

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN



**Ứng dụng theo dõi và tư vấn sức khỏe
kỹ thuật số**

Lê Tiến Mạnh

Trần Minh Đức

Hoàng Văn Phú

Mã học phần: MAT3508

Học kỳ 1, Năm học 2025-2026

Thông tin Dự án

- Học phần:** MAT3508 – Nhập môn Trí tuệ Nhân tạo
- Học kỳ:** Học kỳ 1, Năm học 2025-2026
- Trường:** VNU-HUS (Đại học Quốc gia Hà Nội – Trường Đại học Khoa học Tự nhiên)
- Tên dự án:** AEye Pro
- Ngày nộp:** 30/11/2025
- Báo cáo PDF:** Báo cáo PDF
- Slide thuyết trình:** Slide thuyết trình
- Kho GitHub:** https://github.com/tmanhococ/Final_AI_Project

Thành viên nhóm

Họ tên	Mã sinh viên	Tên GitHub	Đóng góp
Lê Tiến Mạnh	23001535	tmanhococ	Leader, Chatbot Developer
Trần Minh Đức	23001518	tranminhduc9	Frontend Developer, Deploy Server
Hoàng Văn Phú	23001546	phuhoanggg	Computer Vision Developer

Danh sách hình vẽ

2.1	Minh họa kiến trúc RAG truyền thống	16
2.2	Minh họa kiến trúc Adaptive RAG	17
3.1	Giao diện người dùng	26
3.2	Tổng quan về luồng Thị giác	28
3.3	Tổng quan về luồng Thị giác	29
4.1	Nhận diện lưới khuôn mặt trong điều kiện ánh sáng bình thường	35
4.2	Cảnh báo khoảng cách tới màn hình quá gần	35
4.3	Cảnh báo khoảng cách tới màn hình quá xa	36
4.4	Các trường hợp hệ thống phát hiện sai tư thế	37
4.5	Cảnh báo buồn ngủ	37
4.6	Dữ liệu thời gian thực và Điều chỉnh cài đặt	38
4.7	Câu trả lời và luồng hoạt động với những câu hỏi xã giao	39
4.8	Câu trả lời và luồng hoạt động với những câu hỏi truy xuất kiến thức	40
4.9	Chatbot phân tích dữ liệu và đưa ra lời khuyên cá nhân hóa	41
4.10	Luồng hoạt động phân tích Dữ liệu cá nhân	41

Danh sách bảng

2.1	Bảng Tổng hợp so sánh Mediapipe và một số công cụ phổ biến	9
2.2	So sánh tóm tắt giữa RAG Truyền thống (Naive) và Adaptive RAG (Đề xuất) . . .	18

Mục lục

1 Giới thiệu	6
1.1 Tóm tắt	6
1.2 Bài toán đặt ra	6
2 Phương pháp tiếp cận	7
2.1 Cơ sở lựa chọn công nghệ: Sự cân bằng giữa hiệu năng và độ chính xác	7
2.1.1 MediaPipe: Giải pháp cân bằng toàn diện	8
2.1.2 OpenCV: Cơ chế nền tảng và tiền xử lý dữ liệu	12
2.2 Cơ chế hoạt động	13
2.3 Các thông số đánh giá	13
2.3.1 Tỷ số khung hình mắt (Eye Aspect Ratio - EAR)	13
2.3.2 Độ lệch trực vai (Shoulder Alignment)	14
2.3.3 Góc gập cổ (Head Pitch)	14
2.3.4 Góc nghiêng đầu (Head Roll)	15
2.3.5 Các thông số khác	15
2.4 Adaptive RAG Chatbot	15
2.4.1 Sự Tiến Hóa Kiến Trúc: Từ RAG Truyền Thống đến Adaptive RAG	15
3 Triển khai & Phân tích	20
3.1 Triển khai Tổng thể Hệ thống	20
3.1.1 Kiến trúc Tổng thể Hệ thống	20
3.1.2 Thiết kế Backend và API	21
3.1.3 Thiết kế Giao diện Người dùng	25
3.2 Triển khai Hệ thống Thị giác thời gian thực	27
3.2.1 Kích hoạt	27
3.2.2 Thu nhận và xử lý dữ liệu	27
3.2.3 Tính toán và phân luồng	27
3.2.4 Lưu tổng quan dữ liệu của phiên	27
3.3 Triển Khai Hệ Thống Chatbot	28
3.3.1 Kiến Trúc Tổng Thể và Cơ Chế Điều Phối	28

3.3.2	Chi Tiết Các Node Xử Lý Logic	29
3.3.3	Tổng Kết Các Luồng Xử Lý Chính (Key Workflows)	32
4	Kết quả	34
4.1	Kết quả thực nghiệm hệ thống thị giác máy tính	34
4.1.1	Khả năng nhận diện và theo dõi thời gian thực	34
4.1.2	Giám sát khoảng cách hợp lý	35
4.1.3	Phân tích và cảnh báo tư thế	36
4.1.4	Phát hiện dấu hiệu buồn ngủ	37
4.1.5	Bảng điều khiển thông số thời gian thực	38
4.2	Kết quả thực nghiệm hệ thống Chatbot (Adaptive RAG)	38
4.2.1	Luồng giao tiếp xã hội	39
4.2.2	Luồng truy xuất kiến thức Y khoa	39
4.2.3	Luồng phân tích Dữ liệu cá nhân hóa	40
4.2.4	Phân tích	41
5	Kết luận	42
5.1	Những kết quả đạt được	42
5.1.1	Về mặt kỹ thuật	42
5.1.2	Về sản phẩm	42
5.1.3	Ứng dụng thực tế	43
5.1.4	Bài học kinh nghiệm và những khó khăn	43
5.2	Hướng phát triển	44
5.2.1	Mục tiêu ngắn hạn	44
5.2.2	Mục tiêu dài hạn	45
5.2.3	Phương pháp thực hiện	45

Chương 1

Giới thiệu

1.1 Tóm tắt

Việc làm việc lâu trước màn hình máy tính là thực tế phổ biến trong học tập và công việc hiện đại, nhưng kéo theo nhiều vấn đề sức khỏe như mỏi mắt, đau cổ – vai – lưng và giảm tập trung. Dự án **AEYEPro** được phát triển nhằm theo dõi và nâng cao sức khỏe người dùng trong môi trường số, thông qua việc ghi nhận các chỉ số sinh lý và hành vi, bao gồm tần suất chớp mắt, góc cổ và độ nghiêng đầu.

Hệ thống cung cấp cảnh báo kịp thời khi phát hiện dấu hiệu mệt mỏi, đồng thời lưu trữ dữ liệu để phân tích dài hạn. Chatbot tích hợp dựa trên kiến trúc **Retrieval-Augmented Generation (RAG)** có khả năng truy vấn lịch sử, đưa ra thống kê, nhận định và gợi ý cá nhân hóa dựa trên thói quen thực tế của người dùng. Về mặt kỹ thuật, AYEPro ứng dụng **openCV** cho xử lý hình ảnh và **mediapipe** cho việc lấy các điểm dữ liệu trên cơ thể người, **LangChain**, **LangGraph** kết hợp **Chroma Vector Store** cho **Adaptive RAG** và **Gemini API** cho mô hình ngôn ngữ, triển khai dưới dạng ứng dụng web để đảm bảo hiệu suất, bảo mật và vận hành liên tục.

1.2 Bài toán đặt ra

Trong thực tế, nhiều người gặp khó khăn trong việc duy trì tư thế đúng và kiểm soát dấu hiệu mệt mỏi mắt khi làm việc lâu với màn hình. Giải pháp hiện tại thường chỉ cung cấp cảnh báo đơn giản, thiếu khả năng phân tích dữ liệu lịch sử và gợi ý cá nhân hóa.

AEYEPro hướng tới việc khắc phục những hạn chế này bằng một hệ thống theo dõi thời gian thực. Hệ thống ghi nhận dữ liệu sinh lý và hành vi, phân tích tư thế và mức độ mệt mỏi, đồng thời cung cấp thông tin phản hồi và lời khuyên được cá nhân hóa dựa trên dữ liệu thu thập được. Cách tiếp cận này không chỉ giúp người dùng phát hiện sớm các vấn đề sức khỏe mà còn hỗ trợ điều chỉnh thói quen làm việc, nâng cao hiệu quả và giảm nguy cơ các rối loạn cơ xương khớp hoặc thị giác khi phải tiếp xúc liên tục với màn hình máy tính.

Chương 2

Phương pháp tiếp cận

Để giải quyết bài toán theo dõi và tư vấn sức khỏe toàn diện cho người dùng máy tính, đề tài áp dụng phương pháp tiếp cận hệ thống dựa trên sự phân tách module, chia nhỏ vấn đề thành hai bài toán con cốt lõi có mối quan hệ nhân quả:

1. **Bài toán 1: Giám sát và Số hóa hành vi thời gian thực.** Giai đoạn này sử dụng các kỹ thuật thị giác máy tính (Computer Vision) để nhận diện liên tục các chỉ số sinh trắc học của người dùng (như tần suất心跳, ngáp ngáp) thông qua webcam. Dòng dữ liệu này sau đó được chuẩn hóa và lưu trữ vào cơ sở dữ liệu cấu trúc (Structured Database/CSV) nhằm hình thành nhật ký sức khỏe cá nhân theo thời gian thực.
2. **Bài toán 2: Xây dựng Trợ lý ảo tư vấn sức khỏe (Health Care Chatbot).** Tận dụng nguồn dữ liệu từ bài toán 1, hệ thống triển khai kiến trúc Adaptive RAG để xử lý truy vấn của người dùng. Tại đây, Chatbot thực hiện cơ chế tổng hợp thông tin đa nguồn: vừa thống kê, phân tích số liệu lịch sử người dùng, vừa truy xuất tri thức y khoa chuẩn hóa từ cơ sở dữ liệu vector.

Cách tiếp cận này cho phép chuyển hóa dữ liệu thô thành thông tin tư vấn có giá trị, đảm bảo lời khuyên đưa ra vừa chính xác về mặt y lý, vừa được cá nhân hóa sâu sắc theo tình trạng thực tế của người dùng.

2.1 Cơ sở lựa chọn công nghệ: Sự cân bằng giữa hiệu năng và độ chính xác

Trong bài toán xây dựng hệ thống giám sát sức khỏe người dùng máy tính, việc lựa chọn công nghệ không đơn thuần là tìm kiếm thuật toán có độ chính xác tuyệt đối cao nhất, mà là tìm ra điểm cân bằng tối ưu (trade-off) giữa ba yếu tố: độ tin cậy của dữ liệu, tốc độ xử lý thời gian thực và khả năng triển khai trên phần cứng phổ thông. Dựa trên các phân tích so sánh định lượng mới nhất, hệ thống lựa chọn sự kết hợp giữa MediaPipe và OpenCV.

2.1.1 MediaPipe: Giải pháp cân bằng toàn diện

Mediapipe không phải là mô hình có độ chính xác cao nhất trên mọi thước đo, nhưng nó là giải pháp toàn diện, dễ tiếp cận và triển khai trong các dự án liên quan đến thị giác máy tính trong đa số các tình huống thực tế.

Mediapipe và một số công cụ khác

Bảng 2.1: Bảng Tổng hợp so sánh Mediapipe và một số công cụ phổ biến

Tiêu chí / Công cụ	MediaPipe	YOLOv5-Pose	MoveNet (Thunder)	OpenPose
Kiến trúc cốt lõi	Top-down (ROI-based) <i>Detect Face/Body</i> → <i>Landmarks</i>	Single-stage Object Detection (YOLOv5) kèm <i>Keypoint Head</i>	Bottom-up Style <i>CenterNet-based</i> (Regression from center)	Bottom-up Multi-stage CNN (Part Affinity Fields)
Độ chính xác tư thế	93.5% (Accuracy sau lọc) [2] Ôn định cho 1 người	Cao (Dùng module CBAM) Tốt hơn Mediapipe trong phân tích tầm vận động (ROM) [3]	72.0+ mAP (COCO) Rất cao trên các tư thế thể thao/fitness	Rất cao Chuẩn mực (Benchmark) cho video nhiều [4]
Độ chính xác Mắt/Gaze	91.3% (Phân loại vùng) [1] Kết hợp Kalman Filter	Không hỗ trợ gốc	Không hỗ trợ	Không chuyên dụng
Tối ưu	30+ FPS (CPU i7) Latency: 25–32ms [1]	Real-time trên GPU 95 FPS (với TensorRT)	60+ FPS (Mobile/Web) Tối ưu hóa cực tốt	Thấp trên CPU Cần GPU rời để Real-time
Khả năng nhận diện các điểm mốc	Rất cao 33 Pose + 468 Face + Iris	Trung bình 17 Keypoints (COCO)	Trung bình 17 Keypoints	Cao Body + Face + Hand
Chống nhiễu/che khuất	Tốt nhờ Kalman Filter [1]	Xuất sắc Ít bị jitter	Trung bình	Cao Xử lý tốt khi bị che khuất
Tài nguyên	Thấp (CPU/Web) Phù hợp App cá nhân	Trung bình/Cao (GPU) Phù hợp Server/PC mạnh	Rất thấp (Edge/Mobile) TensorFlow Lite	Rất cao (VRAM lớn)

Để làm rõ các so sánh trong Bảng 2.1, dưới đây là giải thích chi tiết các thuật ngữ chuyên môn:

Top-down Approach	Quy trình 2 bước: (1) Tìm khung bao (bounding box) chứa người; (2) Tìm điểm khớp trong khung đó. Độ chính xác cao cho cá nhân (như MediaPipe), nhưng tốc độ giảm khi số lượng người tăng.
Bottom-up Approach	Quy trình phát hiện tất cả điểm khớp trong ảnh trước, sau đó nối chúng lại thành các bộ xương. Tốc độ ổn định bất kể số lượng người (như OpenPose, MoveNet).
ROI-based (Region of Interest)	Kỹ thuật "Vùng quan tâm" (dùng trong MediaPipe). Hệ thống dự đoán vị trí người ở frame tiếp theo và chỉ xử lý vùng đó thay vì toàn bộ ảnh, giúp tiết kiệm tài nguyên CPU.
CenterNet-based	Kiến trúc cốt lõi của **MoveNet**. Thay vì dùng khung bao, nó coi con người là một điểm tâm (center point) và từ đó tính toán ra vị trí các khớp. Cách này giúp MoveNet chạy rất nhanh trên thiết bị di động.
CBAM (Convolutional Block Attention Module)	Một mô-đun "cơ chế sự chú ý" được thêm vào mạng YOLO (trong nghiên cứu của Zhang 2024). Nó giúp mô hình tập trung vào các đặc trưng quan trọng (khớp xương) và bỏ qua nhiễu nền.
Multi-stage CNN	Mạng nơ-ron nhiều giai đoạn (OpenPose). Giai đoạn 1 tạo bản đồ nhiệt (Heatmaps), Giai đoạn 2 tạo trường liên kết (Part Affinity Fields). Rất chính xác nhưng tốn tài nguyên phần cứng.
Single-stage	Kiến trúc mạng (như YOLO-Pose) thực hiện phát hiện người và hồi quy điểm khớp trong cùng một lần truyền thẳng (forward pass), giúp đạt tốc độ thời gian thực (Real-time).
Keypoint Regression	Phương pháp tính toán trực tiếp tọa độ (x, y) của khớp dưới dạng số học, thay vì phải quét bản đồ nhiệt (heatmap). Đây là lý do YOLO và MoveNet có tốc độ cao.
Kalman Filter	Thuật toán lọc nhiễu toán học. Trong MediaPipe, nó giúp dự đoán và làm mượt chuyển động, giảm hiện tượng các điểm khớp bị rung (jitter) khi camera rung lắc.
mAP (mean Average Precision)	"Độ chính xác trung bình". Chỉ số chuẩn để đo lường độ chính xác của AI. Điểm mAP càng cao (max 100%) thì mô hình dự đoán càng chuẩn xác so với thực tế.

