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


# Ước lượng điểm đặc trưng sân bóng đá bằng mô hình YOLO

Nguyễn Thị Thu Phương
Nguyễn Ngọc Huyền
Phạm Thị Hồng Duyên
Khổng Minh Châu

Mã học phần: MAT3508
Học kỳ 1, Năm học 2025-2026

# Thông tin Dự án

*[Thông tin này cũng được ghi trong README.md của kho GitHub.]*

**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:** Ước lượng điểm đặc trưng sân bóng đá bằng mô hình YOLO
**Ngày nộp:** 30/11/2025
**Báo cáo PDF:** Liên kết tới báo cáo PDF trong kho GitHub
**Slide thuyết trình:** Liên kết tới slide thuyết trình trong kho GitHub
**Kho GitHub:** `https://github.com/shuyn93/sight-and-await.git`

# Thành viên nhóm

| Họ tên | Mã sinh viên | Tên GitHub | Đóng góp |
|---|---|---|---|
| Nguyễn Thị Thu Phương | 22001729 | shuyn93 | Nhóm trưởng: Phân chia công việc, phụ trách tổng quan và dữ liệu mô hình, tìm hiểu lí thuyết mô hình YOLOv8x-pose, kiểm tra chất lượng công việc thành viên, hỗ trợ viết mã, sửa mã lập trình, đưa ra hướng cải thiện mô hình |
| Phạm Thị Hồng Duyên | 23001511 | 23001511-sudo | Phụ trách mô hình YOLOv8x-pose, chiến lược huấn luyện |
| Khổng Minh Châu | 23001503 | Chau23001503 | Phụ trách đánh giá, so sánh các phiên bản mô hình; phát triển các phương pháp dự đoán và tối ưu hóa mô hình, thực nghiệm so sánh và kết quả |
| Nguyễn Ngọc Huyền | 23001527 | kidokyubi | Phụ trách triển khai ứng dụng; hỗ trợ tìm hiểu lí thuyết các phương pháp dự đoán, thực nghiệm, so sánh kết quả |

# Lời cảm ơn

Trong quá trình thực hiện và hoàn thành báo cáo cuối kỳ với đề tài *"Ước lượng điểm đặc trưng sân bóng đá bằng mô hình YOLO"*, nhóm chúng em đã nhận được rất nhiều sự quan tâm, giúp đỡ và hỗ trợ từ phía nhà trường, thầy cô và bạn bè. Trước hết, chúng em xin chân thành cảm ơn Ban Giám hiệu Trường Đại học Khoa học Tự nhiên – Đại học Quốc gia Hà Nội cùng Khoa Toán – Cơ – Tin học đã tạo điều kiện thuận lợi về chương trình học tập, cơ sở vật chất và môi trường học thuật để chúng em có cơ hội tiếp cận và nghiên cứu những nội dung chuyên sâu về trí tuệ nhân tạo.

Nhóm xin gửi lời cảm ơn trân trọng tới giảng viên phụ trách học phần *Nhập môn trí tuệ nhân tạo* Thầy Hoàng Anh Đức đã tận tình giảng dạy, định hướng nội dung, cung cấp tài liệu cũng như góp ý cho chúng em trong suốt quá trình thực hiện dự án. Những kiến thức lý thuyết và ví dụ thực hành trên lớp là nền tảng quan trọng giúp nhóm có thể hệ thống hoá lại, mở rộng thêm và hoàn thiện bản báo cáo này.

Bên cạnh đó, chúng em cũng xin cảm ơn các thầy cô, anh chị và các bạn trong ngành *Khoa học dữ liệu* đã chia sẻ kinh nghiệm học tập, hỗ trợ trao đổi chuyên môn, cũng như động viên tinh thần để nhóm có thêm động lực hoàn thành bài báo cáo đúng thời hạn. Cuối cùng, chúng em xin cảm ơn các thành viên trong nhóm đã làm việc nghiêm túc, có trách nhiệm, phân công hợp lý và hỗ trợ lẫn nhau trong suốt quá trình thực hiện.

Tuy nhiên báo cáo khó tránh khỏi những thiếu sót về nội dung và cách trình bày. Nhóm rất mong nhận được những nhận xét và góp ý quý báu của thầy/cô và các bạn để có thể hoàn thiện hơn trong các nghiên cứu và báo cáo sau này.

