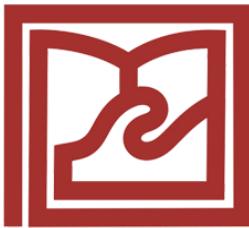


ĐẠI HỌC DUY TÂN
TRƯỜNG KHOA HỌC MÁY TÍNH

慈惠堂



ĐỒ ÁN CHUYÊN NGÀNH: KHOA HỌC DỮ LIỆU

Tên đề tài:

ĐIỀU KHIỂN GAME ĐUA XE BẰNG NHẬN DIỆN CỦ CHỈ TAY

GVHD: PHẠM VĂN DƯỢC

LỚP: DS 445 K

Thành viên :

Trịnh Minh Sơn - 4373

Nguyễn Đình Huy Khang - 2483

Bùi Vĩnh Lợi - 0056

Đinh Tấn Phúc -

Đà Nẵng, tháng 10 năm 2025

Mục lục

Contents

Mục lục	2
MỞ ĐẦU	5
CHƯƠNG 1: GIỚI THIỆU	7
1.1. Lý do chọn đề tài	7
1.2. Mục tiêu dự án	7
1.3 Đối tượng nghiên cứu	8
1.4. Phạm vi nghiên cứu	10
CHƯƠNG 2: CƠ SỞ LÝ THUYẾT	10
2.1. Nhận diện cử chỉ tay	10
2.1.1 .Khái niệm	10
2.1.2. Ứng dụng	11
2.2. Mạng nơ-ron tích chập (CNN) – MobileNetV2	11
2.2.1. Kiến trúc CNN (Convolutional Neural Network)	11
2.2.2. Thành phần chính của CNN	11
2.2.3. Vai trò của CNN trong trích xuất đặc trưng ảnh.....	13
2.2.4. Tích hợp trong dự án	14
2.3. Mạng hồi tiếp (RNN/GRU)	14
2.4 MediaPipe Hands.....	15
2.5 Thư viện hỗ trợ	16
CHƯƠNG 3: PHÂN TÍCH HỆ THỐNG	17
3.1 Mục tiêu & Tóm tắt chức năng hệ thống.....	17
3.2. Yêu cầu chức năng (Functional requirements) — chi tiết & tiêu chí chấp nhận ...	18
3.3. Yêu cầu phi chức năng (Non-functional requirements)	19
3.4. Kiến trúc tổng thể (Component overview).....	20
3.5 Mô tả chi tiết các module	20
3.6. Định dạng dữ liệu & manifest	22
3.7 Luồng dữ liệu & quy trình (end-to-end).....	23
3.8 Xử lý ngoại lệ & lỗi.....	23

3.9 Kiểm thử & Đánh giá (Test plan)	24
3.10 Yêu cầu phần cứng & phần mềm	25
3.11 Bảo mật & Quyền riêng tư.....	25
3.12 Khả năng mở rộng & Bảo trì	25
3.13 Rủi ro & Biện pháp giảm thiểu.....	26
3.14. Kiến trúc tổng thể	26
○ (Tùy chọn) áp dụng smoothing hoặc threshold để ổn định kết quả.	27
CHƯƠNG 4: DỮ LIỆU VÀ TIỀN XỬ LÝ	28
4.1. Pipeline xử lý dữ liệu.....	28
4.2. Tổng quan kiến trúc và luồng dữ liệu.....	28
4.3. Thành phần (module) chi tiết.....	28
4.3.1 Capture Module (capture_gesture_dataset.py).....	28
4.3.2 Hand Preprocessor (MediaPipe – tùy chọn).....	29
4.3.3 Preprocessing.....	29
4.3.4 Frame Buffer.....	29
4.3.5 Feature extractor (MobileNetV2 backbone).....	30
4.3.6 Sequence model (GRU).....	30
4.3.7 Postprocessing & Smoothing	30
4.3.8 Controller Mapper	31
4.3.9 Game Runner (Pygame)	31
4.4. Thiết kế game	32
CHƯƠNG 5: CÀI ĐẶT VÀ THỰC NGHIỆM.....	33
5.1 Công cụ và môi trường	33
5.2 Quy trình thực hiện.....	33
5.3 Kết quả thực nghiệm.....	34
CHƯƠNG 6: ĐÁNH GIÁ & BÀI HỌC KINH NGHIỆM	35
 6.1. Đánh giá ưu điểm.....	35
 6.2. Hạn chế	35
 6.3. Bài học kinh nghiệm	35
CHƯƠNG 7: HƯỚNG PHÁT TRIỂN	36

7.1 Mở rộng tập cử chỉ	36
7.2 Smoothing và Ensemble	36
7.3 Tối ưu giao diện và trải nghiệm người dùng (UI/UX)	36
7.4 Hỗ trợ nhiều người chơi (Multiplayer)	37
7.5 Tích hợp với thiết bị AR/VR	37
7.6 Tối ưu mô hình cho thiết bị di động	37
7.7 Ứng dụng ngoài game.....	38
KẾT LUẬN	38

MỞ ĐẦU

Trong bối cảnh thị giác máy tính (Computer Vision) và trí tuệ nhân tạo (Artificial Intelligence – AI) phát triển mạnh, phương thức tương tác người–máy đang chuyển dịch từ bàn phím/chuột sang các hình thức tự nhiên hơn như giọng nói, khuôn mặt, cử động cơ thể và đặc biệt là cử chỉ tay. Nhận diện cử chỉ tay (Hand Gesture Recognition) là một hướng giàu tiềm năng vì cử chỉ là ngôn ngữ phi lời nói phổ biến, giúp tương tác trở nên trực quan và ít rào cản.

Ở lĩnh vực trò chơi điện tử, nhất là game đua xe – nơi yêu cầu phản xạ nhanh và thao tác chính xác – điều khiển bằng cử chỉ có thể tăng tính nhập vai so với các thiết bị truyền thống. Tuy nhiên, triển khai thời gian thực trên phần cứng phổ thông gặp nhiều thách thức: điều kiện ánh sáng đa dạng, độ trễ xử lý, dữ liệu thiếu bối cảnh và nhầm lẫn giữa các cử chỉ có hình thái/động học giống nhau (điển hình là Left Swipe và Right Swipe).

Từ thực tiễn đó, nhóm thực hiện đề tài “Điều khiển game đua xe bằng nhận diện cử chỉ tay” với các mục tiêu:

- Thu nhận hình ảnh bàn tay người chơi từ webcam theo chuỗi 30 khung hình/sequence.
- Sử dụng học sâu để nhận diện 5 cử chỉ: Thumbs Up, Thumbs Down, Left Swipe, Right Swipe, Stop; ánh xạ thành hành động lái xe.
- Tích hợp pipeline nhận diện vào game xây dựng bằng Pygame, đạt trải nghiệm thời gian thực trên phần cứng phổ thông.

Phương pháp tiếp cận:

- Trích chọn đặc trưng hình ảnh bằng MobileNetV2 (224×224 , pretrained ImageNet), kết hợp mô hình GRU để học động học chuỗi 30 khung hình.
- Hỗ trợ tiền xử lý bằng MediaPipe Hands (crop/định vị bàn tay) nhằm tăng ổn định và giảm nhiễu bối cảnh.

- Thu thập bổ sung dữ liệu “đúng bối cảnh game” bằng script riêng, lưu theo cấu trúc archive/train|val/<sequence>/frame_XXX.png và manifest CSV (sequence;label_slug;label_index).
- Huấn luyện/fine-tune trên Python 3.10, TensorFlow 2.13; dùng các kỹ thuật thực dụng (augmentation nhẹ, EarlyStopping, ReduceLROnPlateau) để cân bằng độ chính xác và tốc độ suy luận.

Đóng góp chính của đề tài:

- Xây dựng đầy đủ pipeline real-time: webcam → (MediaPipe) tiền xử lý → MobileNetV2+GRU → ánh xạ điều khiển game Pygame.
- Công cụ thu thập dữ liệu theo bối cảnh, kịch bản gán nhãn 5 cử chỉ và manifest CSV phục vụ huấn luyện lặp.
- Tối ưu triển khai để chạy mượt trên laptop GPU RTX 3050 Ti (4 GB VRAM), có chế độ bật/tắt MediaPipe phục vụ trình diễn.
- Tài liệu hoá và tự động hoá quy trình huấn luyện, lưu mô hình (.h5) và tích hợp vào runner.

Phạm vi và giới hạn:

- Tập trung 5 cử chỉ điều khiển cơ bản của game đua xe; không xử lý đa bàn tay/đa người chơi cùng lúc.
- Độ chính xác/độ trễ phụ thuộc góc quay, ánh sáng và biên độ cử chỉ; đã bổ sung dữ liệu bối cảnh và smoothing để giảm nhầm lẫn Left/Right.
- Không tối ưu phần đồ họa nâng cao; ưu tiên tính ổn định và tái lập thí nghiệm.