COCO (Common Objects in Context)	Bộ dữ liệu chuẩn mực quốc tế dùng để huấn luyện và chấm điểm các mô hình AI nhận diện vật thể và tư thế người.
SOTA (State-of-the-Art)	Thuật ngữ chỉ công nghệ hoặc mô hình "tiên tiến nhất", đạt kết quả tốt nhất hiện nay trên các bảng xếp hạng (ví dụ: YOLOv7/v8 thường được gọi là SOTA về tốc độ).
Jitter (Rung nhiễu)	Hiện tượng các điểm khớp (keypoints) bị dao động liên tục quanh vị trí thực tế, ngay cả khi chủ thể đứng yên. Nguyên nhân do nhiễu cảm biến camera hoặc độ không chắc chắn của mô hình AI giữa các khung hình (frames). Jitter làm giảm độ chính xác khi đo góc khớp.
ROM (Range of Motion)	"Tầm vận động". Là biên độ chuyển động (góc do bằng độ) tối đa mà một khớp xương có thể thực hiện được. Trong y tế và phục hồi chức năng, việc đo chính xác ROM (như nghiên cứu của Zhang 2024 dùng YOLOv5) là tiêu chí vàng để đánh giá mức độ hồi phục của bệnh nhân.
TensorRT	Công cụ tối ưu hóa của NVIDIA, giúp biên dịch lại mô hình AI để chạy siêu tốc trên card đồ họa (GPU) NVIDIA (tăng FPS cho YOLO).

Giới hạn kỹ thuật và độ chính xác

Một tính năng vượt trội của MediaPipe Pose là khả năng cung cấp *World Landmarks*. Tuy nhiên, cần đánh giá khách quan về sai số của phương pháp suy luận 3D từ ảnh 2D này. Theo công bố chính thức từ nhóm phát triển Google Research (Bazarevsky *et al.*, 2020) [7], khi đánh giá trên tập dữ liệu chuẩn hóa (Yoga dataset), sai số vị trí khớp trung bình (Mean Per Joint Position Error – MPJPE) dao động trong khoảng 35–56 **mm** tùy thuộc vào phiên bản mô hình sử dụng (Lite, Full hay Heavy).

Mặc dù mức sai số này (xấp xỉ 3.5–5.6 cm) chưa đáp ứng được các tiêu chuẩn khắt khe trong chẩn đoán lâm sàng y khoa (thường yêu cầu sai số < 10 mm), nhưng đối với bài toán giám sát hành vi và cảnh báo tư thế sai lệch trong môi trường văn phòng, đây là ngưỡng chấp nhận được. Dựa trên thang đo Ecgonomi RULA [8], các mức đánh giá rủi ro thường cách nhau bởi biên độ góc 10°–20°. Do đó, sai số vị trí nêu trên vẫn đảm bảo độ tin cậy để hệ thống phân loại đúng các trạng thái tư thế của người dùng khi sử dụng máy tính.

Khả năng tiếp cận dễ dàng

Việc lựa chọn MediaPipe còn mang ý nghĩa về khả năng tiếp cận. Trước đây, việc phân tích chuyển động chính xác đòi hỏi các hệ thống Motion Capture đắt tiền hoặc GPU mạnh mẽ. MediaPipe đã đóng vai trò tiên phong trong việc phổ cập công nghệ này, cho phép thực thi các thuật toán phức tạp ngay trên CPU của laptop cá nhân. Dù phải đánh đổi một phần nhỏ độ chính xác

tuyệt đối so với các hệ thống chuyên dụng, MediaPipe mở ra khả năng ứng dụng rộng rãi mà không tạo rào cản về phần cứng cho người dùng.

2.1.2 OpenCV: Cơ chế nền tảng và tiền xử lý dữ liệu

Trong kiến trúc hệ thống AEYEPro, thư viện OpenCV (Open Source Computer Vision Library) đóng vai trò là lớp phần mềm trung gian (middleware) chịu trách nhiệm giao tiếp phần cứng và chuẩn hóa dữ liệu trước khi đưa vào mô hình AI. Không chỉ đơn thuần là công cụ đọc hình ảnh, OpenCV hoạt động như một hệ nền tính toán ma trận tốc độ cao, đảm bảo dữ liệu đầu vào cho MediaPipe đạt độ chính xác và hiệu năng thời gian thực.

Cơ chế hoạt động nền tảng

Về mặt bản chất kỹ thuật, OpenCV xử lý hình ảnh dưới dạng cấu trúc dữ liệu toán học nòng cốt là `cv::Mat`. Mỗi khung hình thu được từ camera được mã hóa thành một ma trận đa chiều (Multidimensional Dense Array). Với một ảnh màu kích thước $H \times W$, OpenCV quản lý nó như một Tensor 3 chiều có kích thước $H \times W \times 3$ (tương ứng với 3 kênh màu Blue-Green-Red). Mỗi phần tử (pixel) là một số nguyên 8-bit nằm trong khoảng [0, 255]:

$$\text{FrameMatrix} = \begin{bmatrix} (B, G, R)_{0,0} & \cdots & (B, G, R)_{0,W} \\ \vdots & \ddots & \vdots \\ (B, G, R)_{H,0} & \cdots & (B, G, R)_{H,W} \end{bmatrix}$$

Mặc dù hệ thống được triển khai trên Python, OpenCV hoạt động thông qua các lớp được đóng gói gọi xuống mã nguồn C++ đã được tối ưu hóa ở tầng thấp. Điều này cho phép ứng dụng tận dụng tốc độ tính toán của ngôn ngữ máy và khả năng quản lý bộ nhớ hiệu quả, đồng thời tương thích hoàn toàn với thư viện số học *NumPy*.

Quy trình tiền xử lý dữ liệu

Để đảm bảo mô hình Deep Learning của MediaPipe hoạt động chính xác, OpenCV thực hiện một quy trình chuẩn hóa dữ liệu đầu vào gồm các bước trọng yếu:

- Thu thập tín hiệu:** Sử dụng mô-đun `cv2.VideoCapture` để thiết lập luồng kết nối với webcam, quản lý bộ đệm để duy trì tốc độ khung hình ổn định và giảm thiểu độ trễ.
- Chuyển đổi không gian màu:** Đây là bước kỹ thuật bắt buộc. Theo mặc định, OpenCV giải mã tín hiệu dưới định dạng **BGR**, trong khi mạng nơ-ron của MediaPipe được huấn luyện trên định dạng **RGB**. Hệ thống thực hiện phép biến đổi tuyến tính trên từng pixel:

$$\text{Input}_{\text{MediaPipe}} = \text{cv2.cvtColor}(\text{Frame}_{\text{Original}}, \text{cv2.COLOR_BGR2RGB})$$

- Chuẩn hóa hình học:** Để tối ưu hóa trải nghiệm người dùng trong ứng dụng Desktop, OpenCV thực hiện phép lật ảnh qua trục dọc (Horizontal Flip) bằng lệnh `cv2.flip(1)`, tạo hiệu ứng “gương soi” tự nhiên. Dữ liệu sau khi xử lý sẽ trở thành đầu vào chuẩn để MediaPipe thực hiện trích xuất đặc trưng và thực hiện tính toán.

2.2 Cơ chế hoạt động

Sự kết hợp giữa MediaPipe và OpenCV tạo nên quy trình xử lý khép kín:

- Input:** OpenCV thu nhận khung hình, chuyển đổi sang RGB.
- Inference:** MediaPipe tính toán các landmarks.
- Geometry Analysis:** Sử dụng landmarks đã tính toán ở trên, kết hợp với các thông số Mediapipe tính toán được để lưu trữ và thực hiện các tác vụ tiếp theo.
- Decision:** Hệ thống so sánh kết quả với ngưỡng an toàn và hiển thị cảnh báo nếu cần thiết.

2.3 Các thông số đánh giá

Để giám sát hiệu quả sức khỏe thị giác và hệ cơ xương khớp của người dùng máy tính, hệ thống tập trung phân tích bốn chỉ số sinh trắc học cốt lõi. Các thông số này được lựa chọn dựa trên các tiêu chuẩn công thái học (như phương pháp RULA/REBA) và khả năng trích xuất ổn định từ dữ liệu webcam thông qua MediaPipe.

2.3.1 Tỷ số khung hình mắt (Eye Aspect Ratio - EAR)

Để đánh giá trạng thái buồn ngủ và hành vi chớp mắt, hệ thống sử dụng chỉ số *Eye Aspect Ratio* (EAR), được đề xuất bởi Soukupová và Čech (2016) [5]. Khác với các phương pháp xử lý ảnh truyền thống dựa trên vùng tối (blob detection), EAR sử dụng hình học các điểm landmarks, giúp thuật toán bất biến với tỉ lệ ảnh và hướng xoay của khuôn mặt.

EAR được định nghĩa là tỷ lệ giữa chiều cao và chiều rộng của mắt:

$$EAR = \frac{||p_2 - p_6|| + ||p_3 - p_5||}{2 \times ||p_1 - p_4||} \quad (2.1)$$

Trong đó:

- p_1, p_4 : Các điểm mốc tại khói mắt (đại diện cho chiều ngang).
- p_2, p_6 và p_3, p_5 : Các cặp điểm đối xứng qua trục ngang tại mí trên và mí dưới.

- **Lưu ý:** Đối với lưới điểm MediaPipe Face Mesh (468 điểm), các điểm $p_1 \dots p_6$ được ánh xạ tới các đỉnh (indices) cụ thể. Ví dụ với mắt trái: $p_1 = 33$ (khóe trong), $p_4 = 133$ (khóe ngoài), các cặp mí mắt tương ứng là (160, 144) và (158, 153).

Trạng thái nhắm mắt dẫn đến tử số tiến về 0, làm giá trị EAR giảm đột ngột. Khi buồn ngủ thì EAR có thể thấp trong một khoảng thời gian dài, có thể là 3 đến 5 giây, đó là dấu hiệu của sự mệt mỏi, người dùng cần được nghỉ ngơi hợp lý.

2.3.2 Độ lệch trục vai (Shoulder Alignment)

Sự cân bằng của đai vai là chỉ số quan trọng để phát hiện tư thế ngồi sai lệch (asymmetric sitting posture). Dựa trên mô hình MediaPipe Pose, hệ thống xác định vector nối hai điểm vai trái (S_L) và vai phải (S_R).

Do MediaPipe trả về tọa độ chuẩn hóa (normalized coordinates) trong khoảng [0, 1], việc tính toán góc trực tiếp sẽ gây sai số nếu tỷ lệ khung hình (aspect ratio) của camera không phải là 1:1. Công thức tính góc nghiêng vai $\theta_{shoulder}$ được hiệu chỉnh theo kích thước ảnh thực tế ($W \times H$) như sau:

$$\theta_{shoulder} = \arctan \left(\frac{(y_{S_L} - y_{S_R}) \times H}{(x_{S_L} - x_{S_R}) \times W} \right) \times \frac{180}{\pi} \quad (2.2)$$

Trong đó W và H lần lượt là chiều rộng và chiều cao của khung hình video đầu vào. Một tư thế ngồi chuẩn yêu cầu $|\theta_{shoulder}| \approx 0^\circ$. Hệ thống sẽ kích hoạt cảnh báo nếu góc nghiêng vượt quá ngưỡng cho phép (thường là $\pm 15^\circ$).

2.3.3 Góc gập cổ (Head Pitch)

Góc Pitch (trục x) phản ánh chuyển động gập (flexion) hoặc duỗi (extension) của đầu. Trong bối cảnh làm việc văn phòng, góc gập cổ quá lớn (Forward Head Posture) làm gia tăng áp lực đè nén lên đốt sống cổ. Theo nghiên cứu của Hansraj (2014) [6], khi đầu cúi 60 độ, áp lực lên cột sống cổ có thể tăng lên tới 27kg (60 lbs), gây ra hội chứng "Text Neck".

Hệ thống sử dụng giải thuật PnP (Perspective-n-Point) trên lưới 468 điểm của MediaPipe Face Mesh để ước lượng góc Euler của đầu.

- **Pitch dương (+):** Đầu cúi xuống. Cảnh báo khi góc $> 15^\circ$ để ngăn ngừa đau mỏi cổ vai gáy.
- **Pitch âm (-):** Đầu ngửa ra sau. Cảnh báo tư thế không tự nhiên.

2.3.4 Góc nghiêng đầu (Head Roll)

Góc Roll (trục z) đo lường độ nghiêng ngang của đầu về phía vai trái hoặc phải (Lateral bending). Việc duy trì tư thế nghiêng đầu tĩnh trong thời gian dài (ví dụ: kẹp điện thoại vào vai hoặc nhìn màn hình phụ không đúng tầm mắt) sẽ gây căng cơ ức đòn chũm và mất cân bằng cơ cổ.

Góc Roll được tính toán từ vector nối hai điểm đuôi mắt hoặc hai điểm tai trong không gian 3D. Hệ thống giám sát độ lệch này so với trục thẳng đứng chuẩn và đưa ra nhắc nhở nếu người dùng duy trì trạng thái nghiêng lệch kéo dài, đảm bảo tầm nhìn luôn song song với mặt đất.

2.3.5 Các thông số khác

Bên cạnh đó hệ thống còn sử dụng một vài thông số khác để đánh giá như **khoảng cách từ mắt đến màn hình** để biết được người dùng có ngồi đúng khoảng cách hay không, **số lần buồn ngủ** mà hệ thống đánh giá được để biết mức độ tập trung của người dùng.

Mặt khác, sau mỗi phiên làm việc của người dùng, hệ thống cũng sẽ tổng hợp lại các thông số đánh giá, lưu giá trị trung bình của chúng để phục vụ cho hệ thống RAG, hỗ trợ đưa ra lời khuyên phù hợp cho người dùng.