Hà Nội, năm 2025
*Nhóm sinh viên thực hiện*

# Mục lục

# Danh sách hình vẽ

# Danh sách bảng

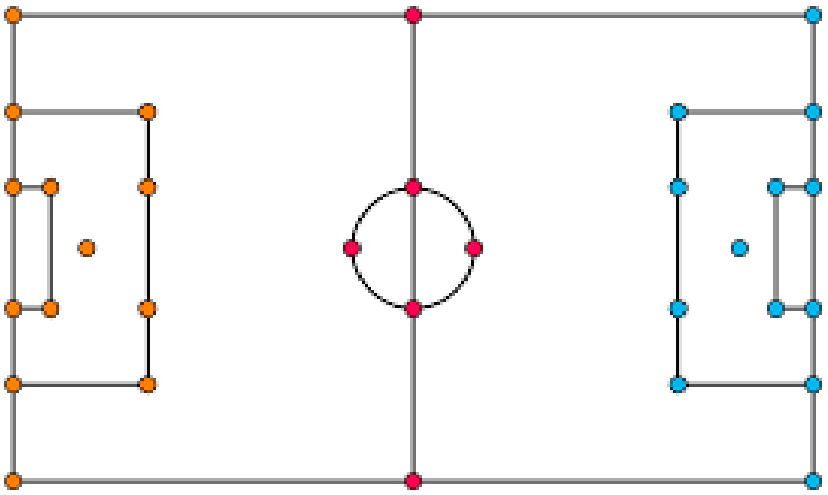
# Chương 1

# Giới thiệu

## 1.1 Tóm tắt

Trong dự án này, nhóm tiến hành xây dựng một hệ thống tự động nhận diện các điểm đặc trưng (keypoints) trên sân bóng đá bằng cách ứng dụng mô hình học sâu YOLOv8-pose. Các điểm đặc trưng bao gồm những vị trí quan trọng như bốn góc sân, tâm sân, điểm penalty, giao điểm các đường giới hạn khu vực cấm địa, đường biên, đường khung thành và nhiều cấu trúc hình học khác. Nhờ khả năng trích xuất đồng thời cả bounding box và tập keypoints của YOLOv8-pose, mô hình có thể mô tả đầy đủ cấu trúc sân bóng chỉ từ một ảnh đầu vào.

Để đạt được độ chính xác cao và khả năng tổng quát hóa tốt, nhóm đã xây dựng một bộ dữ liệu lớn gồm khoảng 66.200 ảnh có gán nhãn đầy đủ 32 keypoints, kết hợp nhiều điều kiện môi trường, góc quay và loại sân khác nhau. Dự án cũng triển khai quá trình huấn luyện, tinh chỉnh siêu tham số, đánh giá hiệu suất trên tập kiểm thử, cũng như thử nghiệm mô hình trên dữ liệu video. Ngoài ra, nhóm xây dựng giao diện web minh hoạ cho phép người dùng tải ảnh và quan sát trực tiếp kết quả suy luận. Kết quả cuối cùng cho thấy mô hình đạt độ chính xác cao (Pose mAP50 = 0.962) cùng tốc độ xử lý đủ nhanh (ms giây khi xử lý trên CPU) để ứng dụng trong các hệ thống phân tích video bóng đá thời gian thực.



Hình 1.1: Cụ thể vị trí 32 điểm đặc trưng trên sân bóng đá[1]

## 1.2 Bài toán đặt ra

Trong những năm gần đây, lĩnh vực phân tích thể thao đã chứng kiến sự phát triển mạnh mẽ nhờ vào khả năng khai thác dữ liệu video và hình ảnh. Các hệ thống thu thập dữ liệu từ camera cố định hoặc di động trên sân bóng đá cung cấp lượng thông tin khổng lồ về vị trí cầu thủ, quỹ đạo bóng, và các sự kiện quan trọng trong trận đấu. Tuy nhiên, để tận dụng hiệu quả dữ liệu này, việc chuẩn hóa không gian sân bóng và nhận diện các điểm đặc trưng *keypoints* trên sân là điều kiện tiên quyết.

Nhận diện chính xác các keypoints trên sân giúp định hình khung tham chiếu chung cho tất cả các camera, đảm bảo rằng các phân tích chiến thuật, thống kê, hay các ứng dụng tăng cường thực tế (AR/VR) đều dựa trên cơ sở không gian chính xác. Đồng thời, với sự bùng nổ của các giải pháp AI trong xử lý hình ảnh và video, việc xây dựng mô hình tự động có khả năng phát hiện các điểm quan trọng trên sân trở thành một bài toán thiết thực, vừa có giá trị học thuật, vừa ứng dụng cao trong thực tế.

Trong bối cảnh đó, bài toán *nhận diện sân bóng đá và các keypoints quan trọng trên sân* trở thành nền tảng cho nhiều ứng dụng phía sau:

- Hiệu chỉnh camera trong hệ thống VAR: từ các keypoints chuẩn trên sân, ta có thể ước lượng ma trận biến đổi giữa mặt phẳng sân và hình ảnh, phục vụ việc dựng lại quỹ đạo bóng/cầu thủ.
- Phân tích chiến thuật: việc xác định chính xác biên sân, vòng cấm, vị trí khung thành giúp gắn các sự kiện trong trận đấu với không gian chuẩn hoá.
- Ứng dụng AR/VR: các keypoints là cơ sở để chồng lớp thông tin (stats, heatmap, highlight) lên khung hình thực tế.

Bài toán cụ thể đặt ra trong học phần AI là: **xây dựng mô hình YOLOv8-pose có khả năng nhận diện 32 điểm đặc trưng trên sân bóng đá với độ chính xác cao (mAP Pose $\geq 0.9$ trên tập kiểm thử), đồng thời đảm bảo tốc độ suy luận đủ lớn để tiệm cận thời gian thực trên GPU**.

Các thách thức chính bao gồm:

- Dữ liệu đầu vào đa dạng về góc nhìn (từ trên cao, từ khán đài, góc chéo), điều kiện ánh sáng (ngày, đêm, mưa, sương mù) và loại sân (cỏ tự nhiên, cỏ nhân tạo, sân trong nhà).
- Keypoints mang tính *ngữ nghĩa* (ví dụ: điểm penalty, giao điểm vạch 16m50) chứ không chỉ là "góc cạnh" thuần tuý nên các phương pháp truyền thống như Harris, SIFT, ORB không còn phù hợp.
- Yêu cầu về tốc độ xử lý: mô hình phải chạy đủ nhanh trên CPU, GPU phổ thông (T4, V100) để phục vụ xử lý video liên tục hoặc gần thời gian thực.