CHƯƠNG 1: GIỚI THIỆU

1.1. Lý do chọn đề tài

Trong kỷ nguyên số, **tương tác người – máy (Human–Computer Interaction, HCI)** ngày càng trở nên quan trọng trong việc nâng cao trải nghiệm và hiệu quả sử dụng công nghệ. Nếu như trước đây, con người chủ yếu giao tiếp với máy tính thông qua **thiết bị ngoại vi truyền thống** như bàn phím, chuột, hoặc tay cầm, thì hiện nay, xu hướng chung đang dịch chuyển sang những phương thức **tự nhiên hơn** như giọng nói, nhận diện khuôn mặt, theo dõi ánh mắt, và đặc biệt là **cử chỉ tay**.

Nhận diện cử chỉ tay được xem là công nghệ nổi bật vì:

- **Bàn tay là công cụ giao tiếp linh hoạt và trực quan:** con người sử dụng tay để chỉ dẫn, ra hiệu và biểu đạt cảm xúc trong đời sống hàng ngày.
- **Khả năng ứng dụng rộng:** từ điều khiển thiết bị gia dụng, robot, hệ thống VR/AR, đến trò chơi điện tử và giáo dục tương tác.
- **Mang lại trải nghiệm mới mẻ:** thay vì phải nhấn phím hoặc cầm tay cầm, người dùng có thể trực tiếp điều khiển bằng hành động quen thuộc.

Trong lĩnh vực **trò chơi điện tử (game)**, sự nhập vai và tính chân thực là yếu tố quan trọng để tăng tính hấp dẫn. Thể loại **game đua xe** vốn đòi hỏi sự tập trung cao độ và phản xạ nhanh, nên việc điều khiển bằng cử chỉ tay hứa hẹn mang lại một **trải nghiệm tự nhiên, hứng thú và gần gũi hơn**.

Do đó, đề tài “**Điều khiển game đua xe bằng nhận diện cử chỉ tay**” vừa mang tính học thuật (áp dụng AI, Deep Learning, Computer Vision) vừa mang tính thực tiễn (tạo sản phẩm giải trí sáng tạo, có tiềm năng ứng dụng trong VR/AR, robot, và hệ thống hỗ trợ thông minh).

1.2. Mục tiêu dự án

Dự án được triển khai với mục tiêu xây dựng một hệ thống **điều khiển game đua xe bằng cử chỉ tay trong thời gian thực**, đảm bảo các yêu cầu sau:

- Xây dựng hệ thống **nhận diện 5 cử chỉ tay thời gian thực** từ webcam (chuỗi 30 frame/sequence).
- **Hệ thống hoạt động ổn định** với tốc độ xử lý đủ nhanh, đáp ứng yêu cầu thời gian thực.

- **Mở rộng dễ dàng:** cho phép bổ sung thêm dữ liệu huấn luyện và thêm các cử chỉ mới trong tương lai.
- **Hoàn thiện sản phẩm cuối cùng,** bao gồm:
 - Game runner (Pygame) + HUD thể hiện top xác suất.
 - Công cụ thu thập/ghi nhãn theo manifest CSV.
 - Mô hình học sâu (MobileNetV2 + GRU) lưu ở định dạng .h5.
 - Tài liệu hướng dẫn cài đặt, chạy và huấn luyện lại.
- Cải thiện độ ổn định Left/Right bằng MediaPipe tiền xử lý, augmentation nhẹ và logic smoothing.

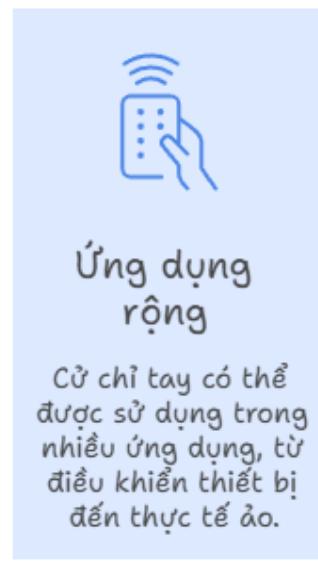
1.3 Đối tượng nghiên cứu

Đối tượng nghiên cứu chính của đề tài bao gồm:

- **Các cử chỉ tay cơ bản** (gắn liền với hành động trong game đua xe):
 -  *Thumbs Up* → **Tăng tốc**
 -  *Thumbs Down* → **Giảm tốc**
 -  *Left Swipe* → **Rẽ trái**
 -  *Right Swipe* → **Rẽ phải**
 -  *Stop* → **Phanh dừng**

- Pipeline xử lý tổng thể:

Tại sao nên phát triển hệ thống điều khiển game bằng cử chỉ tay?



1. **Dữ liệu:** thu thập hình ảnh bàn tay từ webcam, tiền xử lý (resize, chuẩn hóa, ROI).
2. **Mô hình:** MobileNetV2 (trích đặc trưng ảnh) + GRU (động học chuỗi 30 frame) → Softmax 5 lớp.
3. **Ứng dụng game:** mô hình xuất nhãn cử chỉ → ánh xạ thành hành động trong trò chơi đua xe.

Đối tượng nghiên cứu cũng bao gồm việc **tích hợp MediaPipe** như một tùy chọn để hỗ trợ phát hiện bàn tay, giúp mô hình hoạt động ổn định hơn trong môi trường phức tạp (ánh sáng thay đổi, nền nhiễu).

1.4. Phạm vi nghiên cứu

Để đảm bảo tính khả thi trong thời gian và nguồn lực có hạn, phạm vi nghiên cứu của đề tài được giới hạn như sau:

- **Cử chỉ tay:** chỉ tập trung vào **5 cử chỉ cơ bản** (tăng tốc, giảm tốc, rẽ trái, rẽ phải, phanh). Các cử chỉ khác (giữ làn, nitro, đổi camera, vẫy tay...) chưa được tích hợp ở giai đoạn này.
- **Trò chơi:** phát triển game đua xe **2D trên nền Pygame**; chưa triển khai mô hình 3D hay VR.
- **Môi trường & Công cụ:**
 - Hệ điều hành: Windows 11; Ngôn ngữ: Python 3.10.
 - Thư viện chính: TensorFlow 2.13, OpenCV, NumPy, Pygame; MediaPipe (tùy chọn)
- Phần cứng: GPU RTX 3050 Ti để huấn luyện mô hình và kiểm thử **Ứng dụng**: tập trung vào **minh chứng nguyên mẫu (prototype)**, chưa tối ưu cho thương mại hóa hoặc đa nền tảng (Linux, macOS, thiết bị di động).

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1. Nhận diện cử chỉ tay

2.1.1. Khái niệm

- Là quá trình máy tính **phát hiện** và **phân loại** các động tác / cử chỉ của bàn tay từ ảnh hoặc video.
- Thuộc giao thoa giữa **Computer Vision (thị giác máy tính)** và **HCI (Human–Computer Interaction)**, nhằm tạo tương tác tự nhiên hơn giữa người và máy.
- Cử chỉ tay có thể coi là ngôn ngữ phi lời nói, hỗ trợ tương tác tự nhiên hơn giữa người và máy, đặc biệt trong môi trường không dùng chuột hay bàn phím.

Các ứng dụng thực tế

- VR / AR: điều khiển menu, xoay vật thể 3D bằng tay
- Robot / tự động hóa: ra lệnh cho robot (dừng, đi, rẽ)
- Game & giải trí: điều khiển nhân vật hoặc xe đua bằng tay
- Y tế & phục hồi chức năng: theo dõi cử động tay, hỗ trợ bệnh nhân giao tiếp

- Smart Home / IoT: điều khiển thiết bị bằng cử chỉ
- Hỗ trợ lái xe: điều khiển hệ thống giải trí, nhận/gọi điện không cần chạm

2.1.2. Ứng dụng

Thực tế ảo (VR) và thực tế tăng cường (AR)

- Tương tác trực tiếp với môi trường ảo mà không cần bộ điều khiển cầm tay.
- Ví dụ: ra hiệu để mở menu, xoay vật thể 3D bằng tay trong kính VR.

Robot và tự động hóa

- Ra lệnh cho robot bằng cử chỉ tay (Stop, Go, Turn Left/Right).
- Dùng trong sản xuất công nghiệp, robot dịch vụ, hoặc robot hỗ trợ y tế.

Game và giải trí

- Điều khiển nhân vật, xe đua, hoặc thực hiện hành động trong game bằng tay.
- Tạo trải nghiệm trực quan, hấp dẫn hơn so với bàn phím/chuột.

Y tế & phục hồi chức năng

- Theo dõi cử động bàn tay trong quá trình tập vật lý trị liệu.
- Hỗ trợ bệnh nhân mất khả năng nói bằng cử chỉ tay.