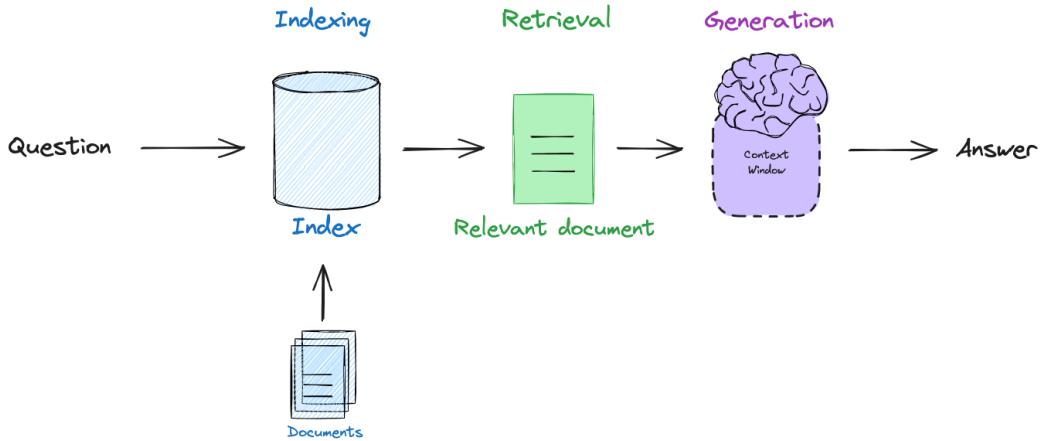
2.4 Adaptive RAG Chatbot

Trong bối cảnh y tế số (Digital Health) đang chuyển mình mạnh mẽ, các hệ thống hỗ trợ tư vấn sức khỏe đổi mới với "tam giác bất khả thi": đảm bảo tính trôi chảy của ngôn ngữ tự nhiên, duy trì độ chính xác tuyệt đối về chuyên môn, và khả năng tích hợp dữ liệu đa phương thức. Chương này thiết lập khung lý thuyết vững chắc cho nghiên cứu, tập trung phân tích sự chuyển dịch mang tính mô thức từ kiến trúc Retrieval-Augmented Generation (RAG) tĩnh sang kiến trúc Adaptive RAG động, đồng thời luận giải việc lựa chọn các công nghệ lõi như LangGraph, ChromaDB để hiện thực hóa hệ thống.

2.4.1 Sự Tiến Hóa Kiến Trúc: Từ RAG Truyền Thông đến Adaptive RAG

Sự phát triển của các hệ thống hỏi đáp y tế không chỉ là sự cải tiến về mô hình ngôn ngữ, mà là sự thay đổi về tư duy điều phối luồng thông tin (Information Flow Orchestration). **Chèn hình minh họa (placeholders)**

A. RAG Truyền Thông (Naive RAG): Tiếp cận tuyến tính và những điểm mù: Các mô hình ngôn ngữ lớn (LLM) sở hữu khả năng suy luận mạnh mẽ nhưng lại bị giới hạn bởi "tri



Hình 2.1: Minh họa kiến trúc RAG truyền thống

thức tham số" (parametric knowledge) tĩnh và nguy cơ "ảo giác" (hallucination). RAG truyền thống ra đời như một giải pháp bổ sung bộ nhớ ngoài (non-parametric memory).

Cơ chế hoạt động của Naive RAG tuân theo quy trình tuyến tính, đơn hướng (One-way Pipeline): Truy xuất (Retrieve) → Đọc (Read) → Sinh (Generate).

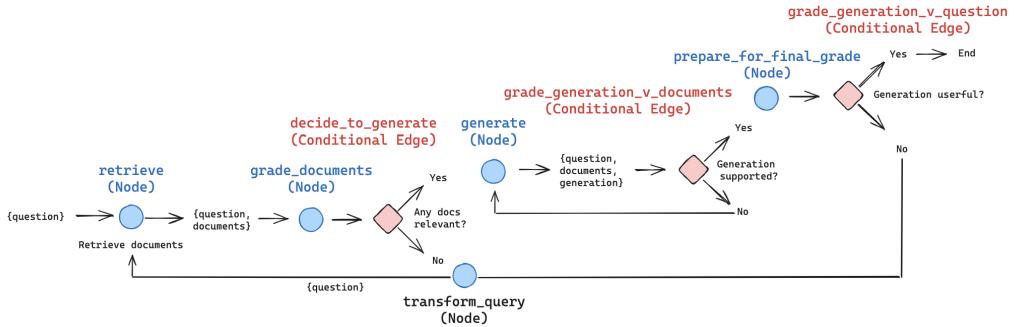
Truy xuất: Hệ thống vector hóa câu hỏi q và tìm kiếm top-K văn bản D có độ tương đồng cosine cao nhất.

Sinh: LLM sinh câu trả lời y dựa trên xác suất: $P(y | q, D)$.

Dánh giá hạn chế (Critical Analysis): Mặc dù giải quyết được vấn đề cập nhật kiến thức, Naive RAG bộc lộ những điểm yếu chí mạng khi áp dụng vào y tế:

- **Sự phụ thuộc tuyệt đối vào bước truy xuất:** Đây là điểm nghẽn duy nhất (Single Point of Failure). Nếu bước truy xuất trả về thông tin nhiễu hoặc không liên quan, LLM buộc phải "bịa" ra câu trả lời hoặc trả lời sai dựa trên ngữ cảnh sai (Garbage In, Garbage Out).
- **Thiếu khả năng nhận thức (Lack of Cognition):** Hệ thống hoạt động một cách mù quáng ("Fire and Forget"). Nó không biết liệu thông tin tìm được có thực sự trả lời được câu hỏi hay không, và không có cơ chế quay đầu để sửa sai.
- **Chiến lược tinh cho mọi vấn đề:** Dù là một câu hỏi đơn giản ("Sốt là gì?") hay phức tạp ("Phân tích tương tác thuốc giữa A và B trên bệnh nhân suy thận"), hệ thống đều dùng chung một quy trình xử lý, dẫn đến sự lãng phí tài nguyên hoặc thiếu hụt chiều sâu suy luận.

B. Adaptive RAG: Kiến trúc nhận thức và luồng xử lý đa vòng lặp: Adaptive RAG (RAG Thích ứng) không chỉ là một bản nâng cấp, mà là sự thay đổi về chất. Nó chuyển dịch hệ thống từ mô hình "Dây chuyền sản xuất" sang mô hình "Tác nhân nhận thức" (Cognitive Agent).



Hình 2.2: Minh họa kiến trúc Adaptive RAG

Thay vì chạy một mạch từ đầu đến cuối, Adaptive RAG mô phỏng quá trình tư duy phản biện của một bác sĩ: Dánh giá → Lập kế hoạch → Thực thi → Kiểm chứng.

Kiến trúc này được xây dựng dựa trên ba trụ cột lý thuyết vượt trội:

- Cơ chế Định tuyến động (Dynamic Query Routing): Phân loại trước khi Xử lý**
Hệ thống không nhầm lẫn tìm kiếm ngay lập tức. Một bộ định tuyến thông minh (Intelligent Router) sẽ phân tích ý định (intent) và độ phức tạp của câu hỏi để chọn chiến lược tối ưu:

- Câu hỏi lý thuyết: Điều hướng đến Vector Store (ChromaDB) để tra cứu y văn.
- Câu hỏi số liệu/thống kê: Điều hướng đến Pandas Agent để phân tích dữ liệu bảng.
- Câu hỏi xã giao/tổng quát: Sử dụng trực tiếp tri thức nội tại của LLM, bỏ qua bước truy xuất để giảm độ trễ.

→ Điều này đảm bảo "đúng công cụ cho đúng việc", tối ưu hóa tài nguyên tính toán.

- Cơ chế tự phản và sửa lỗi (Self-Reflection & Error Correction)**

Đây là sự khác biệt lớn nhất so với RAG truyền thống. Hệ thống tích hợp các "Giám khảo" (Grader/Evaluator Nodes) tại các chốt kiểm soát:

- Retrieval Grader: Dánh giá xem tài liệu tìm được có liên quan đến câu hỏi không. Nếu không, hệ thống tự động viết lại câu hỏi (Query Rewriting) để tìm kiếm lại theo hướng khác.
- Hallucination Grader: Kiểm tra chéo giữa câu trả lời sinh ra và tài liệu nguồn. Nếu phát hiện thông tin không có trong nguồn, hệ thống từ chối trả lời và thực hiện lại bước sinh văn bản.
- Answer Grader: Dánh giá xem câu trả lời có giải quyết triệt để vấn đề của người dùng chưa.

→ Cơ chế này tạo ra các vòng lặp (loops) trong đồ thị xử lý, đảm bảo độ tin cậy cực cao – yếu tố sống còn trong tư vấn sức khỏe.

3. Từ DAG sang Cyclic Graph (Đồ thị có chu trình)

Về mặt toán học và cấu trúc dữ liệu, Adaptive RAG chuyển từ Đồ thị không chu trình có hướng (DAG) sang Đồ thị có chu trình. Điều này cho phép hệ thống duy trì trạng thái (Statefulness), "nhớ" được các nỗ lực truy xuất trước đó thất bại ở đâu để điều chỉnh hướng đi tiếp theo, tạo ra một trải nghiệm hội thoại thông minh và kiên trì tìm kiếm chân lý.

Bảng 2.2: So sánh tóm tắt giữa RAG Truyền thống (Naive) và Adaptive RAG (Đề xuất)

Tiêu chí	RAG Truyền thống (Naive)	Adaptive RAG (Đề xuất)
Cấu trúc	Tuyến tính (Linear), một chiều.	Đồ thị (Graph), đa chiều, có vòng lặp (Cyclic), hỗ trợ trạng thái (Stateful).
Cơ chế xử lý lỗi	Không có cơ chế sửa lỗi tự động — sai ở đâu, hỏng ở đó.	Tích hợp các Grader (Retrieval/ Hallucination/ Answer) và các vòng lặp feedback để tự sửa lỗi (self-correction).
Tính linh hoạt	Cứng nhắc (Rigid), áp dụng quy trình "một kích cỡ cho tất cả".	Linh hoạt, điều chỉnh chiến lược xử lý (Routing) tùy theo độ phức tạp của câu hỏi.
Độ chính xác	Phụ thuộc hoàn toàn vào lần tìm kiếm đầu tiên; dễ sinh ảo giác nếu dữ liệu nhiễu.	Cao hơn nhờ cơ chế kiểm chứng chéo (Cross-check) và truy vấn lại (Retry) khi cần thiết.
Mô phỏng tư duy	Hoạt động máy móc, thụ động theo kịch bản có sẵn.	Mô phỏng tư duy phản biện (Agentic): Đánh giá → Lập kế hoạch → Thực thi → Kiểm chứng.

Kết luận, việc lựa chọn Adaptive RAG là bước đi chiến lược nhằm chuyển đổi vai trò của hệ thống từ một "công cụ tìm kiếm nâng cao" thành một "trợ lý y tế đáng tin cậy", có khả năng chịu trách nhiệm về thông tin mình đưa ra.

Hệ Sinh Thái Công Nghệ và Thư Viện Cốt Lõi

Dể hiện thực hóa kiến trúc phức tạp của Adaptive RAG, dự án sử dụng một tập hợp các công nghệ tiên tiến, được lựa chọn kỹ lưỡng dựa trên tính năng và hiệu năng.

A. LangGraph: Điều phối luồng xử lý đa tác vụ: Trong khi LangChain truyền thống mạnh về việc xây dựng các chuỗi (Chains) tuyến tính, LangGraph được lựa chọn làm xương sống cho hệ thống này nhờ khả năng hỗ trợ các kiến trúc có trạng thái (Stateful) và có chu trình (Cyclic).

Cơ chế hoạt động: LangGraph vận hành dựa trên mô hình tính toán Pregel (Bulk Synchronous Parallel). Quá trình xử lý được chia thành các bước (supersteps). Tại mỗi bước, các Agent (như Retriever, Generator) thực thi và cập nhật trạng thái vào một bộ nhớ chia sẻ gọi là GraphState.

Vai trò trong dự án: LangGraph quản lý toàn bộ workflow, từ việc định tuyến câu hỏi, lưu trữ ngữ cảnh hội thoại, đến việc điều khiển các vòng lặp retry khi gặp lỗi ảo giác.

B. ChromaDB: Cơ sở dữ liệu Vector hiệu năng cao: ChromaDB đóng vai trò là "hải mã" (hippocampus) của hệ thống, chịu trách nhiệm lưu trữ và truy xuất tri thức y khoa dài hạn.

Thuật toán HNSW: ChromaDB sử dụng thuật toán tìm kiếm láng giềng gần nhất xấp xỉ (ANN), cho phép truy xuất dữ liệu với độ phức tạp $O(\log N)$, đảm bảo tốc độ phản hồi thời gian thực ngay cả với lượng dữ liệu lớn.

Ứng dụng: Lưu trữ các vector embedding của hàng nghìn tài liệu y khoa, hướng dẫn lâm sàng và hồ sơ bệnh học.

C. Hugging Face Sentence Transformers: Biểu diễn Ngữ nghĩa: Một thành phần không thể thiếu trong quy trình RAG là mô hình Embedding. Dự án lựa chọn thư viện Hugging Face Sentence Transformers thay vì các API thương mại (như OpenAI) vì các lý do chiến lược:

Công dụng trong dự án: Thư viện này cung cấp các mô hình mã nguồn mở (như all-MiniLM-L6-v2) để chuyển đổi văn bản y khoa và câu hỏi người dùng thành các vector số học chất lượng cao.

Ưu điểm: Việc chạy mô hình cục bộ (Local Inference) thông qua Hugging Face đảm bảo an toàn dữ liệu tuyệt đối – một yêu cầu bắt buộc trong y tế (tuân thủ HIPAA). Dữ liệu bệnh nhân không bao giờ rời khỏi hạ tầng của hệ thống. Đồng thời, nó giúp loại bỏ chi phí tính toán theo token và giảm độ trễ mạng.

D. LangChain & Pandas Agent: Xử lý dữ liệu có cấu trúc: Để giải quyết bài toán dữ liệu bảng (file CSV/Excel), hệ thống sử dụng LangChain kết hợp với Pandas DataFrame Agent.

Cơ chế: Thay vì ép LLM phải "học thuộc" dữ liệu số, Agent này sử dụng khả năng suy luận của LLM để sinh ra mã nguồn Python (sử dụng thư viện Pandas).

Thực thi: Mã Python được chạy trong môi trường sandbox để tính toán chính xác các chỉ số (ví dụ: tính trung bình, tổng hợp, lọc dữ liệu) từ file log sức khỏe, đảm bảo độ chính xác toán học mà vector search không thể làm được.

E. LangSmith: Quan sát và đánh giá (Observability) LangSmith được tích hợp để giải quyết bài toán "hộp đen" của AI. Nó cung cấp khả năng theo dõi (Tracing) từng bước thực thi của Adaptive RAG.

Chức năng: Ghi lại input/output của từng node, thời gian thực thi, và quan trọng nhất là kết quả của các bước đánh giá (Grading). Điều này cho phép đội ngũ phát triển tinh chỉnh prompt và tối ưu hóa ngưỡng quyết định (decision thresholds) cho bộ định tuyến.

Chương 3

Triển khai & Phân tích

3.1 Triển khai Tổng thể Hệ thống

3.1.1 Kiến trúc Tổng thể Hệ thống

Để đáp ứng yêu cầu xử lý tác vụ thị giác máy tính cũng như mô hình RAG đã trình bày ở trên đồng thời duy trì khả năng phản hồi trên giao diện người dùng, hệ thống AEyePro được xây dựng dựa trên mô hình Client–Server với cơ chế giao tiếp song song. Kiến trúc này tách biệt hoàn toàn tầng giao diện và tầng xử lý logic, đảm bảo tính ổn định và khả năng mở rộng của hệ thống.

Mô hình Client–Server

Frontend (Client). Frontend đóng vai trò là giao diện tương tác người dùng, chịu trách nhiệm hiển thị dữ liệu thời gian thực và nhận yêu cầu từ người dùng. Client được xây dựng trên nền tảng HTML, CSS và JavaScript nhằm tối ưu hiệu suất mà không phụ thuộc vào các framework nặng.