[1]Nguồn ảnh: `https://blog.roboflow.com/camera-calibration-sports-computer-vision/`

# Chương 2

# Phương pháp & Triển khai

## 2.1 Phương pháp

### 2.1.1 Dữ liệu sử dụng

Dữ liệu là thành phần quan trọng nhất quyết định chất lượng mô hình YOLOv8-pose trong bài toán nhận diện 32 keypoints trên sân bóng đá. Do không tồn tại một bộ dữ liệu chuẩn hoá và thống nhất cho toàn bộ các loại sân bóng, dự án tiến hành tổng hợp dữ liệu từ nhiều bộ dữ liệu con khác nhau trên Roboflow – nền tảng chuyên về quản lý, chia sẻ và gán nhãn hình ảnh cho các bài toán thị giác máy tính.

#### Nguồn gốc và phương pháp thu thập dữ liệu

Dữ liệu được thu thập chủ yếu từ:

- Các bộ dữ liệu cộng đồng trên Roboflow được công khai bởi nhiều nhóm nghiên cứu, câu lạc bộ bóng đá, hoặc cá nhân quan tâm đến phân tích sân bóng.
- Các bộ dữ liệu tuỳ chỉnh do các workspace khác nhau chia sẻ, bao gồm cả các dự án về *football field detection*, *soccer field localization*, *pitch keypoint detection*, *terrain segmentation* hoặc các dự án liên quan đến tracking và phân tích trận đấu.
- Một phần nhỏ dữ liệu được trích xuất từ video phát sóng bóng đá, camera góc cao, camera cầu môn hoặc các hệ thống phân tích chiến thuật.

Việc lựa chọn Roboflow làm nguồn dữ liệu giúp:

- Tận dụng được các bộ dữ liệu có sẵn, giảm thời gian tự gán nhãn từ đầu.
- Truy cập nhiều phong cách gán nhãn khác nhau: bounding boxes, keypoints, segmentation.
- Dễ dàng chuyển đổi sang định dạng YOLOv8 và quản lý annotation thống nhất.

Trong dự án, danh sách dataset trong mã nguồn chỉ mang tính minh hoạ. Thực tế nhóm đã gom từ số lượng lớn hơn nhiều bộ dữ liệu con để hướng tới độ đa dạng cao nhất về:

- Góc nhìn: góc cao (bird view), góc thấp (broadcast), góc chéo;
- Điều kiện ánh sáng: ngày, đêm, mưa, sương mù;

- Loại sân: cỏ tự nhiên, cỏ nhân tạo, sân mini, sân futsal trong nhà;
- Độ phân giải: từ 640p đến 4K.

### Quy trình hợp nhất và chuẩn hoá dữ liệu

Mỗi bộ dữ liệu khi tải từ Roboflow được lưu theo chuẩn thư mục:

```
dataset/
  train/
      images/
      labels/
  valid/
  test/
```

Do các bộ dữ liệu đến từ nhiều nguồn khác nhau, việc hợp nhất dữ liệu cần qua các bước:

1. **Tải dữ liệu**: Sử dụng Roboflow API để tải từng dataset theo workspace, project và version tương ứng.
2. **Chuẩn hóa annotation**: chuyển toàn bộ nhãn về định dạng YOLOv8-pose (keypoints).
3. **Hợp nhất**: gộp tất cả ảnh và nhãn từ các bộ dữ liệu con vào một thư mục chung.
4. **Loại bỏ ảnh trùng lặp và ảnh lỗi**: dựa vào tên file, hash ảnh hoặc cấu trúc nhãn.
5. **Shuffle và chia lại train/val/test**: theo tỷ lệ 80% – 10% – 10%.

Quy trình này bảo đảm:

- các tập train/val/test chứa hình ảnh đa dạng, ít trùng lặp,
- dữ liệu được phân phối đồng đều theo góc nhìn và điều kiện môi trường,
- mô hình học được khả năng khái quát hóa tốt hơn khi gặp dữ liệu thực tế.

### Cấu trúc thư mục dữ liệu sau khi chuẩn hoá

Sau khi hợp nhất và làm sạch dữ liệu, toàn bộ ảnh và nhãn được tổ chức lại theo đúng chuẩn thư mục của YOLOv8. Cấu trúc cuối cùng của bộ dữ liệu được dùng để huấn luyện mô hình như sau:

```
dataset/

 images/
    train/
        xxx.jpg / xxx.png
    val/
        xxx.jpg / xxx.png
    test/
         xxx.jpg / xxx.png
```

```
labels/
   train/
       xxx.txt
   val/
       xxx.txt
   test/
        xxx.txt

data.yaml
```

Trong đó:

- **images/** chứa ảnh đầu vào cho mô hình.
- **labels/** chứa các file nhãn định dạng YOLO (`.txt`), mỗi dòng gồm: `class x_center y_center kp1_x kp1_y kp1_v ... kp32_x kp32_y kp32_v`.
- Tên file ảnh và tên file label luôn tương ứng nhau, ví dụ: `abc.jpg` ↔ `abc.txt`.

Cách tổ chức này bảo đảm tương thích hoàn toàn với Ultralytics YOLOv8-pose, giúp mô hình có thể tải dữ liệu nhanh và chính xác trong quá trình huấn luyện.