Giao tiếp người – máy nói chung

- Hệ thống thông minh (nhà thông minh, IoT): ra hiệu bật đèn, điều chỉnh TV, mở nhạc.
- Hỗ trợ lái xe: điều khiển hệ thống giải trí, gọi điện bằng cử chỉ.

2.2. Mạng nơ-ron tích chập (CNN) – MobileNetV2

2.2.1. Kiến trúc CNN (Convolutional Neural Network)

CNN là một loại mạng nơ-ron nhân tạo chuyên dùng để xử lý dữ liệu dạng lưới (grid-like data), ví dụ như ảnh (2D pixel grid).

Thay vì kết nối dense (fully connected) như MLP, CNN dùng phép tích chập (convolution) để trích xuất đặc trưng cục bộ từ ảnh.

2.2.2. Thành phần chính của CNN

Lớp Convolution (Conv Layer)

- Sử dụng **bộ lọc (kernel)** trượt qua ảnh để phát hiện các đặc trưng cục bộ.

- Ví dụ:
 - Tầng đầu → phát hiện cạnh (edge, corner).
 - Tầng giữa → phát hiện hình dạng (shape, contour).
 - Tầng sâu → phát hiện đặc trưng trừu tượng (vật thể, bàn tay).

Hàm kích hoạt (Activation Function)

- Thường dùng **ReLU** để tạo tính phi tuyến, giúp mô hình học được đặc trưng phức tạp.

Pooling Layer

- Giảm kích thước (downsampling), giữ lại đặc trưng quan trọng.
- Phổ biến nhất: **Max Pooling** (chọn giá trị lớn nhất trong vùng).

Fully Connected Layer (FC)

- Sau khi qua nhiều lớp tích chập và pooling, đặc trưng được “làm phẳng” (flatten) rồi đưa vào **layer kết nối đầy đủ** để phân loại.

Softmax Output

- Tạo xác suất dự đoán cho từng lớp (ở đây là 5 cử chỉ tay).

2.2.3. Vai trò của CNN trong trích xuất đặc trưng ảnh

- **Tự động học đặc trưng:** Không cần thiết kế thủ công (ví dụ như HOG, SIFT trong xử lý ảnh truyền thống).
- **Phân cấp đặc trưng:**
 - Tầng nông → đặc trưng thấp (cạnh, góc).
 - Tầng sâu → đặc trưng cao (bàn tay, cử chỉ).
- **Bất biến cục bộ (Local Invariance):** Nhận diện được bàn tay/cử chỉ ngay cả khi dịch chuyển nhẹ, thay đổi góc nhìn.
- **Khả năng khái quát tốt:** Giúp mô hình nhận diện cử chỉ trong nhiều điều kiện ánh sáng, nền khác nhau.

MobileNetV2 – kiến trúc nhẹ phù hợp cho thiết bị “thực tế”

- MobileNetV2 là biến thể có thiết kế nhẹ và tối ưu hóa cho thiết bị di động / nền tảng có tài nguyên hạn chế (CPU, GPU yếu).
- Điểm nổi bật:
 - Sử dụng **depthwise separable convolution** (phép tích chập tách biệt) để giảm số tham số và chi phí tính toán so với convolution truyền thống.
 - Thiết kế các **bottleneck residual block** (block rút gọn) để giữ hiệu năng cao nhưng giảm độ phức tạp.
 - Có khả năng fine-tune (giữ pre-trained weights từ ImageNet, sau đó tinh chỉnh cho bài toán của bạn).

2.2.4. Tích hợp trong dự án

Trong dự án game đua xe:

- **MobileNetV2** được dùng làm **CNN backbone**.
- Nó trích xuất vector đặc trưng từ từng frame ảnh ($224 \times 224 \times 3 \rightarrow 1280$ chiều).
- Vector đặc trưng này sau đó được đưa vào **GRU** để xử lý theo chuỗi thời gian → phân loại cờ chỉ.

2.3. Mạng hồi tiếp (RNN/GRU)

RNN & nhu cầu xử lý chuỗi

- Video hay chuỗi ảnh là dữ liệu có thứ tự theo thời gian — cần mô hình có khả năng ghi nhớ trạng thái trước đó để dự đoán hiện tại.
- RNN (Recurrent Neural Network) là dạng mạng có “lưu trạng thái” (state) truyền qua các bước thời gian để xử lý chuỗi.

GRU (Gated Recurrent Unit)

- GRU là một biến thể của RNN giúp giải quyết một số vấn đề của mạng RNN truyền thống (Nha vanishing gradient).
- GRU đơn giản hơn LSTM (ít công hơn) — gồm gate update và gate reset — vẫn giữ hiệu quả học chuỗi tốt.
- Ưu điểm: ít tham số hơn, huấn luyện nhanh hơn, dễ áp dụng cho các ứng dụng yêu cầu nhẹ.

Vai trò trong dự án

- Trong đề tài này, mỗi frame ảnh khi qua CNN (MobileNetV2) cho một **vector đặc trưng**.
- Những vector này được tập hợp thành **chuỗi thời gian** (ví dụ 30 frame, tương ứng 30 vector).
- GRU xử lý chuỗi này, học mối quan hệ giữa các bước thời gian (sự tiến triển động tác) → từ đó dự đoán cờ chỉ.
- Output từ GRU được đưa vào lớp Dense + Softmax để phân lớp sang một trong 5 cờ chỉ.

2.4 MediaPipe Hands

- **MediaPipe Hands** là framework do Google cung cấp để phát hiện bàn tay trong ảnh/video và xác định **21 điểm khớp ngón tay** (landmarks).
- **Lợi ích:**

- Cho phép **crop vùng bàn tay (ROI)** chuẩn xác, loại bỏ phần nền không cần thiết, giúp mô hình CNN tập vào vùng quan trọng.
- Giảm nhiễu, giảm khối lượng xử lý (không dùng toàn ảnh)
- Cải thiện độ chính xác trong điều kiện nền phức tạp, nhiều đối tượng khác, ánh sáng kém.
- Trong dự án, có tùy chọn `--use-mediapipe` khi chạy game hoặc thu dữ liệu để dùng MediaPipe trước khi truyền ảnh vào mô hình. (nếu không dùng, dùng ảnh toàn khung).

Quy trình (pipeline) từ thu thập dữ liệu đến game

Dưới đây là pipeline thực tế trong dự án:

1. Thu thập dữ liệu (`capture_gesture_dataset.py`)

- Người dùng thực hiện cử chỉ theo nhãn tương ứng (1–5)
- Script đếm ngược và ghi **30 frame** liên tục cho mỗi lần thực hiện
- Mỗi sample (folder) chứa 30 ảnh `frame_XXX.png`
- Dữ liệu được tổ chức theo cấu trúc:
`archive/<split>/<split>/<sample>/...`
 - Có file manifest (CSV) mapping `sample_folder` → `label_slug` → `label_index`
 - Khuyến nghị thu trong nhiều điều kiện ánh sáng, nhiều người, thu trong điều kiện game thật (có HUD) để mô hình quen với môi trường thực.

2. Tiền xử lý dữ liệu

- (Tùy chọn) dùng MediaPipe để detect bàn tay và crop ROI
- Resize ảnh về kích thước chuẩn (ví dụ 224×224)
- Normalize pixel (chia 255, trừ mean, ...)
- Có thể augment dữ liệu (xoay, thay đổi độ sáng...) để tăng tính đa dạng
- Tạo batch, shuffle, padding / cắt chuỗi nếu cần để đảm bảo chuỗi đầu vào có độ dài đồng nhất

3. Huấn luyện mô hình (notebook gesture_recognition_minimal.ipynb)

- Sử dụng MobileNetV2 + GRU
- Batch size = 12, số epoch ≈ 35
- Fine-tune hoặc train từ pretrained weights
- Theo dõi loss, accuracy trên tập validation
- Lưu mô hình .h5 (best_model) và dùng mô hình đã train sẵn trong game

4. Chạy game / inference thời gian thực (run_game_5class.py)

- Mở camera / webcam, liên tục lấy frame
- (Nếu dùng) MediaPipe detect & crop vùng bàn tay
- Resize / normalize, tạo window chuỗi (sliding window) nếu cần
- Dùng mô hình (MobileNetV2 + GRU) dự đoán xác suất các cử chỉ
- Áp threshold / smoothing / lọc để chọn lệnh ổn định
- Gửi lệnh điều khiển xe trong game (tăng, giảm, rẽ, phanh)
- HUD hiển thị xác suất top classes, camera feed, sprite đường đua, xe
- Có fallback: nếu không phát hiện tay hoặc xác suất không rõ, giữ lệnh cũ hoặc không thay đổi
 - Có hỗ trợ điều khiển dự phòng bằng bàn phím (WASD, phím mũi tên)
 - Có phím tắt / bật camera feed, thoát game (ESC), ...