Backend (Server). Backend là trung tâm xử lý tính toán của hệ thống. Backend được phát triển bằng Python và framework Flask với khả năng triển khai nhanh các điểm cuối API (API Endpoints).

Cơ chế giao tiếp kép

Một thách thức lớn của các ứng dụng giám sát thời gian thực là độ trễ. Để giải quyết vấn đề này, hệ thống triển khai song song hai giao thức giao tiếp.

Giao thức RESTful API (HTTP). Được sử dụng cho các tác vụ quản trị và thiết lập trạng thái có tần suất thấp nhưng yêu cầu độ tin cậy cao, tiêu biểu như:

- Khởi động hoặc dừng camera: /api/camera/start;

- Cập nhật cài đặt ngưỡng cảnh báo: `/api/settings`.
- Gửi tin nhắn cho chatbot: `/api/chatbot/message`

Việc sử dụng REST giúp đơn giản hóa quản lý tài nguyên và luồng điều khiển hệ thống.

Giao thức Socket.IO. Đây là kênh giao tiếp chủ đạo cho luồng dữ liệu thời gian thực. Khác với HTTP truyền thông vốn tạo độ trễ cao do quá trình đóng/mở kết nối liên tục, WebSocket duy trì một kết nối hai chiều (bi-directional) ổn định giữa Client và Server. Điều này cho phép:

- Truyền tải luồng video (video streaming) 15–30 FPS,
- Cập nhật các chỉ số liên tục,

mà không gây quá tải băng thông.

Kiến trúc Đa Luồng

Nhằm đảm bảo các tác vụ xử lý ảnh không làm gián đoạn khả năng phản hồi, hệ thống sử dụng đa luồng kết hợp với thiết kế Singleton (Design Pattern - OOP).

Tách biệt luồng xử lý (Thread Separation). Bộ máy xử lý thị giác được vận hành trên các luồng riêng biệt, độc lập với luồng chính của Flask Server. Nhờ vậy, Server vẫn có thể tiếp nhận request từ Client ngay cả khi CPU đang thực hiện các tác vụ xử lý thị giác.

Quy trình hoạt động Tổng Thể

Dữ liệu hình ảnh từ Camera được xử lý trên luồng riêng, kết quả được truyền qua kênh Socket.IO để cập nhật dữ liệu liên tục, và cuối cùng được Frontend hiển thị lên giao diện người dùng. Với những tác vụ quản trị hay thiết lập trạng thái (bật/tắt camera, điều chỉnh cài đặt) sẽ được thực hiện thông qua RESTful API giúp tối ưu tài nguyên và luồng cho hệ thống.

3.1.2 Thiết kế Backend và API

Tầng backend đóng vai trò như “bộ não” điều phối, kết nối các module xử lý thị giác máy tính và module RAG với giao diện người dùng. Thiết kế của tầng này ưu tiên sự đơn giản, khả năng mở rộng và hiệu năng thời gian thực.

Lựa chọn công nghệ

Hệ thống được phát triển hoàn toàn trên ngôn ngữ Python với Flask Framework làm nền tảng. Quyết định này dựa trên các yếu tố sau:

Tích hợp tốt với thư viện AI. Việc sử dụng Python cho phép tích hợp trực tiếp các thư viện như:

- OpenCV (xử lý ảnh),
- MediaPipe/PyTorch (mô hình học sâu),
- LangChain (xử lý ngôn ngữ tự nhiên/RAG),

vào cùng một tiến trình mà không cần thông qua các cầu nối phức tạp.

Kiến trúc Micro-framework của Flask. Khác với các framework nguyên khôi (như Django), Flask cung cấp sự linh hoạt tối đa, cho phép nhà phát triển lựa chọn các thư viện bổ trợ (như Flask-SocketIO cho WebSocket) mà không bị ràng buộc bởi cấu trúc dư thừa. Điều này đặc biệt quan trọng nhằm tối ưu thời gian khởi động và tài nguyên bộ nhớ cho các ứng dụng xử lý ảnh thời gian thực.

Cơ chế hoạt động của Flask, RESTfulAPI và Socket.IO

Hệ thống được xây dựng dựa trên sự phối hợp của ba công nghệ chính: **Flask** đóng vai trò là lõi điều phối ứng dụng, **RESTful API** đảm nhiệm lớp điều khiển, và **Socket.IO** vận hành lớp dữ liệu thời gian thực.

1. Phương thức giao tiếp HTTP

HTTP (Hypertext Transfer Protocol) là giao thức truyền tải siêu văn bản được sử dụng để trao đổi dữ liệu giữa Client và Server. HTTP định nghĩa cách thức mà các yêu cầu (requests) từ client được gửi đến server và cách server trả về phản hồi (responses). Các phương thức HTTP chính sẽ dùng:

- GET – truy xuất dữ liệu.
- POST – gửi dữ liệu hoặc thay đổi trạng thái..

2. Cơ chế vận hành của Flask Framework

Flask không chỉ đơn thuần là một thư viện, mà là một framework hoạt động dựa trên chuẩn WSGI (Web Server Gateway Interface) cùng cơ chế quản lý ngôn ngữ cảnh nhằm đảm bảo an toàn luồng [9].

- **Chuẩn giao tiếp WSGI:** Khi máy chủ nhận được một HTTP Request, nó chuyển đổi thông tin thành một dạng dữ liệu mà chương trình Python có thể đọc được (`environ`) và gọi đến Flask. Sau đó, Flask xác định URL mà người dùng truy cập và tìm xem hàm Python nào sẽ xử lý yêu cầu đó.

- **Vòng đời yêu cầu và ngữ cảnh:** Flask quản lý hai loại ngữ cảnh: **Ngữ cảnh ứng dụng** (Cấu hình, các biến toàn cục,...) và **Ngữ cảnh yêu cầu** (chứa thông tin của yêu cầu HTTP hiện tại). Khi một yêu cầu đến, Flask đẩy hai ngữ cảnh này vào **stack**. Điều này cho phép truy cập biến toàn cục và **request** từ mọi nơi trong mã nguồn mà không cần truyền tham số qua từng hàm.
- **Cơ chế Thread-Safety:** Để xử lý đa luồng (như luồng xử lý ảnh trong dự án), Flask sử dụng cơ chế **Thread Local Storage** thông qua cấu trúc **LocalStack**. Mỗi luồng có một định danh riêng (Thread ID), được dùng làm khóa để lưu trữ và truy xuất dữ liệu ngữ cảnh. Điều này đảm bảo rằng biến **request** ở luồng xử lý ảnh không xung đột với **request** của luồng API.

3. Kiến trúc RESTful API

REST được lựa chọn cho các tác vụ quản trị hệ thống nhờ tính ổn định, dễ mở rộng và khả năng phân tách trách nhiệm rõ ràng [10].

- **Nguyên lý Phi trạng thái:** Server không lưu trạng thái phiên (session) của client giữa các yêu cầu. Mỗi yêu cầu HTTP (ví dụ: POST /api/settings) chứa đủ thông tin để xử lý độc lập. Điều này tối ưu hóa khả năng mở rộng và giảm tải cho bộ nhớ server.
- **Giao diện thống nhất:** API sử dụng các HTTP Verbs chuẩn: GET để truy xuất cấu hình và POST để thay đổi trạng thái (như bật/tắt camera).
- **Hạn chế trong truyền tải thời gian thực:** REST không phù hợp để truyền video liên tục do chi phí lớn. Mỗi HTTP Request mang khoảng ~ 800 bytes Header và yêu cầu thiết lập lại kết nối, gây lãng phí băng thông và tăng độ trễ khi tần suất gửi cao.

4. Cơ chế Socket.IO

Để truyền tải video và chỉ số sức khỏe thời gian thực, hệ thống sử dụng Socket.IO với kiến trúc gồm hai tầng: **Engine.IO** (giao tiếp) và **Socket.IO** (tầng ứng dụng) [11].

- **Cơ chế Nâng cấp Giao thức:** Socket.IO áp dụng chiến lược “Độ tin cậy trước – Nâng cấp sau”. Ban đầu, client và server giao tiếp giống như gửi yêu cầu HTTP liên tục, cách này đảm bảo luôn có kết nối hoạt động, ngay cả khi mạng có tường lửa hoặc trình duyệt không hỗ trợ WebSocket. Sau khi xác nhận client có thể dùng WebSocket, Socket.IO nâng cấp kết nối lên WebSocket, cho phép truyền dữ liệu nhanh hơn, ít tốn băng thông hơn.
- **Tối ưu hóa Băng thông:** Khi đã nâng cấp lên WebSocket, mỗi gói tin dữ liệu chỉ có chi phí 2–6 bytes, cho phép truyền luồng video 15–30 FPS và dữ liệu chỉ số sức khỏe liên tục mà không gây quá tải CPU hoặc băng thông nội bộ.

- **Cấu trúc Gói tin:** Dữ liệu ảnh sử dụng gói tin loại BINARY_EVENT, trong khi dữ liệu JSON sử dụng EVENT. Điều này giúp tối ưu tốc độ và đảm bảo tính toàn vẹn của dữ liệu.

Kiến trúc API

Hệ thống API được thiết kế dựa trên hai **kênh giao tiếp**, mỗi lớp phục vụ một mục đích riêng biệt.

1. Lớp điều khiển (RESTful API). Giao thức HTTP (REST) được sử dụng cho các tác vụ quản trị, cấu hình hệ thống và quản lý trạng thái, nơi độ tin cậy được ưu tiên hơn tốc độ.

Các điểm cuối chính bao gồm:

- POST /api/camera/start và /api/camera/stop: Kiểm soát bật/tắt camera đồng thời API này đảm bảo tài nguyên camera được giải phóng đúng cách khi không sử dụng.
- GET /api/settings và POST /api/settings: Quản lý file cấu hình.
- GET /api/camera/status: Lấy trạng thái hiện tại (đang chạy hay tắt, FPS, ...).
- POST /api/settings/face-mesh: Bật tắt các điểm landmarks trên camera.
- POST /api/chatbot/message: gửi tin nhắn cho chatbot

Cơ chế Hot-reload. Một đặc điểm nổi bật trong Lớp điều khiển là khả năng “tải nóng”. Khi người dùng cập nhật tham số (ví dụ: thay đổi ngưỡng cảnh báo gần màn hình từ 30cm thành 25cm) hệ thống cập nhật trực tiếp biến trạng thái trong bộ nhớ. Điều này cho phép tinh chỉnh thuật toán trong thời gian thực mà không cần khởi động lại hệ thống.

2. Lớp dữ liệu (WebSocket Events). Giao thức WebSocket (qua Flask-SocketIO) được sử dụng để truyền tải luồng dữ liệu liên tục với độ trễ thấp. Luồng dữ liệu được chia thành hai sự kiện chính:

Sự kiện camera_frame (Truyền hình ảnh).

- **Dữ liệu gửi đi:** Chuỗi ảnh JPEG mã hoá Base64.
- **Tần suất:** ~ 15 Hz (15 lần mỗi giây).

Tần suất này là sự cân bằng tối ưu: đủ mượt để mắt người cảm nhận chuyển động liên tục, đồng thời giảm khoảng 50% băng thông so với chuẩn 30 FPS để dành tài nguyên cho xử lý AI.

Sự kiện health_metrics (Truyền chỉ số).

- **Dữ liệu gửi đi:** Đối tượng JSON chứa EAR, góc nghiêng đầu (Pitch, Roll), trạng thái cảnh báo.
- **Tần suất:** ~ 2 Hz (2 lần mỗi giây).

Các chỉ số sinh trắc học biến đổi chậm hơn so với dữ liệu hình ảnh, vì vậy giảm tần suất gửi giúp giảm chi phí xử lý, đảm bảo giao diện hoạt động mượt mà.

Tách biệt giữa hai giao thức. Việc phân lớp rõ ràng giúp hệ thống đạt độ ổn định cao:

- Lớp dữ liệu chịu trách nhiệm truyền tải dữ liệu nặng và liên tục,
- Lớp điều khiển đảm bảo khả năng điều khiển không bị nghẽn.

Nhờ đó, người dùng luôn có thể can thiệp vào hoạt động của ứng dụng trong mọi tình huống, ngay cả khi luồng dữ liệu video đang ở tải trọng cao.

3.1.3 Thiết kế Giao diện Người dùng

Giao diện người dùng của **AeyePro** được xây dựng dựa trên các ngôn ngữ HTML, CSS, JavaScript, kết nối với Python Backend. Mỗi thành phần đảm nhiệm một vai trò chuyên biệt trong việc xử lý giao diện và tương tác người dùng.

HTML: Cấu trúc và Định hình Layout

HTML đóng vai trò tạo dựng bộ khung xương và cấu trúc ngữ nghĩa cho toàn bộ ứng dụng. Tệp `index.html` được thiết kế để phân chia bố cục thành các khu vực chức năng rõ rệt:

- **Vùng chứa dữ liệu:** Gồm các thẻ `<div>` và `` có `id` cố định để chứa các chỉ số sức khỏe (Blink Rate, EAR, Distance, Shoulder/Head Angles, Drowsy Events). Các thẻ này đóng vai trò làm người giữ chỗ để JavaScript cập nhật giá trị theo thời gian thực.
- **Vùng hiển thị Camera:** Sử dụng thẻ `<canvas>` để vẽ lại các khung hình video nhận được từ server. Việc dùng `canvas` thay cho `` giúp tối ưu hiệu năng render và giảm giật lag.
- **Vùng điều khiển và Cài đặt:** Chứa phương thức nhập và tăng giảm cho nhóm cài đặt (Frame Rate, Camera Index, EAR Thresholds, Min/Max Distance) cùng các nút chức năng như bật/tắt camera và bật/tắt landmarks.
- **Vùng Chatbot:** Bao gồm khung hiển thị lịch sử hội thoại và ô nhập liệu tin nhắn.

CSS: Trực quan hóa và định hình chủ đề

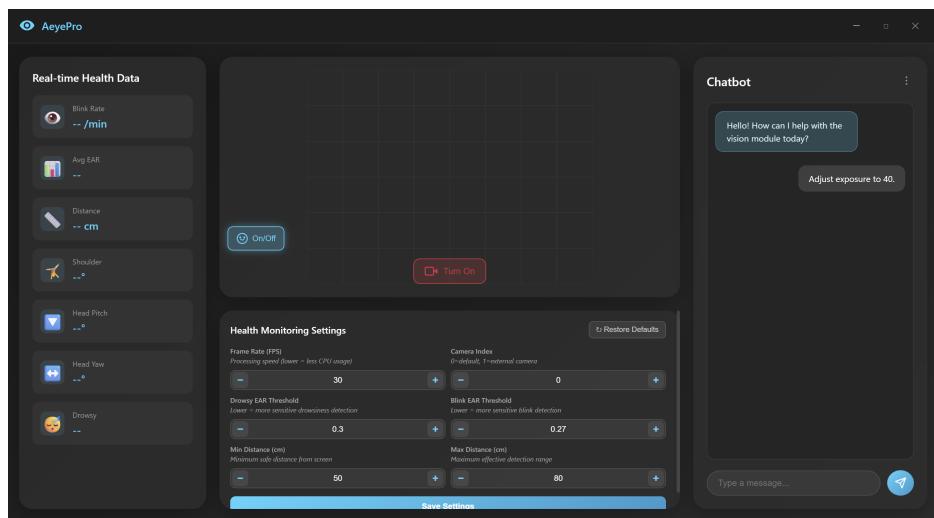
CSS chịu trách nhiệm chuyển đổi cấu trúc HTML thành một giao diện trực quan, hiện đại. Một số thành phần chính:

- **Theming (Chủ đề):** Sử dụng nền tối (#1a1a1a) nhằm giảm mỏi mắt, kết hợp màu neon (xanh/tím) và hiệu ứng *box-shadow* để làm nổi bật các thông số quan trọng và các thông báo cảnh báo.
- **Bố cục linh hoạt (Flexbox / Grid):** Áp dụng CSS Flexbox để duy trì bố cục 3 cột ổn định trên các kích thước cửa sổ khác nhau, tránh tình trạng vỡ giao diện.
- **Trạng thái tương tác:** Định nghĩa hiệu ứng chuyển động khi người dùng tương tác với nút bấm và khi các cảnh báo xuất hiện/biến mất.

