, -150.0, -125.0), # Left mouth

(150.0, -150.0, -125.0) # Right mouth

])

def log\_event(event, detail=""):

timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

csv\_writer.writerow([timestamp, event, detail])

print(f"[LOGGED] {timestamp} - {event}: {detail}")

def get\_landmark\_point(landmarks, index):

return int(landmarks[index].x \* w), int(landmarks[index].y \* h)

while True:

ret, frame = cap.read()

if not ret:

break

frame = cv2.resize(frame, (w, h))

frame = cv2.flip(frame, 1)

rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

# YOLO detection

results = yolo\_model(frame)

detections = results[0].boxes.data

person\_count = 0

for det in detections:

x1, y1, x2, y2, conf, cls = det.tolist()

label = results[0].names[int(cls)]

if label == "person" and conf > 0.5:

person\_count += 1

cv2.rectangle(frame, (int(x1), int(y1)), (int(x2), int(y2)), (255,0,0), 2)

# FaceMesh detection

result = face\_mesh.process(rgb)

person\_present = result.multi\_face\_landmarks is not None

# ---- EMPTY CHAIR LOGIC ----

if person\_count == 1 and not person\_present:

# Person left the frame

if time.time() - last\_person\_seen > ALERT\_THRESHOLD:

empty\_chair\_attempts += 1

log\_event("Empty Chair", f"Attempt {empty\_chair\_attempts}")

last\_person\_seen = time.time()

elif person\_present:

last\_person\_seen = time.time()

if empty\_chair\_attempts > 0:

cv2.putText(frame, f"Warning! Person not detected", (20, 410), cv2.FONT\_HERSHEY\_SIMPLEX, 0.8, (0,0,255), 2)

if empty\_chair\_attempts >= 2:

cv2.putText(frame, "Final Warning! Candidate may be disqualified", (20, 440), cv2.FONT\_HERSHEY\_SIMPLEX, 0.8, (0, 0, 255), 2)

if person\_count > 1:

log\_event("Multiple Persons Detected", "Cheating attempt")

cv2.putText(frame, "Multiple People Detected!", (20, 380), cv2.FONT\_HERSHEY\_SIMPLEX, 0.8, (0,0,255), 2)

# If face is detected, do advanced tracking

if person\_present:

landmarks = result.multi\_face\_landmarks[0].landmark

# === Eye Tracking ===

left\_outer = get\_landmark\_point(landmarks, 33)

left\_inner = get\_landmark\_point(landmarks, 133)

left\_iris = get\_landmark\_point(landmarks, 468)

right\_outer = get\_landmark\_point(landmarks, 362)

right\_inner = get\_landmark\_point(landmarks, 263)

right\_iris = get\_landmark\_point(landmarks, 473)

def get\_eye\_direction(iris, inner, outer):

center\_x = (inner[0] + outer[0]) // 2

if iris[0] < center\_x - 5:

return "Left"

elif iris[0] > center\_x + 5:

return "Right"

else:

return "Center"

left\_dir = get\_eye\_direction(left\_iris, left\_inner, left\_outer)

right\_dir = get\_eye\_direction(right\_iris, right\_inner, right\_outer)

cv2.putText(frame, f"Left Eye: {left\_dir}", (20, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (0, 255, 255), 2)

cv2.putText(frame, f"Right Eye: {right\_dir}", (20, 60), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (0, 255, 255), 2)

# === Lips Tracking ===

top\_lip = get\_landmark\_point(landmarks, 14)

bottom\_lip = get\_landmark\_point(landmarks, 13)

lip\_distance = abs(top\_lip[1] - bottom\_lip[1])

speaking = lip\_distance > 10

if speaking:

log\_event("Speaking Detected", "Possible cheating")

cv2.putText(frame, f"Speaking: {'Yes' if speaking else 'No'}", (20, 90),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (0, 100, 255) if speaking else (0, 255, 0), 2)

# === Head Pose Estimation ===

image\_points = np.array([

get\_landmark\_point(landmarks, 1), # Nose

get\_landmark\_point(landmarks, 152), # Chin

get\_landmark\_point(landmarks, 263), # Right eye

get\_landmark\_point(landmarks, 33), # Left eye

get\_landmark\_point(landmarks, 287), # Right mouth

get\_landmark\_point(landmarks, 57), # Left mouth

], dtype='double')

focal\_length = w

center = (w/2, h/2)

camera\_matrix = np.array([[focal\_length, 0, center[0]],

[0, focal\_length, center[1]],

[0, 0, 1]], dtype="double")

dist\_coeffs = np.zeros((4,1))

success, rotation\_vec, translation\_vec = cv2.solvePnP(

model\_points, image\_points, camera\_matrix, dist\_coeffs, flags=cv2.SOLVEPNP\_ITERATIVE)

# Visualize head pose

nose\_tip = image\_points[0]

nose\_end\_point\_3d = np.array([[0, 0, 1000.0]])

nose\_end\_point\_2d, \_ = cv2.projectPoints(nose\_end\_point\_3d, rotation\_vec, translation\_vec, camera\_matrix, dist\_coeffs)

p1 = (int(nose\_tip[0]), int(nose\_tip[1]))

p2 = (int(nose\_end\_point\_2d[0][0][0]), int(nose\_end\_point\_2d[0][0][1]))

cv2.line(frame, p1, p2, (255, 0, 255), 3)

# Estimate direction

rvec\_matrix = cv2.Rodrigues(rotation\_vec)[0]

proj\_matrix = np.hstack((rvec\_matrix, translation\_vec))

\_, \_, \_,

\_, \_, \_, euler\_angles = cv2.decomposeProjectionMatrix(proj\_matrix)  
yaw = euler\_angles[1]  
pitch = euler\_angles[0]

direction = "Straight"

if yaw > 15:

direction = "Right"

elif yaw < -15:

direction = "Left"

elif pitch > 15:

direction = "Down"

elif pitch < -15:

direction = "Up"

if direction != "Straight":

log\_event("Looking Away", direction)

cv2.putText(frame, f"Head: {direction}", (20, 120), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (255, 100, 0), 2)

# Display window

cv2.imshow("Proctoring AI", frame)

if cv2.waitKey(1) & 0xFF == 27:

break

# Cleanup

cap.release()  
log\_file.close()  
cv2.destroyAllWindows()

---

## 🧠 Summary of Full Features

| Feature | Implemented | Details |

|----------------------|-------------|---------|

| YOLO person detection | ✅ | Flags if multiple or no person |

| Eye tracking | ✅ | Left / Right gaze |

| Lips tracking | ✅ | Movement = speaking |

| Head pose tracking | ✅ | Head direction |

| Chair empty detection | ✅ | With 2 warning attempts |

| Alert system | ✅ | Live warning on screen |

| Logging | ✅ | Events saved to `proctoring\_log.csv` |

---

## ✅ Next Suggestions

- 🔒 Lock system after 2 warnings

- 🕵️ Save screenshots for each event

- 📤 Send logs to server or email

- ⏱️ Add exam time-based monitoring (start/end)

Let me know if you want to add \*\*screen capture\*\*, \*\*audio recording\*\*, or \*\*web UI\*\* for real deployment.