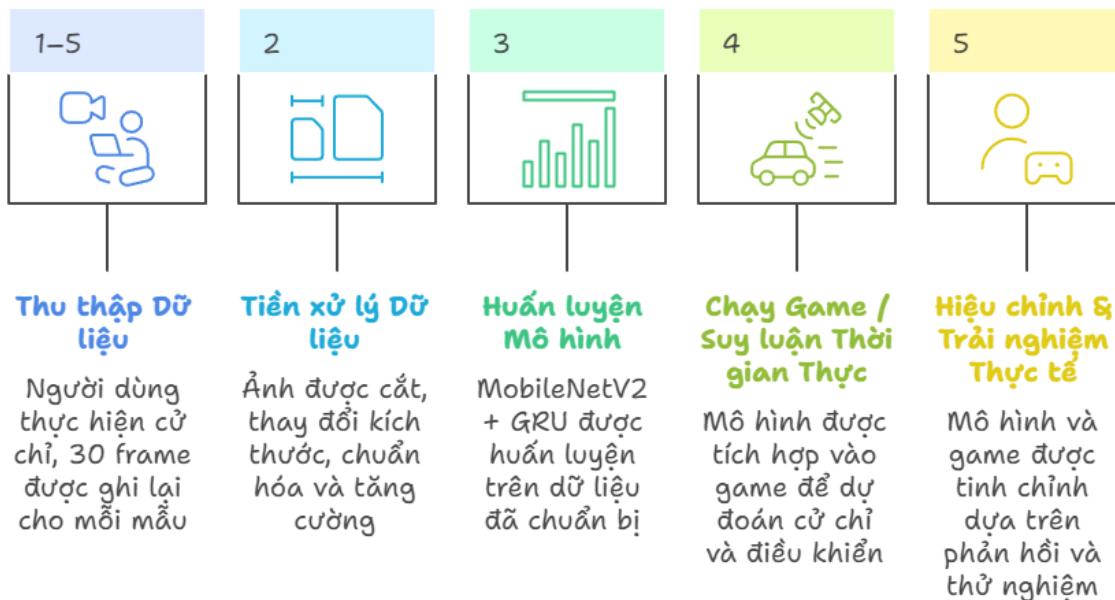
5. Hiệu chỉnh, debug & trải nghiệm thực tế

- Nếu mô hình nhầm (ví dụ rẽ trái / phải bị nhầm) \rightarrow thu thêm dữ liệu, điều chỉnh smoothing
- Nếu game bị lag \rightarrow giảm resolution camera / HUD, tắt camera feed, dùng GPU
- Kiểm tra mô hình load đúng phiên bản, kiểm tra version TensorFlow tương thích

2.5 Thư viện hỗ trợ

- OpenCV: xử lý video/webcam.
- Pygame: xây dựng game 2D.
- TensorFlow/Keras: huấn luyện mô hình.

Quy trình Phát triển Game Nhận Diện Cử Chỉ



CHƯƠNG 3: PHÂN TÍCH HỆ THỐNG

3.1 Mục tiêu & Tóm tắt chức năng hệ thống

Mục tiêu chung: Xây dựng hệ thống nhận diện cử chỉ tay thời gian thực từ webcam và sử dụng đầu ra để điều khiển một trò chơi đua xe 2D. Hệ thống gồm pipeline thu thập dữ liệu → tiền xử lý → huấn luyện mô hình → inference thời gian thực → điều khiển game.

Chức năng chính:

1. Thu thập dữ liệu cử chỉ bằng webcam (script `capture_gesture_dataset.py`).
2. Quản lý manifest/metadata cho mỗi chuỗi (sequence) dữ liệu.
3. Tiền xử lý: phát hiện bàn tay (MediaPipe), trích ROI, resize, chuẩn hoá.
4. Huấn luyện mô hình: MobileNetV2 (backbone) + GRU (xử lý chuỗi).
(notebook: `gesture_recognition_minimal.ipynb`, `hand-gesture-recognition-system.ipynb`)
5. Triển khai và nạp model (.h5) trong runtime để thực hiện phân loại 5 cử chỉ.
6. Ánh xạ kết quả phân loại sang điều khiển game (script `run_game_5class.py`).

7. Hiển thị HUD (xác suất top-k, FPS, trạng thái MediaPipe) để debug và giám sát.
8. Lưu model/ckpt và lịch sử huấn luyện (thư mục `gesture_model_20250924_102037/`, ...).

3.2. Yêu cầu chức năng (Functional requirements) — chi tiết & tiêu chí chấp nhận

1. Nhận diện cử chỉ

- Nhận dạng ít nhất 5 cử chỉ: Thumbs Up, Thumbs Down, Left Swipe, Right Swipe, Stop.
 - Tiêu chí chấp nhận: accuracy (validation) $\geq 90\%$ (mục tiêu dự án $\sim 97.7\%$ như thực nghiệm hiện tại).

```
==== TRAINING RESULTS ====
Best Training Accuracy: 1.0000 (100.00%)
Best Validation Accuracy: 0.9773 (97.73%)
Model saved to: gesture_model_20250927_005401
```

2. Thời gian thực

- Hệ thống phải phản hồi điều khiển đủ nhanh để game cảm thấy mượt.
- Mục tiêu: FPS game ≥ 20 ; latency phản hồi cử chỉ ≤ 100 ms (khi dùng GPU) — nếu chạy trên CPU có thể chấp nhận latency cao hơn, cần có chế độ degrade.

3. Ổn định trong môi trường thật

- Hoạt động tốt khi có biến đổi ánh sáng, phông nền khác nhau; tùy chọn bật/tắt MediaPipe để tăng độ ổn định.
- Tiêu chí: tỷ lệ thất bại (không phát hiện bàn tay) $\leq 5\%$ trong các điều kiện thử nghiệm chuẩn.

4. Ổn định trong môi trường thật

- Công cụ thu thập hỗ trợ đếm ngược, lưu tên theo chuẩn REC_YYYYMMDD_HH_MM_SS_<label>_<context>, cập nhật manifest tự động.

5. Tích hợp và triển khai mô hình

- Mô hình huấn luyện lưu ở .h5 và được load bởi game khi khởi chạy.
- Tùy chọn: sử dụng MediaPipe để tiền xử lý trước khi inference.

6. Giao diện HUD & Debug

- Hiển thị xác suất top-3, trạng thái MediaPipe, FPS, buffer length.
- Để bật/tắt (cờ --use-mediapipe).

7. Quản lý mô hình & checkpoint

- Lưu best_model.h5, final_gesture_model.h5, training_history.pkl.
- Cho phép thay model mà không sửa code game.

8. Bảo mật & quyền riêng tư

- Dữ liệu camera và dataset lưu cục bộ (khuyến nghị không upload công khai nếu chứa người thật).

3.3. Yêu cầu phi chức năng (Non-functional requirements)

1. Hiệu năng & Latency

- Inference cho chuỗi 30 frame (một dự đoán) phải trong giới hạn chấp nhận được để giữ FPS.
- Tối ưu: dùng batch size inference = 1, tận dụng GPU nếu có.

2. Tính mở rộng & bảo trì

- Để thêm cử chỉ mới: quy trình collect → append manifest → train → deploy.
- Cấu hình (config file) cho các tham số (sequence length, input_size, threshold, smoothing window).

3. Tính di động

- Chạy trên Windows 11; có thể port sang Linux; hỗ trợ CPU/GPU.
- Hướng tương lai: chuyển sang TFLite/ONNX để chạy trên mobile.

4. Độ tin cậy & ổn định

- Xử lý trường hợp mất kết nối camera, lỗi load model, thiếu quyền truy cập; hệ thống không crash.

5. Khả năng mở rộng dữ liệu

- Hệ thống quản lý manifest phải hỗ trợ append, remove, merge datasets mà không bị corrupt (đã lưu ý sử dụng dấu ;).

6. Bảo mật & quyền riêng tư

- Lưu trữ dataset cục bộ, mã hóa metadata nếu cần, cung cấp hướng dẫn xoá dữ liệu.

3.4. Kiến trúc tổng thể (Component overview)

Thành phần chính (high-level):

1. **Capture Module:** đọc frame từ webcam; có chế độ ghi dữ liệu (capture_gesture_dataset.py) và chế độ inference (run_game_5class.py).
2. **MediaPipe Hands (tuỳ chọn):** phát hiện landmarks, trả về bounding box/ROI hoặc grip points; giúp giảm nhiễu.
3. **Preprocessing:** crop ROI, resize (thường 224×224), chuẩn hoá pixel (scale 0–1 hoặc [-1,1]).
4. **Frame Buffer:** lưu chuỗi N frame (ví dụ 30) làm input một “sample” cho mô hình chuỗi.
5. **Feature Extractor:** MobileNetV2 (FC trước GRU) trích vector đặc trưng cho từng frame.
6. **Sequence Model:** GRU (ví dụ 128 units) nhận chuỗi feature vectors → output class probabilities.
7. **Postprocessing:** smoothing (moving average / majority vote), thresholding (confidence $> t$), debounce để tránh chói lệnh nháy.
8. **Controller Mapper:** ánh xạ label → hành động trong game (left, right, accel, brake).
9. **Game Runner:** xử lý vật lý, cập nhật giao diện, HUD.