### Cấu trúc file `data.yaml`

YOLOv8 yêu cầu một file cấu hình `data.yaml` để xác định đường dẫn dữ liệu, số lượng lớp và thông tin keypoints. Nội dung file được sử dụng trong dự án như sau:

```
# Dataset paths
train: images/train
val: images/val
test: images/test

# Number of classes
nc: 1

# Class names
names: ["pitch"]

# Keypoint information
kpt_shape: [32, 3]   # 32 keypoints, mỗi keypoint gồm (x, y, visibility)

# Flip index (dùng khi lật ảnh trái-phải để augmentation)
flip_idx: [
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
  10, 11, 12, 13, 14, 15, 16, 17,
  18, 19, 20, 21, 22, 23, 24, 25,
  26, 27, 28, 29, 30, 31
]
```

Ý nghĩa của các trường:

- **train / val / test**: chỉ đường dẫn tương đối tới thư mục chứa ảnh của từng tập dữ liệu.
- **nc**: số lượng lớp cần nhận diện (1 lớp: sân bóng).
- **names**: tên của lớp duy nhất.
- **kpt_shape**: định nghĩa cấu trúc keypoints — bao gồm 32 điểm, mỗi điểm có 3 giá trị $(x, y, v)$ với $v \in \{0, 1, 2\}$ (mất, che khuất, hiện rõ).
- **flip_idx**: ánh xạ các điểm keypoints khi thực hiện phép lật ảnh trái–phải (left-right flip).

Cấu trúc file `data.yaml` này là yêu cầu bắt buộc để YOLOv8-pose hiểu đúng định dạng dữ liệu và skeleton của 32 keypoints. Nó đảm bảo mô hình được huấn luyện nhất quán, đặc biệt khi sử dụng các kỹ thuật augmentation liên quan như `horizontal flip`.

**Tổng quan dữ liệu**

Tổng số ảnh sau khi hợp nhất và làm sạch là **khoảng 66.000 ảnh**, đến từ hàng chục nguồn Roboflow khác nhau. Bộ dữ liệu có sự phong phú về bố cục sân, màu cỏ, vị trí camera, và các loại sân, giúp mô hình YOLOv8x-pose đạt được độ chính xác mAP cao và khả năng tổng quát hóa tốt.

### 2.1.2 Mô hình đề xuất

## Đặc tả bài toán và dạng đầu ra

Bài toán có thể được mô tả như sau: với mỗi ảnh đầu vào $I \in R^{H \times W \times 3}$, mô hình cần dự đoán:

- Hộp bao (bounding box) của sân bóng: toạ độ $(x, y, w, h)$ trong hệ toạ độ ảnh.
- Tập các điểm đặc trưng (keypoints) $\{(u_i, v_i)\}_{i=1}^{K}$ với $K = 32$, tương ứng với các góc sân, điểm giữa sân, điểm penalty, giao điểm đường 16m50, v.v.

Đầu ra của mô hình gồm hai phần:

1. **Head detection (Box head)**: dự đoán hộp bao và confidence của sân.
2. **Head pose (Keypoint head)**: dự đoán toạ độ chuẩn hoá của $K$ keypoints gắn với mỗi bounding box tương ứng.

## Lý do chọn YOLOv8-pose

Bài toán của nhóm không chỉ là phát hiện sân bóng (object detection) mà còn phải dự đoán 32 keypoint( các góc sân, chấm penalty, giao điểm vạch 16m50, điểm giữa sân. . . ). Đây là bài toán detection + pose estimation trong một model duy nhất.

So với các hướng tiếp cận khác:

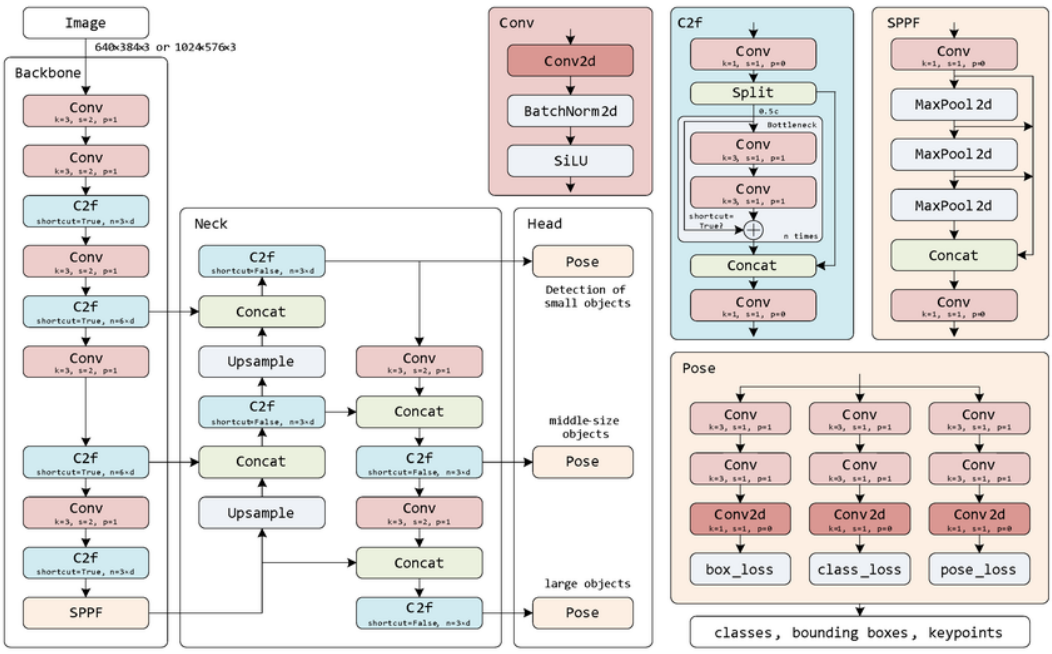
- Các detector cổ điển như Faster R-CNN, RetinaNet cần thêm một nhánh (head) tuỳ chỉnh để dự đoán keypoints và thường chậm hơn YOLO, khó tối ưu thời gian suy luận.