JavaScript: Điều khiển Logic và Xử lý Thời gian thực

JavaScript là thành phần quan trọng của phía Client, chịu trách nhiệm và giao tiếp với Backend thông qua WebSocket và REST API. Các logic bao gồm:

- **Quản lý giao tiếp WebSocket (Real-time Communication)**
- **Điều khiển và Gọi API (RESTful Interaction)**
- **Hệ thống Thông báo và Logic Frontend:** quản lý các cảnh báo và thông báo tới người dùng



Hình 3.1: Giao diện người dùng

3.2 Triển khai Hệ thống Thị giác thời gian thực

Đây là luồng xử lý quan trọng, hoạt động liên tục với tần suất từ 15–30 Hz nhằm đảm bảo độ trễ thấp cho trải nghiệm người dùng. Quy trình khép kín bao gồm bốn giai đoạn:

3.2.1 Kích hoạt

- Quy trình bắt đầu khi người dùng nhấn nút Start Camera trên giao diện.
- Một tín hiệu HTTP POST được gửi tới API /api/camera/start.
- VisionManager khởi tạo luồng đọc camera và nạp các mô hình AI vào bộ nhớ.

3.2.2 Thu nhận và xử lý dữ liệu

- Camera thu nhận khung hình thô (*Raw Frame*).
- Khung hình được tiền xử lý bằng MediaPipe và đưa vào mô hình học sâu.
- Hệ thống trích xuất 478 điểm đặc trưng khuôn mặt (*Face Mesh*) và 33 điểm cơ thể (*Pose*).

3.2.3 Tính toán và phân luồng

- Dữ liệu thô được chuyển thành các chỉ số phân tích (EAR, Head Pose).
- Luồng dữ liệu được tách thành hai nhánh song song:
 - **Luồng hiển thị:** Các điểm đặc trưng được vẽ chồng lên video, mã hoá JPEG và truyền qua `camera_frame`.
 - **Luồng số liệu:** Các chỉ số định lượng được đóng gói thành JSON và phát qua `health_metrics`.

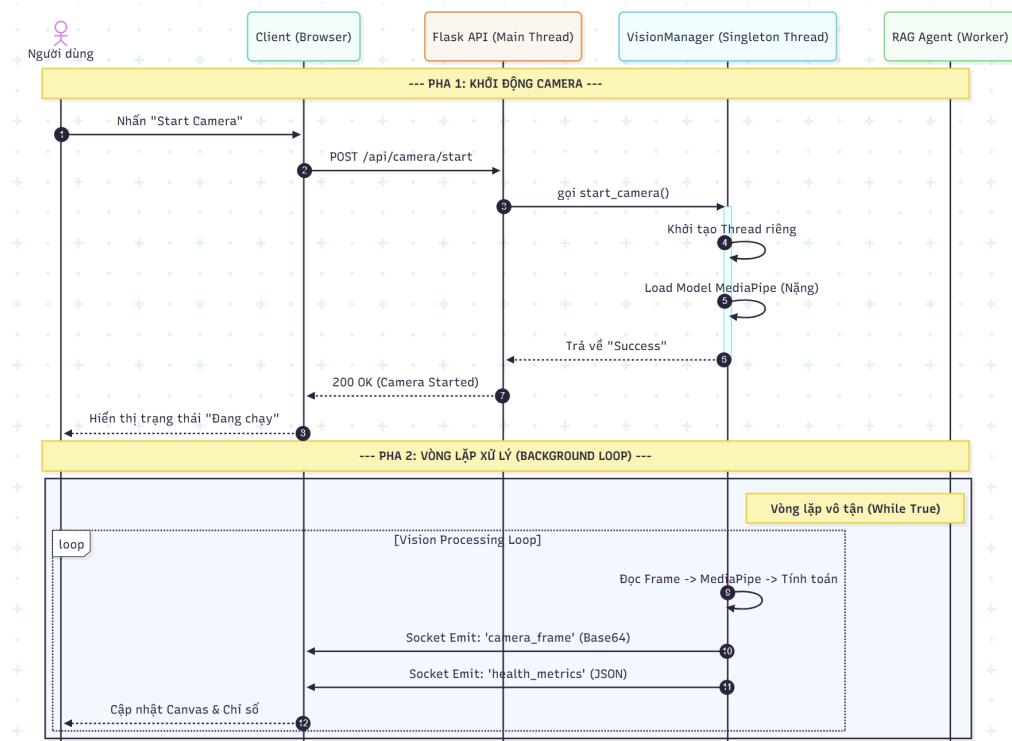
3.2.4 Lưu tổng quan dữ liệu của phiên

Kết quả phân tích mới nhất được lưu trong biến toàn cục của `VisionManager`. Khi đang trong phiên, dữ liệu sẽ được ghi liên tục. Và sau khi kết thúc phiên, dữ liệu sẽ được lưu vào bộ nhớ, dữ liệu này phục vụ trực tiếp cho luồng RAG.

Lớp `VisionManager` (lớp xử lý logic nghiệp vụ) được hiện thực theo mẫu Singleton, đảm bảo chỉ có một tồn tại duy nhất quản lý truy cập camera và trạng thái của các mô hình AI. Cơ chế này giúp:

- Tránh xung đột tài nguyên phần cứng (camera, GPU),

- Nhất quán dữ liệu toàn cục trong quá trình vận hành.



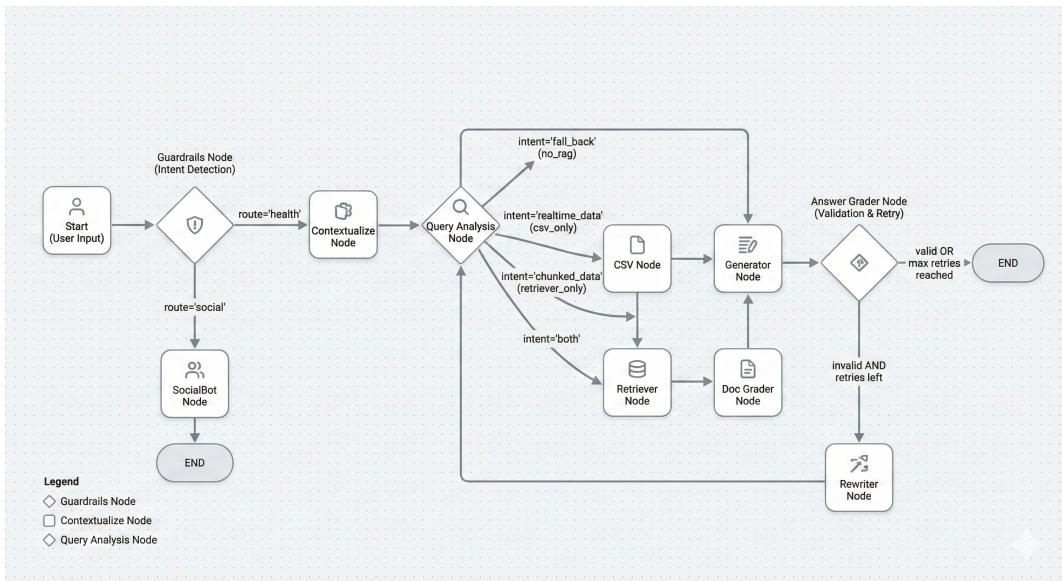
Hình 3.2: Tổng quan về luồng Thị giác

3.3 Triển Khai Hệ Thông Chatbot

Phần này trình bày chi tiết kiến trúc kỹ thuật và cơ chế vận hành của Health Care Adaptive RAG Chatbot. Thay vì sử dụng các chuỗi xử lý tuyến tính (*linear chains*) truyền thống, hệ thống được hiện thực hóa dưới dạng một **Đồ thị Trạng thái (State Graph)** có hướng và có chu trình. Trong mô hình này, mỗi đỉnh (*Node*) đại diện cho một tác nhân nhận thức (*cognitive agent*) hoặc một hàm xử lý logic, và các cạnh (*Edges*) đại diện cho luồng chuyển giao dữ liệu được điều kiện hóa bởi các quyết định thời gian thực.

3.3.1 Kiến Trúc Tổng Thể và Cơ Chế Điều Phối

Hệ thống không vận hành theo cơ chế “Input-Output” đơn giản mà là một quy trình **điều phối đa bước (Multi-step Orchestration)**. Kiến trúc này cho phép hệ thống duy trì “bộ nhớ làm việc” (*working memory*) thông qua một đối tượng trạng thái chia sẻ (**GraphState**), lưu trữ lịch sử hội thoại, các tài liệu truy xuất được, và các cờ trạng thái (*flags*) để kiểm soát luồng đi.



Hình 3.3: Tổng quan về luồng Thi giác

Sơ đồ workflow bao gồm các phân hệ chức năng:

- **Lớp Nhận thức & Điều hướng:** Guardrails, SocialBot, Contextualize, Query Analysis.
- **Lớp Thực thi & Truy xuất:** Retriever (Văn bản), CSV Agent (Số liệu).
- **Lớp Tổng hợp & Kiểm định:** Doc Grader, Generator, Answer Grader, Rewriter.

3.3.2 Chi Tiết Các Node Xử Lý Logic

Dưới đây là phân tích sâu về chức năng và thuật toán của từng Node trong đồ thị:

1. Guardrails Node (Lớp Bảo vệ & Phân loại Sơ cấp) Đây là cổng tiếp nhận đầu tiên (*Entry Point*), đóng vai trò như bộ lọc an toàn và định hướng cấp cao.

Chức năng: Sử dụng kỹ thuật *Zero-shot Classification* hoặc *Keyword Matching* để phân tích ý định sơ khởi của người dùng với độ trễ thấp nhất.

Logic Điều hướng:

- Nếu input mang tính xã giao (greeting, identity questions), hệ thống định tuyến sang nhánh `route='social'`.
- Nếu input chứa từ khóa y tế, triệu chứng, hoặc yêu cầu tra cứu, hệ thống định tuyến sang nhánh `route='health'`.

2. SocialBot Node (Tác nhân Hội thoại Xã hội) Mục đích: Tách biệt luồng hội thoại thường nhật khỏi quy trình RAG vốn kém tài nguyên.

Hoạt động: Sử dụng một prompt template đơn giản để phản hồi thân thiện, duy trì sự tự nhiên của cuộc trò chuyện mà không cần kích hoạt các kết nối cơ sở dữ liệu.

3. Contextualize Node (Tái cấu trúc Ngữ cảnh Hội thoại) Trong nhánh xử lý y tế (*health*), câu hỏi của người dùng thường ở dạng ẩn dụ hoặc phụ thuộc vào câu trước (ví dụ: “Nó có lây không?” – “Nó” ở đây là bệnh đã nhắc đến trước đó).

Chức năng: Node này sử dụng **Lịch sử Trò chuyện (Chat History)** làm đầu vào cho LLM để thực hiện kỹ thuật *Coreference Resolution* (giải quyết đồng tham chiếu).

Đầu ra: Một “Câu hỏi độc lập” (*Standalone Question*) mang đầy đủ ngữ nghĩa, thay thế cho câu hỏi gốc bị thiếu ngữ cảnh. Đây là bước tiền xử lý bắt buộc để đảm bảo độ chính xác cho các bước tìm kiếm sau này.

4. Query Analysis Node (Bộ Định tuyến Thông minh - The Router) Đây là trung tâm điều phối (*Dispatcher*) quyết định chiến lược giải quyết vấn đề.

Công nghệ: Sử dụng khả năng *Function Calling* hoặc *Structured Output* của LLM để phân loại câu hỏi vào các cấu trúc dữ liệu định trước.

Cơ chế Định tuyến (Routing Logic):

- **intent='realtime_data'**: Phát hiện nhu cầu tính toán, thống kê, so sánh số liệu (Input: log nhịp tim, bảng huyết áp). → Chuyển đến CSV Node.
- **intent='chunked_data'**: Phát hiện nhu cầu tra cứu kiến thức, định nghĩa, phác đồ (Input: y văn, sách giáo khoa). → Chuyển đến Retriever Node.
- **intent='both'**: Câu hỏi phức tạp yêu cầu tổng hợp đa nguồn (ví dụ: “Huyết áp của tôi hôm nay có nằm trong ngưỡng an toàn theo hướng dẫn của Bộ Y tế không?”). → Kích hoạt song song (*Parallel Execution*) cả hai node trên.
- **intent='fall_back'**: Câu hỏi tối nghĩa hoặc ngoài phạm vi. → Kết thúc và yêu cầu làm rõ.

5. Các Node Xử Lý Dữ Liệu (Specialized Executors) Hệ thống xử lý sự kiện không đồng nhất của dữ liệu y tế bằng hai luồng riêng biệt:

Retriever Node (Dữ liệu Phi cấu trúc):

- Thực hiện *Dense Vector Retrieval* bằng cách so khớp cosine similarity giữa vector câu hỏi và vector trong ChromaDB.
- Kết quả trả về là tập hợp các đoạn văn bản ($D_{retrieved}$) tiềm năng.

CSV Node (Dữ liệu Có cấu trúc):

- Kích hoạt một *Pandas DataFrame Agent*. Node này hoạt động như một trình thông dịch mã (*Code Interpreter*).
- LLM sinh ra mã Python để thực thi các thao tác toán học (mean, max, groupby, filter) trực tiếp trên file CSV. Điều này khắc phục điểm yếu tính toán kém của các mô hình ngôn ngữ thuần túy.

6. Doc Grader Node (Bộ lọc Nhiều Thông tin) Không phải mọi tài liệu truy xuất được đều hữu ích. Việc đưa thông tin rác vào Generator sẽ gây ra ảo giác.

Chức năng: Một LLM đóng vai trò giám khảo (*Judge*) chấm điểm nhị phân (Yes/No) cho từng đoạn văn bản dựa trên độ liên quan (*Relevance*) với câu hỏi.

Tác động: Chỉ những tài liệu đạt điểm “Yes” mới được chuyển tiếp. Node này hoạt động như một cơ chế “Reranking” (xếp hạng lại) đơn giản hóa.