3.5 Mô tả chi tiết các module

A. Capture Module (capture_gesture_dataset.py / runtime capture)

- **Chức năng:**
 - Ghi chuỗi frames khi người nhấn phím tương ứng label (1..5).
 - Tên file theo chuẩn REC YYYYMMDD HH MM SS <label> <context>.
 - Cập nhật manifest CSV (new_train.csv) bằng dấu ;.
- **Input:** webcam frames, phím bấm.
- **Output:** thư mục chứa chuỗi frames (hoặc video), record entry vào manifest CSV.
- **Lưu ý:** đảm bảo atomic write vào CSV (tránh ký tự rác).

B. Preprocessing Module

- **Chức năng:**
 - Nếu bật MediaPipe: lấy bounding box từ landmarks, mở rộng margin (ví dụ 0.4), crop ROI.
 - Resize → 224×224, normalize.
 - (Optional) augmentation nhẹ (brightness, flip, small rotation) dùng cho training only.
- **Input:** raw frame hoặc landmarks.
- **Output:** preprocessed frame hoặc numpy array.

C. Feature Extractor (MobileNetV2 backbone)

- **Chức năng:** trích vector đặc trưng cho từng frame.
- **Kiến trúc:** MobileNetV2 pretrained (imagenet) → loại bỏ phần top → output feature vector (ví dụ 1280 dims).
- **Output:** sequence of feature vectors shape (SEQ_LEN, feature_dim).

D. Sequence Model (GRU)

- **Chức năng:** nhận sequence features → học temporal dependencies → trả về logits → softmax → probabilities.
- **Cấu hình đề xuất:** GRU 1–2 layer, 128 units, dropout 0.2, dense + softmax (5 classes).

E. Postprocessing & Smoothing

- **Chức năng:** giảm jitter bằng:
 - sliding-window majority vote, hoặc
 - exponential moving average on probabilities, hoặc
 - simple debounce (yêu cầu same label xuất hiện K lần liên tiếp).
- **Parameters:** window_size (ví dụ 5), confidence_threshold (ví dụ 0.6).

F. Controller Mapper

- **Chức năng:** ánh xạ label sang hành động game; ví dụ:
 - thumbs_up → tăng tốc
 - thumbs_down → giảm tốc
 - leftSwipe → rê trái
 - rightSwipe → rê phải
 - stop → phanh

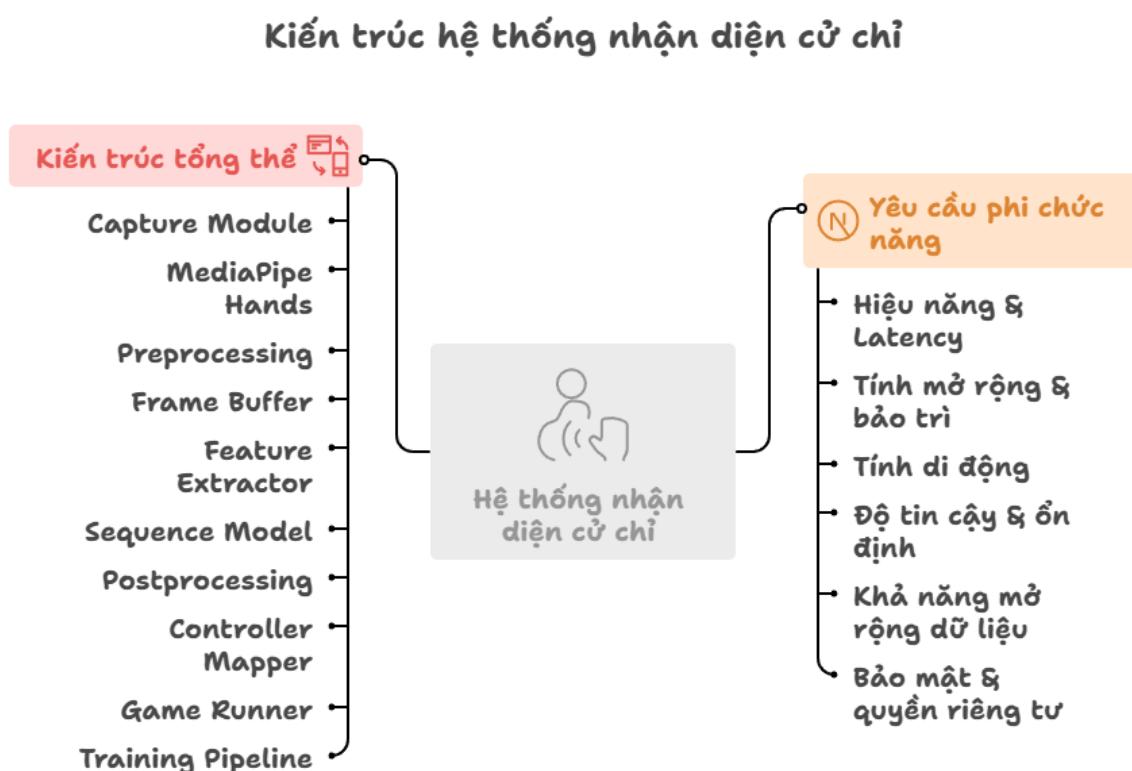
G. Game Runner (run_game_5class.py)

- **Chức năng:** nhận lệnh điều khiển từ Controller Mapper → cập nhật state game (vị trí, tốc độ) → render frame → hiển thị HUD.
- **HUD:** hiện top-k probabilities, FPS, trạng thái MediaPipe, biểu đồ nhỏ.

H. Training Pipeline (notebooks & model_colab)

- **Chức năng:** pipeline training end-to-end: load manifest, dataset generator (sequence loader), augmentation, compile model, train, checkpoint best model, save best_model.h5, final_gesture_model.h5, training_history.pkl.

3.6. Định dạng dữ liệu & manifest



Manifest CSV (dùng dấu ;) — mỗi dòng đại diện cho một sequence (chuỗi 30 frames):

sequence_path;label;context:frame_count;timestamp

REC_20250924_102037_thumbs_up_room1;thumbs_up;room1_hud
;30;2025-09-24T10:20:37

Chuỗi frames: lưu dưới dạng thư mục chứa ảnh frame_000.jpg ... frame_029.jpg hoặc file video .mp4 + index. Khi huấn luyện, generator phải đọc theo manifest.

Model artifact:

- best_model.h5 (được nạp bởi game)
- final_gesture_model.h5
- training_history.pkl (loss/metrics)

3.7 Luồng dữ liệu & quy trình (end-to-end)

1. **Thu thập:** capture_gesture_dataset.py → save sequences → append manifest.
2. **Tiền xử lý offline:** kiểm tra manifest, clean corrupted files, balance classes.
3. **Huấn luyện:** chạy notebook, fine-tune MobileNetV2 + GRU, lưu checkpoints.
4. **Đánh giá:** confusion matrix, accuracy, per-class recall/precision.
5. **Triển khai:** copy best_model.h5 vào gesture_racing_game/gesture_model_xxx/.
6. **Runtime:** run_game_5class.py load model → inference → control mapping.

3.8 Xử lý ngoại lệ & lỗi

1. Không mở được camera

- Hiển thị thông báo lỗi, fallback: tắt feature camera mode; không crash.

2. Không phát hiện bàn tay

- HUD hiển thị “No hand detected”; không gửi lệnh (hoặc gửi lệnh “stop” an toàn).
 - Nếu liên tục nhiều frame, hiển thị hint cho người dùng (điều chỉnh vị trí/ánh sáng).

3. Model load fail

- Kiểm tra file tồn tại; nếu lỗi, load model fallback (cũ hơn) hoặc tắt tính năng cử chỉ.

4. Dữ liệu manifest hỏng

- Khi append, validate mỗi dòng; ghi log lỗi và skip.

5. FPS giảm mạnh

- Tự động giảm chế độ (tắt MediaPipe, giảm input_size, giảm smoothing window).

3.9 Kiểm thử & Đánh giá (Test plan)

1. Unit tests

- Test cho preprocessor (crop, resize, normalize).
- Test parser manifest.
- Test mapper label→action.

2. Integration tests

- Từ webcam → preprocessor → model → game, test end-to-end với sample inputs (có thể dùng recorded videos).

3. Performance tests

- Đo latency: capture→inference→action.
- Đo FPS khi bật/tắt MediaPipe, chạy trên CPU vs GPU.