- Các mô hình chuyên cho human pose (OpenPose, HRNet) mạnh nhưng nặng, thiết kế cho skeleton người, phải chỉnh sửa khá nhiều để phù hợp với keypoints của sân bóng.
- YOLOv8-pose:
  - Gộp detection & pose trong một mô hình duy nhất, tối ưu pipeline từ đầu vào đến đầu ra.
  - Anchor-free, dễ huấn luyện và ổn định khi triển khai.
  - Thư viện Ultralytics hỗ trợ mạnh về huấn luyện, đánh giá, export (ONNX, TensorRT), rất phù hợp với yêu cầu của một dự án môn học và kết nối với thực tập.

## Kiến trúc YOLOv8-pose

YOLOv8x-pose là biến thể của họ mô hình YOLOv8 được mở rộng cho bài toán *pose estimation*, cho phép dự đoán đồng thời hộp bao (bounding box) và tập keypoint cho mỗi đối tượng. Trong đồ án này, đối tượng là *sân bóng*, với mỗi sân được gán 32 keypoint hình học (các góc sân, chấm penalty, giao điểm vạch 16m50, tâm sân, ...). Về tổng thể, kiến trúc YOLOv8x-pose có thể được chia thành ba khối chính:

- **Backbone**: trích xuất đặc trưng từ ảnh đầu vào.
- **Neck (FPN/PAN)**: kết hợp đặc trưng đa tỉ lệ (multi-scale feature fusion).
- **Heads**: bao gồm detection head (dự đoán box) và pose head (dự đoán keypoint).




Hình 2.1: Minh họa kiến trúc mô hình yolov8x-pose[1]

[1]Nguồn ảnh: https://www.researchgate.net/publication/The-architecture-of-the-YOLOv8-pose-model-used-for-detecting-both-bounding-boxes-and.tif

Trong hình 2.1 này, Backbone trích xuất đặc trưng từ ảnh đầu vào thông qua các khối Conv, C2f và SPPF giúp mô hình học các đặc trưng từ thấp đến cao. Neck kết hợp các đặc trưng ở nhiều mức tỉ lệ bằng cơ chế upsample và concat để mô hình nhận diện được cả đối tượng nhỏ, vừa và lớn. Cuối cùng, Head dự đoán bounding boxes, class scores và keypoints (32 điểm pose) cho từng đối tượng.

### (i) Backbone: C2f và SPPF

Backbone của YOLOv8 kế thừa ý tưởng từ YOLOv5 nhưng thay thế block CSP bằng block C2f nhằm tăng hiệu quả biểu diễn trong khi vẫn giảm chi phí tính toán.

#### * Block C2f (Cross-Stage Feature Fusion)

C2f (Cross-Stage Feature Fusion) là block cơ bản trong backbone của YOLOv8, có nhiệm vụ:

- Chia đặc trưng thành nhiều nhánh (*branch*) để xử lý song song qua các lớp tích chập.
- Hợp nhất (*feature fusion*) các nhánh ở cuối block để thu được đặc trưng giàu thông tin hơn.

Về trực giác, C2f vừa kế thừa ý tưởng "chia nhánh" của CSP, vừa bổ sung cơ chế trộn đặc trưng, giúp mạng:

- Ở các tầng nông: học được các đặc trưng cục bộ như cạnh, vạch trắng, góc sân.
- Ở các tầng sâu: học được hình dạng tổng thể của sân, đường biên, vòng tròn giữa sân, vòng cấm, ...

#### * SPPF (Spatial Pyramid Pooling Fast)

SPPF (Spatial Pyramid Pooling Fast) là phiên bản tối ưu của Spatial Pyramid Pooling, được đặt ở cuối backbone. Ý tưởng chính:

- Áp dụng nhiều lớp *max pooling* tuần tự lên cùng một feature map với cùng kích thước kernel (thông thường là $5 \times 5$).
- Ghép (*concatenate*) các output này với feature ban đầu để thu được đặc trưng ở nhiều "tầm nhìn" khác nhau.

Nhờ đó, SPPF cho phép mô hình:

- "Nhìn gần" để bắt được chi tiết vạch sân.
- "Nhìn xa" để nắm bắt cấu trúc lớn hơn như toàn bộ vòng cấm hay biên sân.

### (ii) Neck: FPN và PAN

Neck của YOLOv8 sử dụng sự kết hợp giữa FPN (Feature Pyramid Network) và PAN (Path Aggregation Network) nhằm trộn đặc trưng từ nhiều tầng khác nhau.

*** FPN (Feature Pyramid Network)**

FPN thực hiện lan truyền thông tin theo hướng **top–down**:

- Từ các tầng sâu (đặc trưng trừu tượng, độ phân giải thấp) → các tầng nông hơn (độ phân giải cao hơn).
- Giúp mô hình giữ được thông tin "toàn cảnh" về sân bóng ở các tầng có độ phân giải cao.

*** PAN (Path Aggregation Network)**

PAN thực hiện lan truyền thông tin theo hướng **bottom–up**:

- Từ các tầng nông (đặc trưng chi tiết, vạch mảnh) → lên các tầng sâu hơn.
- Tăng cường các chi tiết quan trọng, tránh mất mát các vạch trắng nhỏ hoặc keypoint xa camera.

Nhờ kết hợp FPN và PAN, YOLOv8x-pose có thể đồng thời:

- Nhận diện được *toàn bộ sân* (global context).
- Bảo toàn được các chi tiết hình học quan trọng (local details) phục vụ dự đoán keypoint.