7. Generator Node (Tổng hợp và Sinh Lời giải) **Đầu vào:** Câu hỏi đã chuẩn hóa + Dữ liệu số (từ CSV Node) + Dữ liệu văn bản sạch (từ Doc Grader).

Xử lý: LLM thực hiện suy luận (*Reasoning*) để tổng hợp các mảnh ghép thông tin rời rạc thành một câu trả lời mạch lạc, có cấu trúc, đóng vai trò như một chuyên gia y tế tư vấn cho người dùng.

8. Answer Grader Node (Kiểm định và Vòng lặp Tự sửa lỗi) Đây là chốt chặn cuối cùng (*Quality Assurance*) tạo nên sự “Thích ứng” (*Adaptive*). Node này thực hiện quy trình kiểm tra 2 bước:

- **Hallucination Check (Kiểm tra độ trung thực):** So sánh câu trả lời với nguồn tài liệu. Nếu câu trả lời chứa thông tin không có trong tài liệu → Dán dấu là ảo giác.
- **Relevance Check (Kiểm tra độ phù hợp):** So sánh câu trả lời với câu hỏi gốc. Nếu không trả lời đúng trọng tâm → Dán dấu là lạc đề.

Logic Quyết định:

- Nếu đạt cả 2 tiêu chí → Kết thúc (END).
- Nếu vi phạm và còn lượt thử lại (Max Retries > 0) → Chuyển sang Rewriter Node.

9. Rewriter Node (Tối ưu hóa và Tái truy vấn) **Chức năng:** Phân tích lý do thất bại từ Answer Grader để biến đổi câu hỏi gốc. Node này có thể thực hiện *Query Expansion* (mở rộng truy vấn) hoặc *Query Decomposition* (chia nhỏ câu hỏi) để tìm kiếm thông tin theo góc độ mới, hiệu quả hơn.

3.3.3 Tổng Kết Các Luồng Xử Lý Chính (Key Workflows)

Dựa trên kiến trúc đồ thị nêu trên, hoạt động của hệ thống được khái quát hóa thành 3 luồng xử lý (*pathways*) chính, phục vụ các mục đích khác nhau của người dùng:

Luồng 1: Giao tiếp Xã hội (Fast Path - Low Latency) **Mục đích:** Xử lý các câu chào hỏi, tán gẫu, giúp chatbot thân thiện và phản hồi tức thì.

Quy trình:

- Start → **Guardrails**: Phân loại ý định là “social”.
- Guardrails → **SocialBot**: Sinh câu trả lời giao tiếp xã hội.
- SocialBot → **End**: Trả kết quả ngay lập tức, bỏ qua toàn bộ quy trình RAG.

Luồng 2: Phân tích Số liệu Sức khỏe (Structured Data Path) **Mục đích:** Trả lời các câu hỏi định lượng, thống kê từ hồ sơ sức khỏe cá nhân (ví dụ: “Nhịp tim trung bình tuần qua là bao nhiêu?”).

Quy trình:

- Start → **Guardrails**: Phân loại ý định là “health”.
- **Contextualize**: Làm rõ câu hỏi dựa trên lịch sử chat.
- **Query Analysis**: Router xác định cần dữ liệu số liệu (`realtime_data`).
- **CSV Node**: Pandas Agent viết và chạy code Python để tính toán trên file CSV.
- **Generator**: Nhận kết quả con số, chuyển thành lời văn tự vấn.
- **Answer Grader**: Kiểm tra độ chính xác của câu trả lời.
- **End**: Trả kết quả phân tích.

Luồng 3: Tư vấn Kiến thức Y khoa Chuyên sâu (Unstructured Data Path) **Mục đích:** Trả lời các câu hỏi về bệnh lý, thuốc, triệu chứng dựa trên y văn (ví dụ: “Triệu chứng của sốt xuất huyết là gì?”). Đây là luồng phức tạp nhất với khả năng tự sửa lỗi.

Quy trình:

- Start → **Guardrails**: Phân loại ý định là “health”.
- **Contextualize**: Làm rõ câu hỏi.
- **Query Analysis**: Router xác định cần kiến thức y văn (`chunked_data`).

- **Retriever Node:** Truy xuất các đoạn văn bản (chunks) từ ChromaDB.
- **Doc Grader:** Dánh giá và lọc bỏ các đoạn văn bản không liên quan.
- **Generator:** Tổng hợp thông tin từ các tài liệu đã lọc để trả lời.
- **Answer Grader:** Kiểm tra ảo giác và độ liên quan.
 - Nếu Đạt: → **End**.
 - Nếu Không Đạt: → **Rewriter Node** (Viết lại câu hỏi) → Quay lại bước Query Analysis (Vòng lặp sửa lỗi).

Chương 4

Kết quả

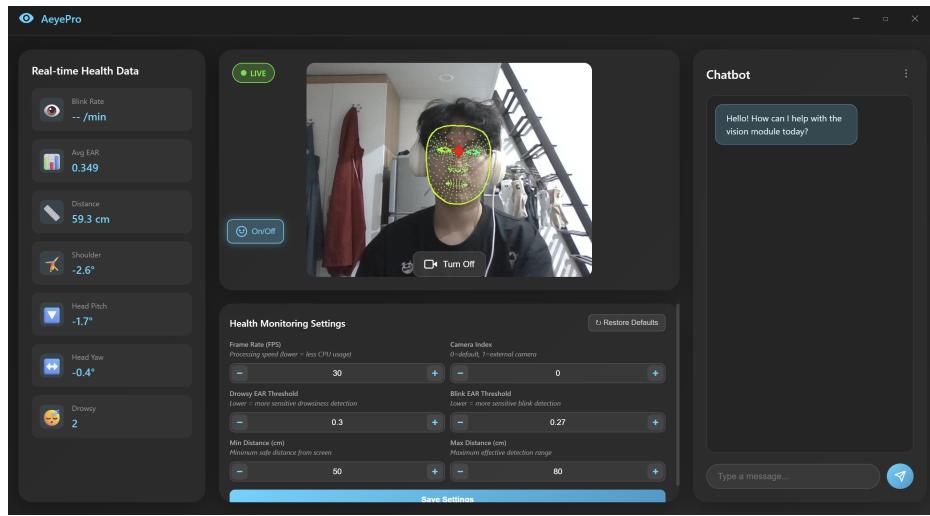
4.1 Kết quả thực nghiệm hệ thống thị giác máy tính

Hệ thống thị giác máy tính của AEye Pro đã được triển khai và thử nghiệm thực tế trên máy tính cá nhân sử dụng Webcam tích hợp. Kết quả thực nghiệm cho thấy module Thị giác máy tính hoạt động ổn định, trích xuất chính xác các đặc trưng khuôn mặt và đưa ra các cảnh báo tức thời với độ trễ thấp. Dưới đây là chi tiết kết quả thử nghiệm trên các kịch bản giám sát sức khỏe chính.

4.1.1 Khả năng nhận diện và theo dõi thời gian thực

Hệ thống tích hợp thành công mô hình **MediaPipe Face Mesh**, cho phép nhận diện và vẽ lối 468 điểm (landmarks) trên khuôn mặt người dùng với tốc độ khung hình ổn định (được trì ở mức ~ 30 FPS).

- **Độ chính xác:** Các điểm landmarks bám sát chuyển động của khuôn mặt ngay cả khi người dùng xoay đầu hoặc di chuyển nhanh.
- **Hiển thị:** Giao diện hiển thị trực quan lối khuôn mặt và vùng mắt, giúp người dùng nhận biết hệ thống đang hoạt động.



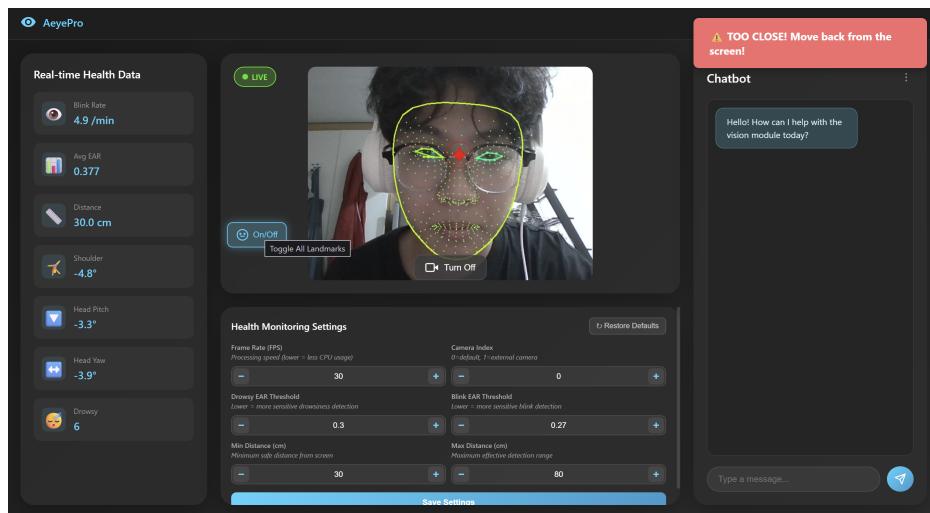
Hình 4.1: Nhận diện lưỡi khuôn mặt trong điều kiện ánh sáng bình thường

4.1.2 Giám sát khoảng cách hợp lý

Dựa trên thuật toán tam giác lượng giác được trình bày ở Chương 2, hệ thống đo lường và cảnh báo chính xác khoảng cách từ mắt người dùng đến màn hình.

Kịch bản quá gần:

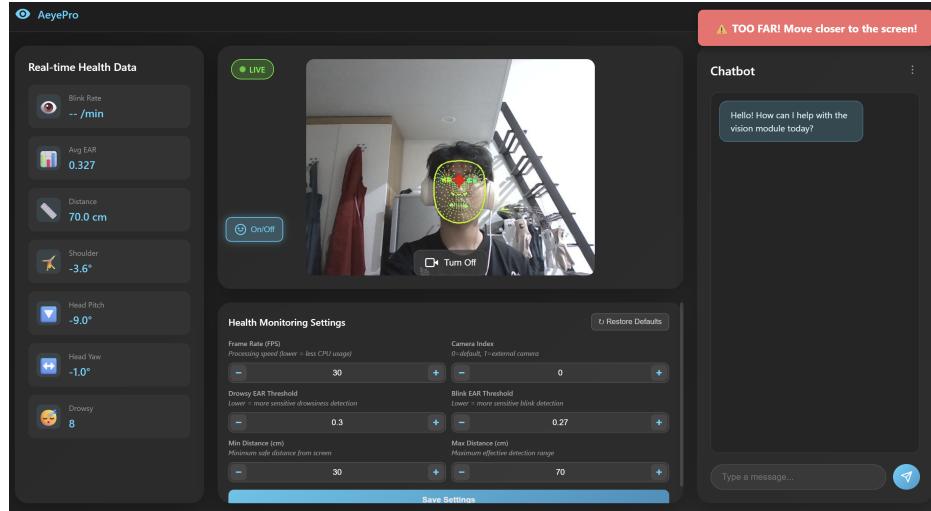
- Khi người dùng di chuyển lại gần camera dưới ngưỡng an toàn, hệ thống kích hoạt cảnh báo: “TOO CLOSE! Move back from the screen!”.
- Thủ nghiệm cho kết quả khoảng cách 30.0 cm, kích hoạt cảnh báo đỏ ngay lập tức.



Hình 4.2: Cảnh báo khoảng cách tới màn hình quá gần

Kịch bản quá xa:

- Khi người dùng ngồi quá xa, hệ thống đưa ra cảnh báo: “TOO FAR! Move closer to the screen!”.
- Thủ nghiệm ghi nhận khoảng cách 80.0 cm, đảm bảo người dùng luôn ngồi trong vùng làm việc tối ưu.

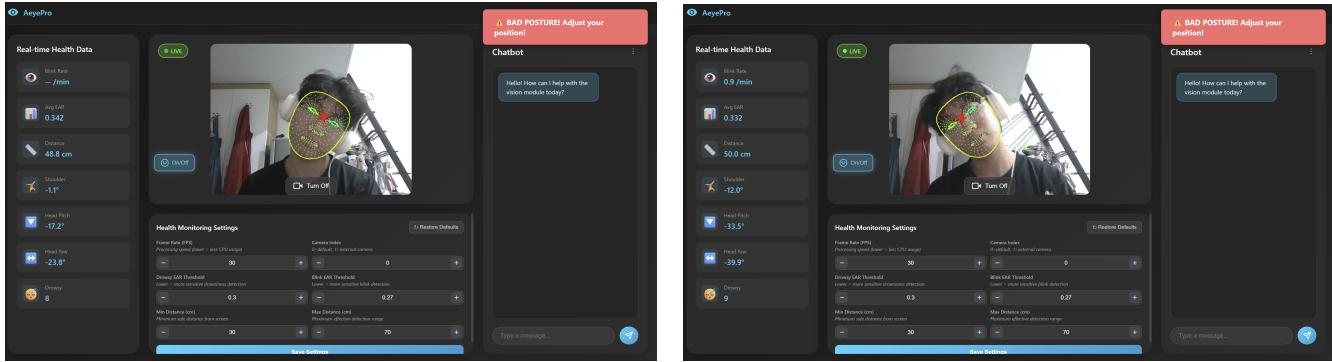


Hình 4.3: Cảnh báo khoảng cách tới màn hình quá xa

4.1.3 Phân tích và cảnh báo tư thế

Hệ thống tính toán góc của đầu và độ lệch vai để phát hiện các tư thế ngồi sai lệch.

- **Phát hiện sai tư thế:** Khi người dùng nghiêng đầu quá mức hoặc cúi gập đầu, hệ thống phát hiện sự thay đổi bất thường và đưa ra cảnh báo: “BAD POSTURE! Adjust your position!”.
- **Số liệu thực nghiệm:**
 - Trường hợp 1: Head Pitch = -17.2° , Head Yaw = -23.8° .
 - Trường hợp 2: Shoulder Tilt = -12.0° , Head Pitch = -33.5° .
- **Độ nhạy:** Cảnh báo xuất hiện khi góc nghiêng vượt ngưỡng an toàn ($\pm 15^\circ$) trong thời gian duy trì nhất định.



(a) Tư thế sai – trường hợp 1

(b) Tư thế sai – trường hợp 2

Hình 4.4: Các trường hợp hệ thống phát hiện sai tư thế

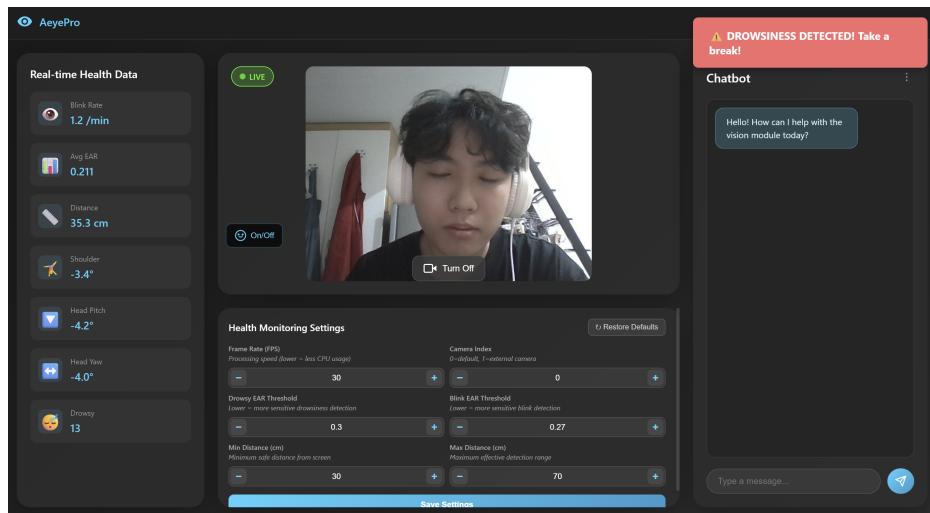
4.1.4 Phát hiện dấu hiệu buồn ngủ

Dây là tính năng quan trọng nhằm đảm bảo hiệu suất làm việc. Hệ thống sử dụng chỉ số **EAR** (Eye Aspect Ratio) để phân tích trạng thái mắt.