4. Model evaluation

- Train/val/test split; báo cáo: accuracy, precision, recall, F1 cho mỗi lớp.
- Confusion matrix để tìm cử chỉ dễ nhầm lẫn.

5. User acceptance testing (UAT)

- Người dùng thực hiện kịch bản: lái 1 vòng/5 phút, ghi lỗi, trải nghiệm.
- Bộ tiêu chí: cảm giác mượt, false positive/negative rate, dễ học cử chỉ.

3.10 Yêu cầu phần cứng & phần mềm

Phần mềm

- Python 3.10
- TensorFlow 2.13
- OpenCV (cv2)
- MediaPipe
- Pygame
- NumPy, Pandas, scikit-learn (cho evaluation)
- Jupyter Notebook (cho huấn luyện & thử nghiệm)

Phần cứng (khuyến nghị)

- Phát triển/huấn luyện: GPU (NVIDIA RTX 3050 Ti hoặc tương đương) + CUDA/cuDNN.
- Chạy inference & game: PC có CPU tầm trung và GPU rời khuyến nghị, nếu không có GPU cần tối ưu model/tải nhẹ.

3.11 Bảo mật & Quyền riêng tư

- Xử lý local: khuyến nghị toàn bộ pipeline camera & dữ liệu thực hiện cục bộ.
- Quyền truy cập camera: hướng dẫn người dùng cấp quyền; cung cấp chức năng tắt ghi (no-save) khi chỉ chơi.
- Quản lý dataset: cung cấp script xóa dữ liệu nhạy cảm; khuyến nghị không đăng ảnh người thật lên public repo.

3.12 Khả năng mở rộng & Bảo trì

1. Thêm cử chỉ mới

- Quy trình: thu dữ liệu mới → cập nhật manifest → retrain → validate → deploy.

2. Multi-modal fusion

- Kết hợp âm thanh/IMU/khớp xương (landmarks) để tăng độ chính xác.

3. Deployment trên mobile/edge

- Chuyển sang TensorFlow Lite; prune/quantize model.

4. Smoothing & Ensemble

- Thêm ensemble model để cải thiện robust; thêm smoothing để giảm jitter.

5. Cấu hình

- Dùng config.yaml để lưu params: SEQ_LEN, INPUT_SIZE, CONF_THRESHOLD, USE_MEDIPIPE, MODEL_PATH.

3.13 Rủi ro & Biện pháp giảm thiểu

- Rủi ro: Overfitting do dữ liệu ít / không đa dạng → giải pháp: augmentation, thu thêm dữ liệu trong nhiều bối cảnh.
- Rủi ro: Latency cao trên CPU → giải pháp: tối ưu model, giảm input_size, dùng TFLite.
- Rủi ro: Nhận diện sai trong ánh sáng yếu → giải pháp: thêm augmentation về sáng/contrast; recommend user dùng ánh sáng tốt.

3.14. Kiến trúc tổng thể

Hệ thống được xây dựng theo pipeline từ **Input** → **Processing** → **Output**, cụ thể:

1. Input (Dữ liệu đầu vào)

- Nguồn dữ liệu: Webcam (30 FPS, độ phân giải 640×480 hoặc 1280×720).
- MediaPipe Hands thực hiện:
 - Phát hiện vị trí bàn tay trong khung hình.
 - Cắt (crop) vùng quan tâm (ROI) để loại bỏ background thừa.

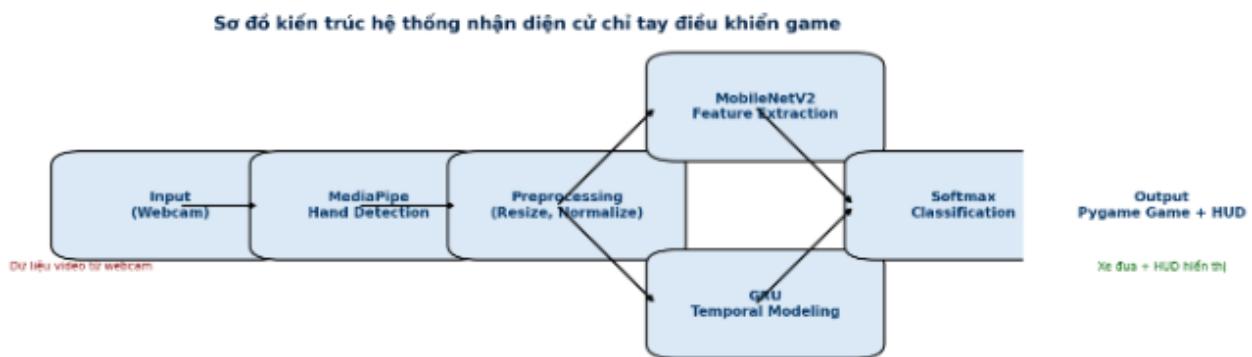
2. Processing (Xử lý trung gian)

- **Tiền xử lý ảnh:**
 - Resize ảnh về 224×224 pixel.
 - Chuẩn hóa giá trị pixel $[0,1]$.
 - (Tùy chọn) Data augmentation như xoay, thay đổi sáng.
- **Trích xuất đặc trưng không gian (Spatial Feature Extraction):**
 - Dùng **MobileNetV2** (pretrained ImageNet, fine-tune cho dataset cụ thể).
 - Kết quả: mỗi frame được biểu diễn thành vector đặc trưng (feature vector ~ 1280 chiều).
- **Xử lý chuỗi thời gian (Temporal Modeling):**
 - Ghép các vector đặc trưng từ nhiều frame liên tiếp (ví dụ 30 frame).
 - Đưa vào **GRU (Gated Recurrent Unit)** để học sự thay đổi động tác theo thời gian.
- **Phân loại (Classification):**
 - Lớp Dense + Softmax để dự đoán xác suất cho 5 lớp cử chỉ.

- Chọn lớp có xác suất cao nhất làm nhãn dự đoán.
- (Tùy chọn) áp dụng smoothing hoặc threshold để ổn định kết quả.

3. Output (Đầu ra)

- **Game Engine (Pygame):**
 - Nhận tín hiệu điều khiển từ mô hình.
 - Thực hiện hành động tương ứng: tăng tốc, giảm tốc, rẽ trái, rẽ phải, phanh.
 - Render môi trường đường đua và sprite xe.
- **HUD (Head-Up Display):**
 - Hiển thị camera feed.
 - Biểu đồ xác suất top-5 dự đoán.
 - Thông tin game: tốc độ xe, số điểm, trạng thái.



CHƯƠNG 4: DỮ LIỆU VÀ TIỀN XỬ LÝ

4.1. Pipeline xử lý dữ liệu

Để đảm bảo mô hình nhận diện cử chỉ tay hoạt động ổn định, hệ thống cần có quy trình xử lý dữ liệu nhất quán. Pipeline dữ liệu trong dự án được thiết kế gồm các bước sau:

1. **Thu thập dữ liệu ảnh từ webcam:** Hệ thống ghi lại video trực tiếp, sau đó trích xuất thành các frame đơn lẻ.
2. **Phát hiện và trích ROI (Region of Interest) bàn tay:** Sử dụng **MediaPipe Hands** để xác định vị trí bàn tay và khớp ngón tay, loại bỏ các vùng nền dư thừa.
3. **Tiền xử lý ảnh:**
 - o Chuẩn hóa kích thước ảnh về **224 × 224 pixels**, 3 kênh màu RGB.
 - o Chuẩn hóa giá trị pixel về khoảng **[0,1]** để tăng hiệu quả huấn luyện.
4. **Xây dựng chuỗi dữ liệu theo thời gian:** Mỗi cử chỉ tay được biểu diễn bởi **30 frame liên tiếp**. Điều này cho phép mô hình học được sự thay đổi theo thời gian thay vì chỉ một ảnh tĩnh.
5. **Lưu trữ dữ liệu:** Các chuỗi frame và nhãn được lưu thành dataset huấn luyện (train/test/validation) để phục vụ cho quá trình học sâu.

4.2. Tổng quan kiến trúc và luồng dữ liệu

Hệ thống gồm hai luồng chính:

1. **Offline pipeline (thu thập → tiền xử lý → huấn luyện → đánh giá → lưu model)**
2. **Realtime pipeline (webcam → preprocess → model inference → postprocess → ánh xạ lệnh → game)**

4.3. Thành phần (module) chi tiết

4.3.1 Capture Module (`capture_gesture_dataset.py`)

- Chức năng: thu các chuỗi frame cho từng label (mỗi sequence = 30 frame). Hỗ trợ đếm ngược, context tag, cập nhật manifest CSV tự động.
- Input: **webcam frames, phím chọn label.**

- Output: **thư mục sequence (frame_000.jpg ... frame_029.jpg) và entry mới trong manifest.csv.**
- Tên file/thu mục:
REC_YYYYMMDD_HH_MM_SS_<label>_<context>.
- Lưu ý: khi append manifest cần write atomically (ví dụ ghi vào temp rồi rename) để tránh corrupt.