**(iii) Heads**

*** Detection Head: Anchor-free**

YOLOv8 sử dụng cơ chế **anchor-free** cho detection head, khác với các phiên bản YOLO cũ. Thay vì định nghĩa trước một tập anchor box, mô hình:

- Đặt một lưới (*grid*) trên feature map.
- Tại mỗi vị trí lưới, mô hình dự đoán trực tiếp:

$$(x, y, w, h, \text{objectness})$$

với $(x, y)$ là toạ độ tâm box, $(w, h)$ là kích thước và *objectness* thể hiện xác suất "có sân".

Cách tiếp cận anchor-free giúp:

- Đơn giản hoá quá trình thiết kế anchor, đặc biệt trong bối cảnh sân bóng có thể xuất hiện với nhiều tỉ lệ và góc nhìn khác nhau.
- Giảm độ phức tạp của mô hình, cải thiện tốc độ suy luận.

*** Pose Head: Dự đoán 32 keypoint**

Pose head là phần mở rộng quan trọng nhất trong YOLOv8x-pose so với YOLOv8 gốc. Ngoài box, mỗi đối tượng (sân bóng) còn được gán:

- $K = 32$ keypoint: biểu diễn các điểm hình học quan trọng trên sân.
- Mỗi keypoint gồm toạ độ $(u_i, v_i)$ và một giá trị *visibility* (nhìn thấy / bị che khuất).

Tổng số output cho phần pose là:

$$32 \times (u, v, \text{visibility}) = 96 \text{ giá trị.}$$

Dự đoán dưới dạng offset:
Thay vì dự đoán trực tiếp toạ độ tuyệt đối trên ảnh, YOLOv8x-pose dự đoán **offset** của mỗi keypoint so với hộp bao (bounding box) của sân:

$$(u_i, v_i) = (x_{\text{box}}, y_{\text{box}}) + (\Delta u_i, \Delta v_i) \odot (w_{\text{box}}, h_{\text{box}})$$

trong đó:

- $(x_{\text{box}}, y_{\text{box}})$ là tâm hộp bao.
- $(w_{\text{box}}, h_{\text{box}})$ là chiều rộng và chiều cao hộp bao.
- $(\Delta u_i, \Delta v_i)$ là offset đã được chuẩn hoá theo kích thước box.

Cách dự đoán này có các ưu điểm:

- Ổn định hơn khi kích thước sân thay đổi (zoom in/out, sân xa hoặc gần camera).
- Mô hình học được quan hệ hình học giữa hình dạng sân và vị trí các điểm ngữ nghĩa (chấm penalty, giao điểm vạch 16m50, ...).

## Hàm mất mát

Trong quá trình huấn luyện, YOLOv8-pose tối ưu một hàm mất mát tổng hợp:

$$\mathcal{L} = \lambda_{\text{box}}\mathcal{L}_{\text{box}} + \lambda_{\text{obj}}\mathcal{L}_{\text{obj}} + \lambda_{\text{cls}}\mathcal{L}_{\text{cls}} + \lambda_{\text{pose}}\mathcal{L}_{\text{pose}}$$

Trong đó:

- $L_{\text{box}}$: đo độ khớp giữa box dự đoán và box thật (thường dùng CIoU/DIoU).
- $L_{\text{obj}}$: loss cho *objectness* (có/không có sân bóng).
- $L_{\text{cls}}$: loss phân lớp (trong bài toán này chỉ có một lớp "football pitch", nên thành phần này đơn giản).
- $L_{\text{pose}}$: lỗi dự đoán keypoint, thường dùng chuẩn $L_1$ hoặc $L_2$, có kèm *mask* để bỏ qua các điểm bị che khuất (visibility = 0).

Các hệ số $\lambda$ được lựa chọn theo đề xuất của Ultralytics và được tinh chỉnh nhẹ qua một số thí nghiệm nhỏ, ưu tiên giữ cân bằng giữa việc vẽ box đúng và dự đoán keypoints chính xác.

### 2.1.3 Chiến lược huấn luyện

## Dữ liệu và tiền xử lý

Bộ dữ liệu được xây dựng theo hướng lặp dần:

- Giai đoạn 1: $\sim$ 300 ảnh sân bóng, chủ yếu để kiểm chứng pipeline huấn luyện và nhãn keypoints.
- Giai đoạn 2–4: mở rộng lên $\sim$ 700, $\sim$ 1,400, $\sim$ 4,000 và $\sim$ 11,000 ảnh, bổ sung thêm nhiều góc quay, điều kiện ánh sáng, loại sân.
- Giai đoạn cuối: $\sim$ 66,000 **ảnh** đã gán nhãn đầy đủ 32 keypoints, là cơ sở để huấn luyện mô hình cuối cùng.

Các bước tiền xử lý:

- Chuẩn hoá kích thước ảnh về **640, 1024 hoặc 1280** (tuỳ thí nghiệm), giữ đúng tỉ lệ để tránh méo hình sân.
- Tăng cường dữ liệu (data augmentation) tích hợp sẵn trong Ultralytics: lật ngang, thay đổi độ sáng, tương phản, xoay nhẹ, blur nhẹ để mô hình quen với nhiều điều kiện thu hình khác nhau.
- Làm sạch dữ liệu: loại bỏ các ảnh quá mờ, crop quá sát hoặc che khuất phần lớn sân, nhằm đảm bảo mô hình học được cấu trúc sân rõ ràng.

Dữ liệu được chia theo tỉ lệ:

$$\text{Train} : \text{Val} : \text{Test} \approx 8 : 1 : 1$$

giúp có đủ mẫu để vừa huấn luyện, vừa điều chỉnh siêu tham số và đánh giá khách quan.