- Cơ chế phát hiện:** Khi người dùng nhắm mắt hoặc chớp mắt chậm kéo dài ($EAR < 0.25$), hệ thống ghi nhận dấu hiệu mệt mỏi.

Kết quả thực nghiệm:

- Hiển thị cảnh báo đỏ: “DROWSINESS DETECTED! Take a break!”.
- Bộ đếm “Drowsy” tăng lên (hình thử nghiệm ghi nhận 13 lần).
- Tần suất chớp mắt (Blink Rate) giảm mạnh xuống còn 1.2 lần/phút.

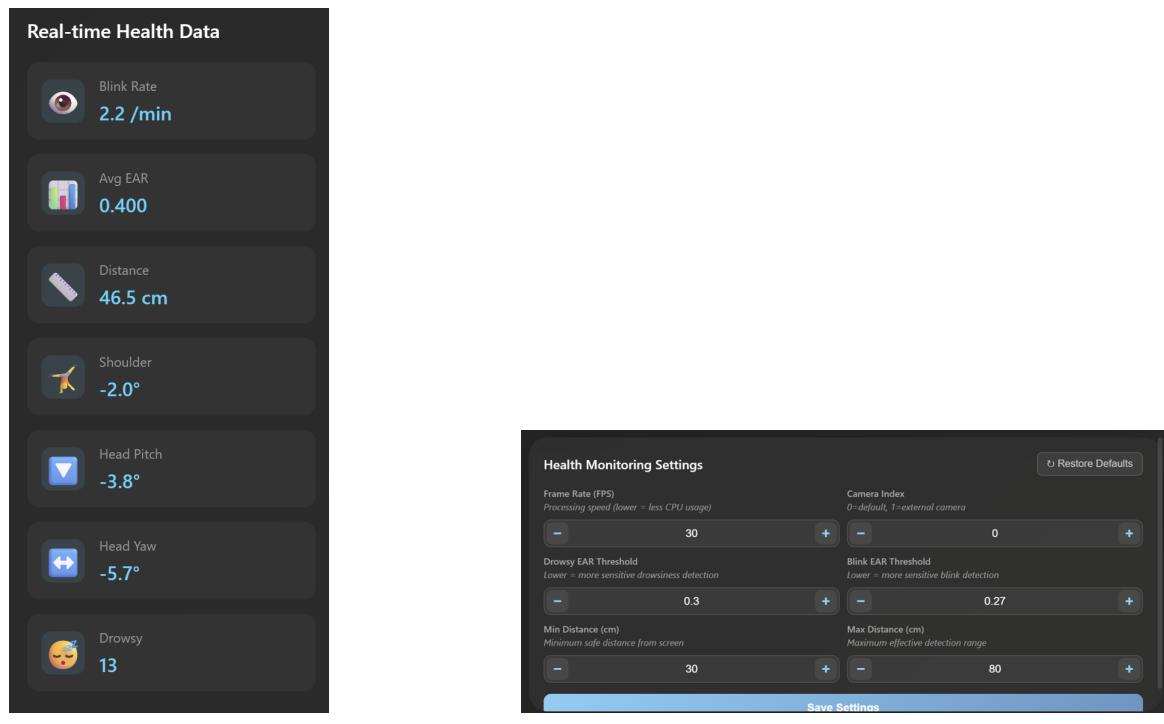


Hình 4.5: Cảnh báo buồn ngủ

4.1.5 Bảng điều khiển thông số thời gian thực

Ngoài các cảnh báo trực quan, hệ thống cung cấp bảng điều khiển cập nhật liên tục các thông số sinh trắc học.

- **Độ ổn định:** Các thông số như Avg EAR (0.357–0.400), khoảng cách (30–70 cm), và các góc Head Pitch/Yaw được cập nhật qua WebSocket với độ trễ thấp (< 100 ms).
- **Tính khả dụng:** Các nút điều chỉnh ngưỡng hoạt động chính xác, cho phép người dùng tự cấu hình độ nhạy phù hợp với môi trường và thói quen cá nhân.



(a) Hiển thị dữ liệu thời gian thực

(b) Tùy chỉnh cài đặt

Hình 4.6: Dữ liệu thời gian thực và Điều chỉnh cài đặt

4.2 Kết quả thực nghiệm hệ thống Chatbot (Adaptive RAG)

Hệ thống Chatbot được thử nghiệm dựa trên kiến trúc *Adaptive RAG* sử dụng LangGraph. Các kịch bản kiểm thử tập trung đánh giá khả năng phân loại ý định người dùng (Intent Classification), điều hướng luồng xử lý và độ chính xác của câu trả lời khi kết hợp nhiều nguồn dữ liệu. Kết quả thực nghiệm cho thấy hệ thống hoạt động đúng theo luồng xử lý đa nhánh, chi tiết như sau.

4.2.1 Luồng giao tiếp xã hội

Hệ thống ưu tiên tốc độ phản hồi đối với các câu thoại xã giao nhằm tạo cảm giác tự nhiên.

Kịch bản: Người dùng nhập câu chào: “hello”.

Xử lý của hệ thống:

- Bộ định tuyến (Router) nhận diện ý định thuộc nhóm xã giao (*social*).
- Hệ thống bỏ qua bước truy xuất dữ liệu (Retrieval) để đảm bảo phản hồi tức thời.

Kết quả: Chatbot phản hồi: “Chào bạn! Tôi có thể giúp gì cho bạn hôm nay?” với độ trễ thấp.

The screenshot shows a chatbot interface on the left and its internal state on the right. The interface has a dark theme. On the left, a message from the bot says "Hello! How can I help with the vision module today?" and a user message "hello". On the right, under "Output", there are two tabs: "MESSAGES" (1 message) and "HUMAN" (hello). Below these are sections for "ADDITIONAL FIELDS" (9 items) and "Route social". The "ADDITIONAL FIELDS" section lists the following items:

- Analyzed Intent fall_back
- Generation Chào bạn! Tôi có thể giúp gì cho bạn hôm nay?
- Original Question hello
- Reformulated Question Empty string
- Route social
- Answer Valid true
- Context []
- Retry Count 0
- Sub Queries []

(a) Câu chào hỏi giao tiếp

(b) Luồng hoạt động

Hình 4.7: Câu trả lời và luồng hoạt động với những câu hỏi xã giao

4.2.2 Luồng truy xuất kiến thức Y khoa

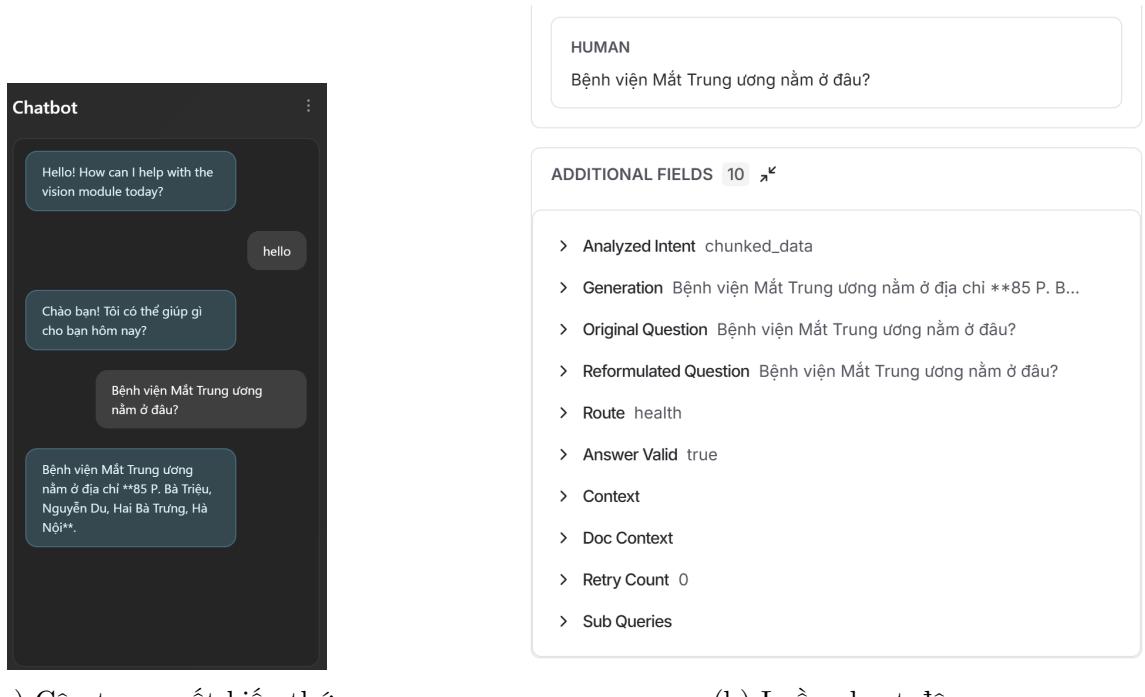
Đối với các câu hỏi liên quan đến vị trí, định nghĩa hoặc kiến thức y tế, hệ thống kích hoạt nhánh RAG với truy xuất dữ liệu từ ChromaDB.

Kịch bản: Người dùng hỏi: “Bệnh viện Mắt Trung ương nằm ở đâu?”

Xử lý của hệ thống:

- Intent được phân loại là *chunked_data*.
- Bộ định tuyến điều hướng sang nhánh *health*.
- Hệ thống truy xuất thông tin từ các tài liệu đã được nhúng trước đó.

Kết quả: Chatbot trả lời chính xác địa chỉ: “85 P. Bà Triệu, Nguyễn Du, Hai Bà Trưng, Hà Nội”.



Hình 4.8: Câu trả lời và luồng hoạt động với những câu hỏi truy xuất kiến thức

4.2.3 Luồng phân tích Dữ liệu cá nhân hóa

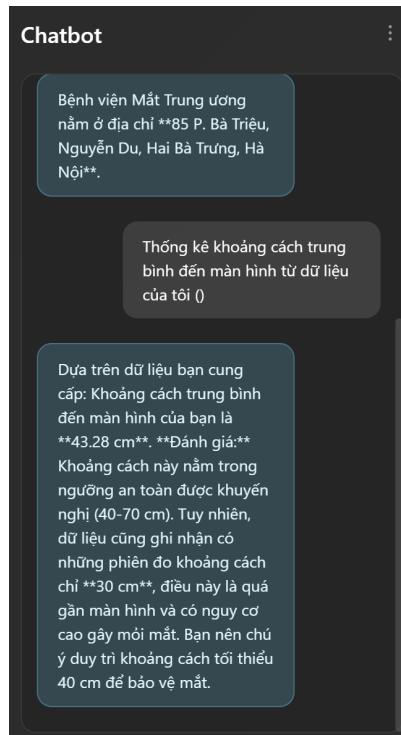
Đây là tính năng nổi bật nhất của AEyePro. Hệ thống kết hợp khả năng suy luận của LLM với xử lý số liệu thông qua Pandas Agent để phân tích dữ liệu thu được từ module Vision.

Kịch bản: Người dùng yêu cầu: “Thông kê khoảng cách trung bình đến màn hình từ dữ liệu của tôi.”

Xử lý của hệ thống:

- Intent được phân loại là *both*, nghĩa là cần vừa phân tích số liệu vừa đưa ra tư vấn.
- Pandas Agent được kích hoạt để đọc file log và thực hiện tính toán thống kê.

Kết quả: Kết quả phân tích cho thấy khoảng cách trung bình tới màn hình là *43.28 cm*, với giá trị gần nhất đạt *30.00 cm* và xa nhất là *60.32 cm*. Dựa trên so sánh với ngưỡng an toàn khuyến nghị (*40–70 cm*), Chatbot phát hiện sự bất thường khi có nhiều thời điểm khoảng cách chỉ còn *30 cm*.



Hình 4.9: Chatbot phân tích dữ liệu và đưa ra lời khuyên cá nhân hóa

Hình 4.10: Luồng hoạt động phân tích Dữ liệu cá nhân

4.2.4 Phân tích

Tổng hợp kết quả thực nghiệm từ cả hai phần *Thị giác máy tính* và *Chatbot* cho thấy kiến trúc hệ thống đề xuất hoạt động hiệu quả và đồng bộ. Về phương diện dữ liệu, hệ thống đã thiết lập được một luồng xử lý khép kín, trong đó các chỉ số sinh trắc học như EAR, khoảng cách và tư thế được module Vision thu thập chính xác theo thời gian thực, trở thành nguồn đầu vào tin cậy cho Chatbot trong quá trình phân tích và suy luận. Chatbot có thể hiểu dữ liệu người dùng, nhận diện các mẫu hành vi tiêu cực, đồng thời đưa ra khuyến nghị y tế phù hợp.

Chương 5

Kết luận

5.1 Những kết quả đạt được

Trong khuôn khổ môn học, nhóm đã xây dựng và hoàn thiện thành công hệ thống **AEye Pro** – một giải pháp hỗ trợ giám sát sức khỏe dành cho người dùng máy tính. Các kết quả đạt được bao gồm:

5.1.1 Về mặt kỹ thuật

Các kết quả kỹ thuật phản ánh khả năng tích hợp công nghệ, tối ưu hóa hệ thống và xây dựng kiến trúc phần mềm hoàn chỉnh:

- Xây dựng thành công kiến trúc phần mềm đa luồng, kết hợp giữa *Computer Vision* thời gian thực và *Generative AI*.
- Hệ thống thị giác sử dụng **MediaPipe** và **OpenCV** hoạt động ổn định với độ trễ thấp (khoảng 15–30 FPS), trích xuất chính xác các chỉ số như tỷ lệ khung hình mắt (EAR), góc nghiêng đầu và vai.
- Phát triển hệ thống Chatbot dựa trên kiến trúc **Adaptive RAG** và **LangGraph**, giải quyết hiệu quả bài toán tích hợp dữ liệu đa phương thức (văn bản y tế và dữ liệu thu được từ hệ thống thị giác).
- Thiết kế hệ thống theo mô hình **Client–Server** tách biệt, sử dụng cơ chế giao tiếp kép: RESTful API cho các tác vụ quản trị và Socket.IO cho luồng truyền tải dữ liệu thời gian thực, giúp tối ưu băng thông và giảm độ trễ.