4.3.2 Hand Preprocessor (MediaPipe – tuỳ chọn)

- **Chức năng:** phát hiện landmarks, tạo bounding box (với margin, ví dụ 0.4), crop ROI bàn tay, trả về ảnh crop. Nếu MediaPipe không có (hoặc tắt), fallback crop trung tâm hoặc toàn khung.
- **Tham số:**
 - enabled: bool
 - margin: float (0.2–0.5)
- **Output:** cropped frame (color BGR hoặc RGB tùy pipeline), trả về ảnh đã crop.

4.3.3 Preprocessing

- **Các bước:**
 1. Crop ROI (từ hand preprocessor).
 2. Resize về **224×224**.
 3. Chuyển dtype → float32.
 4. Áp tf.keras.applications.mobilenet_v2.preprocess_input (scale về [-1,1]).
 5. (only training) augmentation: brightness, contrast, random flip, small rotation (-10..10°), gaussian noise.
- **Tại runtime:** chỉ dùng resize + preprocess_input (không augmentation).

4.3.4 Frame Buffer

- **Ý tưởng:** Lưu SEQ_LEN = 30 frames đã tiền xử lý.
- Được khai báo trong class GestureRecognizer5:

```
self.frame_buffer: deque[np.ndarray] = deque(maxlen=30)
```

- **Cách hoạt động:** mỗi frame preprocessed append vào buffer; khi đầy 30 frame → trigger `_predict()`.

4.3.5 Feature extractor (MobileNetV2 backbone)

- **Vai trò:** trích vector đặc trưng cho mỗi frame.
- **Thiết lập:**
 - Pretrained weights: imagenet.
 - Loại bỏ top (`include_top=False`), global average pooling → `feature_dim` (ví dụ 1280).
 - Fine-tune: mở khóa 24 lớp cuối (như bạn đã thực nghiệm).
- **Output:** `feature_vector shape (feature_dim,)` cho mỗi frame.

4.3.6 Sequence model (GRU)

- **Kiến trúc mấu:**
 - Input shape: (`SEQ_LEN, feature_dim`)
 - GRU layers: 1 layer GRU, 128 units, dropout 0.2
 - Dense -> Softmax (5 classes)
- **Loss / optimizer:**
 - Loss: `categorical_crossentropy`
 - Optimizer: Adam lr=1e-4 (fine-tune: 1e-5–1e-4)
- **Output:** logits → softmax probabilities.

4.3.7 Postprocessing & Smoothing

- **Mục đích:** giảm jitter — tránh lệnh nhấp nháy khi model dự đoán dao động.
- **Kỹ thuật:**
 - Exponential Moving Average (EMA) trên xác suất:

$$\text{prob_smoothed} = \alpha * \text{prob_prev} + (1-\alpha) * \text{prob_now}$$
 - Hoặc majority-vote trên sliding window (window size = 5)

- Debounce: chỉ gửi lệnh khi label ổn định K lần liên tiếp (K=2..3) và confidence > CONF_THRESHOLD (vd. 0.30).
- **Cài đặt mẫu:**
 - CONF_THRESHOLD = 0.30
 - SMOOTH_ALPHA = 0.6
 - DEBOUNCE_COUNT = 2

4.3.8 Controller Mapper

- **Chi tiết mapping:**
 - 0 -> thumbs_up -> accel (tăng tốc)
 - 1 -> thumbs_down -> decel
 - 2 -> leftSwipe -> turn_left
 - 3 -> rightSwipe -> turn_right
 - 4 -> stop -> brake
 - **Format truyền vào game:** event hoặc call function game API (ví dụ game.apply_action(action)).
-

4.3.9 Game Runner (Pygame)

- **Chức năng:** xử lý loop game, vật lý (speed, x-position), render, HUD.
- **Phương án tích hợp:**
 - **Option A (đồng bộ):** inference gọi trực tiếp game.apply_action() trong cùng thread — đơn giản nhưng có thể gây drop FPS.
 - **Option B (khuyên dùng):** tách thành 2 thread/process:
 - Thread A: camera + preprocessing + inference (producer) → đẩy lệnh vào queue.Queue.
 - Thread B (main thread): Pygame loop (consumer) đọc lệnh từ queue không chặn (use queue.get_nowait()), áp dụng action.

- Lý do: Pygame yêu cầu render chỉ trong main thread trên nhiều nền tảng; inference có thể block, nên tách thread giúp giữ FPS.

4.4. Thiết kế game

Để thử nghiệm mô hình trong môi trường ứng dụng thực tế, nhóm xây dựng một game đua xe 2D bằng thư viện Pygame:

- **Giao diện:**
 - Đường đua được thiết kế đơn giản với các chướng ngại vật ngẫu nhiên.
 - Xe của người chơi có thể di chuyển theo 5 hành động cơ bản.
- **Cơ chế điều khiển:**
 - **Tiến** → xe tăng tốc.
 - **Lùi** → xe giảm tốc.
 - **Rẽ trái / rẽ phải** → xe di chuyển sang hai bên.
 - **Phanh** → xe dừng lại hoặc giảm tốc đột ngột.
- **HUD (Head-Up Display):**
 - Hiển thị **xác suất dự đoán top-2** của mô hình ngay trên màn hình.
 - Giúp người chơi biết được mô hình đang nhận diện cử chỉ nào và mức độ tin cậy ra sao.

CHƯƠNG 5: CÀI ĐẶT VÀ THỰC NGHIỆM

5.1 Công cụ và môi trường

Hệ thống được xây dựng và triển khai trên nền tảng sau:

- **Ngôn ngữ lập trình:** Python 3.10
- **Thư viện chính:**
 - TensorFlow/Keras 2.13 → huấn luyện mô hình CNN–GRU.
 - OpenCV → xử lý ảnh và video từ webcam.
 - MediaPipe Hands → phát hiện bàn tay và khớp ngón tay.
 - Pygame → xây dựng game đua xe 2D.
- **Phần cứng thử nghiệm:**
 - Hệ điều hành: Windows 11
 - GPU: NVIDIA RTX 3050 Ti
 - RAM: 16 GB

5.2 Quy trình thực hiện

Quy trình triển khai được chia thành 5 bước chính:

1. **Xây dựng công cụ thu thập dữ liệu**
 - File `capture_gesture_dataset.py` được viết để thu thập video từ webcam.
 - Người dùng thực hiện 5 cử chỉ tay khác nhau, mỗi cử chỉ được ghi lại thành **30 frame** liên tiếp.
2. **Làm sạch và tổ chức dữ liệu**
 - Dữ liệu được lọc bỏ các frame mờ, sai ROI.
 - Tạo **CSV manifest** chứa đường dẫn ảnh và nhãn cử chỉ.
3. **Huấn luyện mô hình**
 - Mô hình **MobileNetV2 + GRU** được huấn luyện trên tập dữ liệu đã tiền xử lý.
 - Sử dụng **augmentation nhẹ** (xoay $\pm 10^\circ$, thay đổi sáng, dịch chuyển nhỏ) để tăng khả năng khái quát.
 - Tách dữ liệu thành **train/validation/test** theo tỉ lệ 70/20/10.
4. **Triển khai mô hình**
 - Sau khi huấn luyện, mô hình được lưu dưới dạng **.h5**.
 - Copy mô hình vào thư mục game, tích hợp vào vòng lặp xử lý Pygame.
5. **Chạy game thử nghiệm**

- Người dùng khởi động game, webcam nhận diện cử chỉ tay và điều khiển xe đua theo thời gian thực.

5.3 Kết quả thực nghiệm

- **Độ chính xác (Validation accuracy): ~97.7%**

→ Mô hình học tốt, phân biệt rõ các cử chỉ tay.