## Khảo sát các siêu tham số quan trọng

- **Kích thước ảnh**: so sánh 640, 1024, 1280.
  - Ảnh nhỏ (640): nhanh, phù hợp thử nghiệm nhanh.
  - Ảnh trung bình (1024): cân bằng giữa chi tiết đường kẻ và thời gian huấn luyện.
  - Ảnh lớn (1280): chi tiết tốt nhất nhưng tốn tài nguyên, thích hợp cho thí nghiệm cuối khi đã ổn định workflow.
- **Batch size**: Số ảnh dùng trong một lần cập nhật trọng số. Với batch lớn, gradient ổn định, học mượt nhưng tốn VRAM. Với batch nhỏ, gradient nhiễu nhưng đôi khi tổng quát tốt hơn.
- **Số epoch**: bắt đầu với $\sim$ 100 epoch để quan sát đường loss, sau đó tăng lên 150–200 epoch khi dùng dataset lớn; theo dõi hiện tượng overfitting qua độ chênh lệch loss train/val.
- **Độ chính xác số học**: huấn luyện và suy luận với cả `fp32` và `fp16` để đánh giá ảnh hưởng đến mAP và thời gian suy luận.

### 2.1.4 Thước đo đánh giá

Các thước đo chính được sử dụng trong đánh giá mô hình bao gồm:

- **Box mAP50, Box mAP50–95:** đánh giá khả năng phát hiện sân bóng.
- **Pose mAP50, Pose mAP50–95:** đánh giá độ chính xác của 32 keypoints.
- **Thời gian suy luận (ms/frame):** đo trên GPU (T4 hoặc V100), từ khi nạp ảnh đến khi nhận được tọa độ keypoints.

## Định nghĩa mAP

Độ chính xác trung bình (**mAP**) là thước đo hiệu suất quan trọng trong thị giác máy tính, đặc biệt đối với các mô hình phát hiện đối tượng và định vị keypoint. Khác với độ chính xác phân loại đơn giản chỉ kiểm tra nhãn hình ảnh, mAP đánh giá đồng thời khả năng phân loại chính xác các đối tượng (*Box*) và định vị chính xác chúng (*Pose*) trong ảnh bằng hộp giới hạn hoặc keypoint. Vì vậy, mAP trở thành tiêu chuẩn công nghiệp để so sánh các mô hình tiên tiến như YOLOv8-pose với các kiến trúc hiện đại khác.

## Thành phần cấu thành

Để tính mAP, mô hình sử dụng ba thành phần cơ bản:

- **Intersection over Union (IoU):** đo lường sự chồng lấp không gian giữa dự đoán và nhãn thực tế, giá trị trong khoảng $[0, 1]$. IoU cao cho thấy dự đoán sát với thực tế.
- **Precision (Độ chính xác):** tỉ lệ dự đoán đúng trên tổng số dự đoán, phản ánh độ tin cậy của mô hình.
- **Recall (Thu hồi):** tỉ lệ dự đoán đúng trên tổng số đối tượng thực tế, phản ánh khả năng mô hình phát hiện tất cả các đối tượng.

## Công thức tính mAP:

AP (Average Precision) được tính cho từng lớp là diện tích dưới đường cong Precision-Recall, và mAP là trung bình các AP trên tất cả các lớp:

$$AP_i = \int_0^1 P_i(R)\, dR, \qquad mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i$$

Trong đó $P_i(R)$ là độ chính xác tại mức thu hồi $R$ của lớp thứ $i$, $N$ là tổng số lớp.

## Các chỉ số phổ biến:

- **Box mAP50:** đánh giá độ chính xác của bounding box với ngưỡng IoU $\geq 0.5$.
- **Box mAP50–95:** trung bình mAP trên các ngưỡng IoU từ 0.5 đến 0.95, phản ánh hiệu suất khi yêu cầu định vị chính xác hơn.

- **Pose mAP50:** đánh giá độ chính xác định vị keypoint với IoU $\geq 0.5$.
- **Pose mAP50–95:** trung bình mAP keypoint trên các ngưỡng IoU từ 0.5 đến 0.95, đánh giá độ ổn định định vị keypoint.

### Ý nghĩa thực tiễn:

- Giá trị mAP càng gần 1 → mô hình càng chính xác và ổn định trong việc phát hiện đối tượng và định vị keypoint.
- Box mAP50 cao → mô hình phát hiện đối tượng chính xác, hữu ích cho ứng dụng real-time.
- Pose mAP50 cao → mô hình định vị keypoint tốt, thích hợp cho phân tích chuyển động hoặc đo lường vị trí chi tiết.
- mAP50–95 cao → độ ổn định tốt khi yêu cầu định vị chặt chẽ.

### Khoảng giá trị:

- 0: dự đoán hoàn toàn sai.
- 1: dự đoán hoàn toàn chính xác.

### Xu hướng cải thiện mAP:

Các hướng nâng cao hiệu suất mô hình bao gồm:

- Tăng số lượng và chất lượng dữ liệu huấn luyện.
- Sử dụng mô hình lớn hơn, multi-scale input, augmentation.
- Điều chỉnh anchor, loss function, learning rate.

### Phân tích kết quả

Trong báo cáo, phân tích tập trung so sánh: (i) ảnh hưởng của kích thước ảnh; (ii) ảnh hưởng của việc tăng quy mô dữ liệu; và (iii) trade-off giữa độ chính xác và tốc độ khi dùng fp16 so với fp32.

## 2.1.5 Các phương pháp tăng tốc độ suy luận khi dự đoán

Bên cạnh việc đạt mAP cao, rubric yêu cầu sinh viên thể hiện hiểu biết và thực hành các **kỹ thuật tối ưu suy luận**. Dự án đã áp dụng và/hoặc thử nghiệm ba nhóm phương pháp: *tối ưu ở mức mô hình*, *mức framework/phần cứng* và *mức thuật toán suy luận video*.