5.1.2 Về sản phẩm

Những kết quả dưới đây phản ánh tính hoàn thiện của sản phẩm và trải nghiệm thực tế mang lại cho người dùng:

- Giao diện người dùng thân thiện, đầy đủ chức năng và dễ sử dụng.
- Hệ thống hỗ trợ các tính năng thiết thực như: cảnh báo thời gian thực và báo cáo lịch sử sức khỏe.
- Cung cấp trải nghiệm được **cá nhân hóa**: Chatbot không chỉ trả lời câu hỏi chung mà còn dựa trên dữ liệu lịch sử và trạng thái thời gian thực của người dùng để đưa ra lời khuyên cụ thể.

5.1.3 Ứng dụng thực tế

Hệ thống AEye Pro mang lại giá trị ứng dụng rõ rệt và có tính phổ cập cao. Các ứng dụng thực tế bao gồm:

- **Giải quyết vấn đề sức khỏe học đường và văn phòng:** AEye Pro hỗ trợ giảm thiểu các nguy cơ như hội chứng thị giác máy tính (CVS) và đau mỏi cổ vai gáy – những vấn đề phổ biến của học sinh, sinh viên và nhân viên văn phòng.
- **Tính phổ cập cao:** Với việc sử dụng MediaPipe chạy trên CPU và mô hình Gemini Flash tối ưu chi phí, hệ thống có thể triển khai rộng rãi trên các máy tính phổ thông mà không yêu cầu cấu hình cao, từ đó tăng khả năng áp dụng trong cộng đồng.

5.1.4 Bài học kinh nghiệm và những khó khăn

Trong suốt quá trình triển khai AEye Pro, nhóm đã tích luỹ được nhiều kinh nghiệm thực tiễn và đồng thời đối mặt với những thách thức đáng kể. Các nội dung chi tiết được trình bày dưới đây.

Bài học kinh nghiệm

Các bài học được rút ra tập trung vào tư duy kiến trúc, điều phối luồng xử lý và thiết kế AI Agent:

- **Quản lý luồng:** Nhóm nắm vững phương pháp quản lý luồng dữ liệu thời gian thực giữa Backend (Flask) và Frontend (Client) thông qua giao thức WebSocket. Bài học quan trọng là tách biệt lớp điều khiển (RESTful API) và lớp dữ liệu (Socket.IO) để duy trì độ ổn định ngay cả khi băng thông chịu tải từ luồng video.
- **Tư duy thiết kế AI Agent với LangGraph:**

- *Quản lý trạng thái:* Duy trì ngữ cảnh hội thoại qua các bước xử lý phức tạp.
- *Cơ chế tự sửa lỗi:* Thiết kế các vòng lặp cho phép Agent tự đánh giá kết quả và thực hiện lại tác vụ nếu chưa đạt yêu cầu, tránh trả về kết quả sai.

Những khó khăn và thách thức

Quá trình phát triển đặt ra nhiều vấn đề kỹ thuật phức tạp; nhóm đã phân tích và giải quyết như sau:

- **Thách thức về độ trễ và đồng bộ dữ liệu:** Việc duy trì tốc độ khung hình ổn định (15–30 FPS) trong khi Backend xử lý tác vụ RAG nặng gây ra xung đột tài nguyên.
 - *Vấn đề:* Xung đột CPU và bộ nhớ dẫn đến phản hồi chậm từ Chatbot.
 - *Giải quyết:* Tối ưu cơ chế khóa luồng và sử dụng hàng đợi để tổ chức xử lý tác vụ bất đồng bộ.
- **Xử lý dữ liệu không đồng nhất và ảo giác của LLM:** Tích hợp dữ liệu định lượng như Log/CSV vào mô hình ngôn ngữ gây khó khăn.
 - *Vấn đề:* LLM có thể tính toán sai hoặc “ảo giác” khi tự diễn giải dữ liệu thô.
 - *Giải quyết:* Sử dụng *Pandas Agent* chuyên biệt để thực thi phép tính thống kê, đồng thời tinh chỉnh Prompt để hướng Agent sử dụng công cụ đúng ngữ cảnh.

5.2 Hướng phát triển

Để đưa hệ thống AEye Pro từ một đồ án môn học trở thành một sản phẩm hoàn chỉnh có thể triển khai rộng rãi, nhóm đề xuất lộ trình phát triển theo hai giai đoạn: ngắn hạn và dài hạn, cùng với các định hướng phương pháp thực hiện như sau.

5.2.1 Mục tiêu ngắn hạn

Trong giai đoạn đầu, mục tiêu trọng tâm là cải thiện trải nghiệm người dùng và tối ưu hóa hoạt động của hệ thống:

- **Tối ưu hóa trải nghiệm người dùng (UX):** Bổ sung các biểu đồ trực quan hóa dữ liệu trên Dashboard để người dùng theo dõi xu hướng sức khỏe theo tuần hoặc tháng.
- **Cải thiện thuật toán:** Tinh chỉnh các ngưỡng cảnh báo dựa trên phản hồi thực tế nhằm giảm tỷ lệ cảnh báo sai, đặc biệt trong điều kiện ánh sáng yếu hoặc các yếu tố ngoại cảnh.
- **Đóng gói cài đặt:** Xây dựng bộ cài đặt (.exe/installer) chuyên nghiệp, tự động cài đặt môi trường Python và các thư viện phụ thuộc, giúp người dùng phổ thông có thể cài đặt một cách thuận tiện.

5.2.2 Mục tiêu dài hạn

Giai đoạn dài hạn tập trung vào mở rộng hệ thống, nâng cao khả năng tích hợp và mở rộng phạm vi ứng dụng:

- **Phát triển phiên bản Mobile và Cross-platform:** Xây dựng phiên bản cho thiết bị di động, cho phép sử dụng camera điện thoại để giám sát và nhận cảnh báo trực tiếp qua rung hoặc thông báo đầy.
- **Điện toán đám mây và Đồng bộ hóa:** Xây dựng hệ thống Backend Cloud giúp lưu trữ và đồng bộ dữ liệu sức khỏe giữa nhiều thiết bị. Từ đó, Chatbot có thể thu thập dữ liệu và đưa ra lời khuyên được chính xác hơn cho từng nhóm người dùng.
- **Hướng phát triển dài hạn:** Nâng cấp hệ thống từ mô hình Adaptive RAG thụ động sang kiến trúc *Agentic AI* chủ động. Mục tiêu là xây dựng một trợ lý y tế có khả năng tự chủ thực hiện các tác vụ phức tạp như: tự động đặt lịch hẹn với bác sĩ khi phát hiện bất thường, gửi cảnh báo khẩn cấp dựa trên dữ liệu sinh trắc học thời gian thực, và tích hợp sâu với hệ sinh thái IoT y tế cá nhân.

5.2.3 Phương pháp thực hiện

Để hiện thực hóa các mục tiêu trên, nhóm đề xuất các định hướng kỹ thuật và phương pháp triển khai cụ thể như sau:

- **Nghiên cứu và Tích hợp Small Language Models (SLM):** Để giảm sự phụ thuộc vào kết nối mạng và đảm bảo tuyệt đối quyền riêng tư cho người dùng, nhóm sẽ nghiên cứu thay thế hoặc kết hợp mô hình đám mây (Gemini Flash) bằng các mô hình ngôn ngữ nhỏ gọn chạy trực tiếp trên thiết bị như Google Gemma 2. Điều này cho phép Chatbot hoạt động ngoại tuyến với tốc độ phản hồi nhanh hơn.
- **Chuyển đổi sang Kiến trúc Microservices:** Hiện tại hệ thống đang chạy trên một ứng dụng lớn với Flask. Để phục vụ mục tiêu đồng bộ hóa đám mây và đa nền tảng, hệ thống cần được tách nhỏ thành các dịch vụ độc lập (Microservices):
 - Dịch vụ thị giác máy tính (Vision Service).
 - Dịch vụ RAG và Chatbot (Chatbot Service).
 - Dịch vụ quản lý người dùng và dữ liệu (User & Data Service).

Việc này giúp dễ dàng mở rộng quy mô và bảo trì từng phần riêng biệt.

- **Cách thức thực hiện:** Tập trung mở rộng kho công cụ (Tools) cho Agent thông qua việc tích hợp các API ngoại vi (như Google Calendar API, Hospital RESTful API). Bên cạnh đó, tối ưu hóa cơ chế *Long-term Memory* để hệ thống ghi nhớ lịch sử bệnh lý trọn đời và áp dụng các mô hình suy luận đa bước (Multi-step Reasoning) trên nền tảng LangGraph để giải quyết các kịch bản ra quyết định y tế phức tạp.
- **Hợp tác chuyên môn:** Kết hợp với lời khuyên từ các chuyên gia y tế để chuẩn hóa bộ dữ liệu tri thức (Knowledge Base). Điều này đảm bảo các câu trả lời luôn tuân thủ các kiến thức y tế hiện hành.

Tài liệu tham khảo

- [1] V. Ramesh, A. S. Al-Fahdawi, et al., "MediaPipe Iris and Kalman Filter for Robust Eye Gaze Tracking," *Proc. ICSICE 2024*, Advances in Computer Science Research, vol. 120, pp. 1631–1641, Atlantis Press, 2025. https://doi.org/10.2991/978-94-6463-718-2_136
- [2] Y. Q. Lin, X. Y. Jiao, L. Zhao, "Detection of 3D Human Posture Based on Improved Mediapipe," *Journal of Computer and Communications*, vol. 11, pp. 102–121, 2023. <https://doi.org/10.4236/jcc.2023.112008>
- [3] W. Zhang et al., "Combined MediaPipe and YOLOv5 range of motion assessment system for spinal diseases and frozen shoulder," *Scientific Reports*, vol. 14, art. no. 15879, July 2024. <https://doi.org/10.1038/s41598-024-66221-8>
- [4] K. A. Phu, V. D. Hoang, V. T. L. Le, "Performance Evaluation of MediaPipe and OpenPose for Skeleton Data Extraction," *Proc. ICSICE 2023*, 2023.
- [5] T. Soukupová and J. Čech, "Real-Time Eye Blink Detection using Facial Landmarks," *21st Computer Vision Winter Workshop (CVWW)*, 2016.
- [6] K. K. Hansraj, "Assessment of stresses in the cervical spine caused by posture and position of the head," *Surgical Technology International*, vol. 25, pp. 277-279, 2014.
- [7] V. Bazarevsky, I. Grishchenko, K. Raveendran, T. Zhu, F. Zhang, and M. Grundmann, "BlazePose: On-device real-time body pose tracking," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2020.
- [8] L. McAtamney and E. N. Corlett, "RULA: a survey method for the investigation of work-related upper limb disorders," *Applied Ergonomics*, vol. 24, no. 2, pp. 91–99, 1993.
- [9] Pallets Projects, *The Application Context*, Flask Documentation, Available: <https://flask.palletsprojects.com/en/stable/appcontext/>
- [10] Ritesh Patel, *REST is not about APIs, Part 1*, Nirmata Blog, October 1, 2013, Available: <https://nirmata.com/2013/10/01/rest-apis-part-1/>

- [11] Socket.IO Documentation, *How it works*, v4, Available: <https://socket.io/docs/v4/how-it-works/>
- [12] LangChain, *Build a RAG agent with LangChain*, LangChain Documentation, Available: <https://docs.langchain.com/oss/python/langchain/rag>
- [13] Real Python, *Embeddings and Vector Databases With ChromaDB*, Real Python Tutorials, Available: <https://realpython.com/chromadb-vector-database/>
- [14] Svetmaraju, *Vector database: What is chromadb doing under the hood?*, DEV Community, Available: <https://dev.to/svetmaraju/vector-database-what-is-chromadb-doing-under-the-hood-177j>
- [15] AWS, *What is RAG? - Retrieval-Augmented Generation AI Explained*, Amazon Web Services, Available: <https://aws.amazon.com/what-is/retrieval-augmented-generation/>
- [16] Analytics Vidhya, *Introduction to HNSW: Hierarchical Navigable Small World*, Analytics Vidhya Blog, October 2023, Available: <https://www.analyticsvidhya.com/blog/2023/10/introduction-to-hnsw-hierarchical-navigable-small-world/>
- [17] Meilisearch, *14 types of RAG (Retrieval-Augmented Generation)*, Meilisearch Blog, Available: <https://www.meilisearch.com/blog/rag-types>
- [18] LangChain AI, *Adaptive RAG*, LangGraph Tutorials, Available: https://langchain-ai.github.io/langgraph/tutorials/rag/langgraph_adaptive_rag/
- [19] ProjectPro, *The Self-RAG Shortcut Every AI Expert Wishes They Knew*, ProjectPro, Available: <https://www.projectpro.io/article/self-rag/1176>
- [20] Gabriel Gomes, PhD, *Advanced RAG Techniques — The Adaptive-RAG strategy*, Medium, Available: <https://gabrielgomes61320.medium.com/advanced-rag-techniques-the-adaptive-rag-strategy-d4e6879b39b8>
- [21] MDPI, *Hallucination Mitigation for Retrieval-Augmented Large Language Models: A Review*, Mathematics 2025, 13(5), 856, Available: <https://www.mdpi.com/2227-7390/13/5/856>
- [22] Internal Document, *Design docs RAG.docx*, Unpublished Project Documentation.
- [23] Madhukar Kumar, *Chapter 1 — How to Build Accurate RAG Over Structured and Semi-structured Databases*, Medium, Available: <https://medium.com/madhukarkumar/chapter-1-how-to-build-accurate-rag-over-structured-and-semi-structured-databases-996cc>
- [24] PromptHub, *Least-to-Most Prompting Guide*, PromptHub Blog, Available: <https://www.prompthub.us/blog/least-to-most-prompting-guide>

- [25] Learn Prompting, *Least-to-Most Prompting*, Learn Prompting Documentation, Available: https://learnprompting.org/docs/intermediate/least_to_most
- [26] Towards AI, *Refining RAG: Advanced Query Strategies, Prompt Mastery, and Precise Evaluation*, Towards AI, Available: <https://pub.towardsai.net/refining-rag-advanced-query-strategies-prompt-mastery-and-precise-evaluation-2ed031be92>
- [27] Diptanshu Gautam, *Mastering Query Decomposition for Efficient Information Retrieval*, Medium, Available: <https://medium.com/@diptanshu.gautam/mastering-query-decomposition-for-efficient-information-retrieval-ea6b3c1bc184>
- [28] Deepset, *Advanced RAG: Query Decomposition & Reasoning*, Haystack Blog, Available: <https://haystack.deepset.ai/blog/query-decomposition>
- [29] Tejpal Kumawat, *Beyond Basic RAG: Mastering Routing, Query Construction, and Advanced Retrieval — part 2*, Medium, Available: <https://medium.com/@tejpal.abhyuday/beyond-basic-rag-mastering-routing-query-construction-and-advanced-retrieval-part-2-bdf>
- [30] YouTube, *Logical vs. Semantic Routing: A Deep Dive into RAG Query Routing (Part 10)*, Video Resource, Available: <https://www.youtube.com/watch?v=h85608ly7mE>
- [31] Towards Data Science, *Routing in RAG Driven Applications*, Towards Data Science, Available: <https://towardsdatascience.com/routing-in-rag-driven-applications-a685460a7220/>
- [32] Asai et al., *Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection*, arXiv preprint arXiv:2310.11511, Available: <https://arxiv.org/abs/2310.11511>
- [33] Pinecone, *Advanced RAG Techniques*, Pinecone Learn, Available: <https://www.pinecone.io/learn/advanced-rag-techniques/>