- **Hiệu năng game:**

- Game chạy ổn định, FPS > 20.
- Độ trễ giữa cử chỉ và hành động gần như không đáng kể.
- **MediaPipe Hands** giúp loại bỏ nền, giữ ROI ổn định → cải thiện đáng kể chất lượng dữ liệu.
- Hệ thống có thể mở rộng thêm cử chỉ mới bằng cách thu thêm dữ liệu và huấn luyện lại.

```
mermaid
```

```
flowchart TD
    A[Webcam] --> B[Thu thập dữ liệu - capture_gesture_dataset.py]
    B --> C[Làm sạch & CSV manifest]
    C --> D[Huấn luyện MobileNetV2 + GRU]
    D --> E[.h5 mô hình đã huấn luyện]
    E --> F[Tích hợp vào game Pygame]
    F --> G[Chạy game & kiểm thử]
```

CHƯƠNG 6: ĐÁNH GIÁ & BÀI HỌC KINH NGHIỆM

6.1. Đánh giá ưu điểm

Sau khi triển khai và kiểm thử hệ thống, có thể rút ra những ưu điểm nổi bật như sau:

- **Mô hình nhẹ, chạy thời gian thực:** Việc sử dụng MobileNetV2 kết hợp GRU giúp hệ thống vừa đảm bảo độ chính xác cao, vừa duy trì tốc độ xử lý ổn định (>20 FPS).
- **Tích hợp HUD trong game:** Giao diện hiển thị xác suất dự đoán (top-2) giúp người dùng dễ dàng theo dõi, đồng thời hỗ trợ quá trình debug và cải thiện trải nghiệm chơi.
- **Công cụ thu thập dữ liệu hiệu quả:** Việc xây dựng script `capture_gesture_dataset.py` cho phép nhanh chóng thu thập và quản lý dữ liệu huấn luyện theo định dạng chuẩn.

6.2. Hạn chế

Bên cạnh những điểm mạnh, hệ thống vẫn còn một số hạn chế:

- **Phạm vi nhận diện còn hạn chế:** Mới chỉ nhận diện được 5 cử chỉ cơ bản (tiến, lùi, rẽ trái, rẽ phải, phanh).
- **Phụ thuộc điều kiện ánh sáng:** Mô hình dễ bị ảnh hưởng bởi môi trường ánh sáng yếu hoặc thay đổi mạnh.
- **Chưa có cơ chế smoothing:** Dự đoán hiện tại dựa trên từng frame riêng lẻ, dễ gây hiện tượng dao động (jitter) khi tay di chuyển nhanh.

6.3. Bài học kinh nghiệm

Trong quá trình thực hiện, nhóm rút ra được một số kinh nghiệm quan trọng:

- **Quản lý dữ liệu:** Việc xây dựng và duy trì **CSV manifest** đóng vai trò cốt lõi trong huấn luyện, giúp đảm bảo tính đồng bộ và dễ dàng mở rộng tập dữ liệu.
- **Chất lượng dữ liệu thực tế ảnh hưởng trực tiếp đến hiệu quả mô hình:** Dữ liệu thu thập trong môi trường gần giống với bối cảnh sử dụng thật giúp mô hình khái quát tốt hơn.

- **Cân bằng giữa độ chính xác và tốc độ xử lý:** Với ứng dụng thời gian thực như game, không chỉ accuracy mà cả tốc độ dự đoán (latency) và FPS đều cần được ưu tiên.

CHƯƠNG 7: HƯỚNG PHÁT TRIỂN

7.1 Mở rộng tập cử chỉ

- **Ý tưởng:** Thêm các cử chỉ nâng cao như:
 - Giữ lái (Keep steering).
 - Bật nitro / tăng tốc đặc biệt.
 - Đổi góc nhìn camera.
 - Pause / Resume game.
- **Lợi ích:** Tăng tính đa dạng và hấp dẫn cho game.
- **Thách thức:**
 - Cần thu thập thêm nhiều dữ liệu mẫu cho mỗi cử chỉ.
 - Dữ liệu phải cân bằng giữa các lớp để tránh mất cân đối khi huấn luyện.

7.2 Smoothing và Ensemble

- **Ý tưởng:**
 - Thay vì dựa trên dự đoán từng frame riêng lẻ, áp dụng **trung bình trượt (moving average)** hoặc **bộ lọc Kalman** để làm mượt dự đoán.
 - Dùng **ensemble nhiều mô hình** (ví dụ MobileNetV2 + EfficientNet) để tăng độ ổn định.
- **Lợi ích:**
 - Giảm hiện tượng dự đoán dao động (jitter).
 - Đảm bảo hành động trong game mượt hơn.
- **Thách thức:**
 - Tăng độ trễ (latency).
 - Cần tối ưu để giữ FPS ổn định.

7.3 Tối ưu giao diện và trải nghiệm người dùng (UI/UX)

- **Ý tưởng:**
 - Thêm hiệu ứng âm thanh khi xe tăng tốc/phanh.
 - Thêm bảng điểm, nhiều màn chơi, vật cản mới.

- Thiết kế HUD trực quan hơn (ví dụ vòng tròn hiển thị % xác suất từng cử chỉ).
- **Lợi ích:** Game hấp dẫn hơn, dễ tiếp cận với người dùng phổ thông.
- **Thách thức:** Cần cân bằng giữa hiệu ứng và hiệu năng, tránh làm giảm FPS.

7.4 Hỗ trợ nhiều người chơi (Multiplayer)

- **Ý tưởng:**
 - Thêm chế độ 2 người chơi → mỗi người điều khiển bằng webcam riêng.
 - Tích hợp online multiplayer (qua socket/P2P).
- **Lợi ích:** Tăng tính cạnh tranh và giải trí.
- **Thách thức:**
 - Đồng bộ dữ liệu webcam giữa nhiều client.
 - Yêu cầu băng thông mạng và tối ưu hóa truyền dữ liệu.

7.5 Tích hợp với thiết bị AR/VR

- **Ý tưởng:**
 - Kết hợp kính VR hoặc AR để tạo cảm giác lái xe thực tế.
 - Tay điều khiển trực tiếp trong môi trường ảo.
- **Lợi ích:** Đem lại trải nghiệm “immersive” (đắm chìm).
- **Thách thức:**
 - Yêu cầu phần cứng mạnh hơn.
 - Cần tối ưu tốc độ nhận diện để không gây **motion sickness** (chóng mặt, buồn nôn).

7.6 Tối ưu mô hình cho thiết bị di động

- **Ý tưởng:**
 - Chuyển mô hình sang TensorFlow Lite hoặc ONNX.
 - Chạy game trực tiếp trên smartphone hoặc tablet.
- **Lợi ích:** Người dùng không cần PC mạnh, dễ tiếp cận.
- **Thách thức:**
 - Hạn chế tài nguyên (RAM, GPU di động).
 - Cần nén mô hình nhưng vẫn giữ độ chính xác.

7.7 Ứng dụng ngoài game

- **Điều khiển robot:** ra hiệu tay để robot di chuyển, dừng lại.
- **Giáo dục & đào tạo:** dùng để dạy học tương tác (giơ tay phát biểu, trả lời câu hỏi).
- **Y tế:** theo dõi phục hồi chức năng bàn tay.

KẾT LUẬN

Trong khuôn khổ đề tài “Nhận diện cử chỉ tay ứng dụng vào điều khiển game đua xe”, nhóm đã nghiên cứu, xây dựng và triển khai thành công một hệ thống nhận diện cử chỉ tay dựa trên **MobileNetV2 kết hợp GRU**. Hệ thống có khả năng phân loại 5 cử chỉ tay cơ bản và điều khiển xe đua trong game theo thời gian thực.

Các kết quả đạt được:

- **Mô hình đạt độ chính xác ~97.7%** trên tập dữ liệu kiểm thử.
- **Game chạy ổn định** với tốc độ khung hình >20 FPS, đảm bảo tính tương tác mượt mà.
- **MediaPipe Hands** hỗ trợ hiệu quả trong việc phát hiện bàn tay và giảm nhiễu nền.
- Công cụ thu thập dữ liệu được xây dựng giúp việc huấn luyện và mở rộng tập dữ liệu trở nên dễ dàng.

Tuy nhiên, hệ thống vẫn tồn tại một số hạn chế nhất định:

- Chỉ nhận diện được số lượng cử chỉ hạn chế (5 cử chỉ).
- Mô hình còn phụ thuộc vào điều kiện ánh sáng và môi trường.
- Chưa áp dụng cơ chế smoothing, do đó đôi khi vẫn xảy ra hiện tượng dao động (jitter) trong dự đoán.

Từ những kết quả và hạn chế này, nhóm rút ra nhiều bài học quan trọng: **chất lượng dữ liệu đầu vào và quy trình quản lý dữ liệu** có vai trò then chốt, đồng thời cần cân bằng giữa **độ chính xác và tốc độ xử lý** để đáp ứng yêu cầu thời gian thực.

Đề tài đã chứng minh tiềm năng ứng dụng của công nghệ nhận diện cử chỉ tay trong lĩnh vực **Human–Computer Interaction (HCI)**, không chỉ trong game mà

còn có thể mở rộng sang nhiều lĩnh vực khác như **robot**, **y tế**, **giáo dục**, và **hệ thống thông minh**.

Trong tương lai, nhóm hướng đến việc mở rộng tập cử chỉ, áp dụng smoothing/ensemble để cải thiện độ ổn định, tối ưu mô hình cho thiết bị di động, và tích hợp với AR/VR để nâng cao trải nghiệm người dùng.

Như vậy, đề tài đã đạt được các mục tiêu đề ra, đồng thời mở ra nhiều hướng phát triển và ứng dụng thực tiễn.