## Tối ưu ở mức mô hình

**– Lựa chọn kích thước ảnh phù hợp.**

Kích thước ảnh càng lớn thì số phép tính (FLOPs) càng tăng theo bình phương. Sau khi khảo sát, mô hình với ảnh 1024x1024 cho thấy:

- mAP tăng rõ rệt so với 640x640 (đặc biệt với các đường kẻ mảnh, xa camera).
- Thời gian suy luận vẫn giữ ở mức chấp nhận được trên GPU T4.

Vì vậy, cấu hình 1024x1024 được lựa chọn cho phiên bản cân bằng giữa chất lượng và tốc độ.

**– Giảm số lớp/tầng nếu cần.**

Trong một số thí nghiệm, nhóm thử các biến thể nhẹ hơn (như YOLOv8m/yolov8l) để so sánh. Kết quả cho thấy:

- Mô hình nhỏ hơn cho tốc độ tốt hơn nhưng mAP Pose giảm đáng kể khi xử lý góc quay khó.
- Với mục tiêu ưu tiên độ chính xác, nhóm chấp nhận dùng YOLOv8x-pose nhưng kết hợp với các phương pháp tối ưu suy luận khác (fp16, frame skipping).

## Tối ưu ở mức framework và phần cứng

**– Sử dụng FP16 (mixed precision).**

Suy luận với **fp16** thay vì **fp32**:

- Giảm một nửa dung lượng bộ nhớ cần để lưu trọng số và activation.
- Tận dụng tốt hơn các nhân tính toán chuyên dụng (Tensor Cores) trên GPU NVIDIA.

Các bước thực hiện:

1. Xuất mô hình từ Ultralytics với tuỳ chọn `half=True` (khi suy luận).
2. Kiểm tra định tính bằng cách so sánh visual output (box, keypoints) giữa fp32 và fp16 trên cùng ảnh/video.

Kết quả: fp16 giúp rút ngắn thời gian suy luận rõ rệt, trong khi khác biệt mAP không đáng kể.

**– Export ONNX / TensorRT (thử nghiệm).**

Để chuẩn bị cho việc triển khai thực tế, nhóm thử:

- Export mô hình sang **ONNX**, cho phép chạy trên nhiều runtime tăng tốc.
- Xem xét chuyển tiếp sang **TensorRT** để tối ưu sâu hơn (fusion, kernel tuning).

Trong điều kiện thực tập/học phần:

- Việc export ONNX thành công, nhưng chưa triển khai sâu vì hạn chế thời gian và cấu hình máy.
- Việc build engine TensorRT trên máy cloud mất khá nhiều thời gian, chưa kịp tối ưu triệt để cho phiên bản cuối.

Tuy vậy, phần thử nghiệm ONNX/TensorRT cho thấy hướng triển khai phù hợp trong tương lai khi chuyển sang môi trường sản phẩm.

## Tối ưu ở mức thuật toán suy luận video

Khi áp dụng mô hình lên video, nếu chạy suy luận trên *từng frame* thì chi phí tính toán sẽ rất lớn. Dự án áp dụng các kỹ thuật:

**– Dự đoán thưa (frame skipping) và lặp kết quả.**

Chiến lược đơn giản:

- Chỉ chạy mô hình trên mỗi **frame thứ** $k$ (ví dụ $k = 5$).
- Với các frame ở giữa, **lặp lại** kết quả suy luận của frame gần nhất.

Ưu điểm:

- Tốc độ suy luận tăng xấp xỉ $k$ lần.
- Không cần thêm mô hình/tracking phức tạp.

Nhược điểm: khi camera panning mạnh hoặc cảnh thay đổi nhanh, hình ảnh keypoints có thể bị "giật".

**– Dự đoán thưa kết hợp nội suy.**

Để mượt hơn, nhóm thử nghiệm:

- Chỉ chạy mô hình trên các frame 0, 5, 10, ...
- Với các frame 1–4, nội suy tuyến tính toạ độ keypoints giữa frame 0 và frame 5.

Ý tưởng: các keypoints trên sân và camera thường di chuyển tương đối mượt, nên nội suy tuyến tính cho kết quả chấp nhận được trong đa số trường hợp.
Kết quả:

- FPS tăng mạnh so với việc chạy trên mọi frame.
- Chuyển động của keypoints mượt hơn phương án lặp đơn thuần.
- Với các chuyển động bất thường (zoom nhanh, pan gấp), nội suy tuyến tính vẫn còn hạn chế, gợi ý cần dùng các phương pháp tracking nâng cao hơn (optical flow, Kalman filter,...).

– **Batch inference (nếu bộ nhớ cho phép).**

Thay vì xử lý từng ảnh độc lập, có thể gom nhiều frame thành một batch:

- Lợi dụng tốt hơn khả năng song song của GPU.
- Giảm overhead khởi chạy kernel cho từng frame.

Trong thực nghiệm, batch size nhỏ (2–4) được thử trong giới hạn bộ nhớ GPU, giúp tăng nhẹ throughput tổng thể khi xử lý video offline.

## 2.2 Triển khai

Quá trình triển khai hệ thống nhận diện keypoints sân bóng đá sử dụng mô hình YOLOv8-pose bao gồm mô tả tổng thể hệ thống, các công nghệ được sử dụng và cấu trúc mã nguồn của dự án. Việc tổ chức rõ ràng giúp làm nổi bật pipeline xử lý thống nhất từ dữ liệu, huấn luyện, suy luận cho đến triển khai ứng dụng.

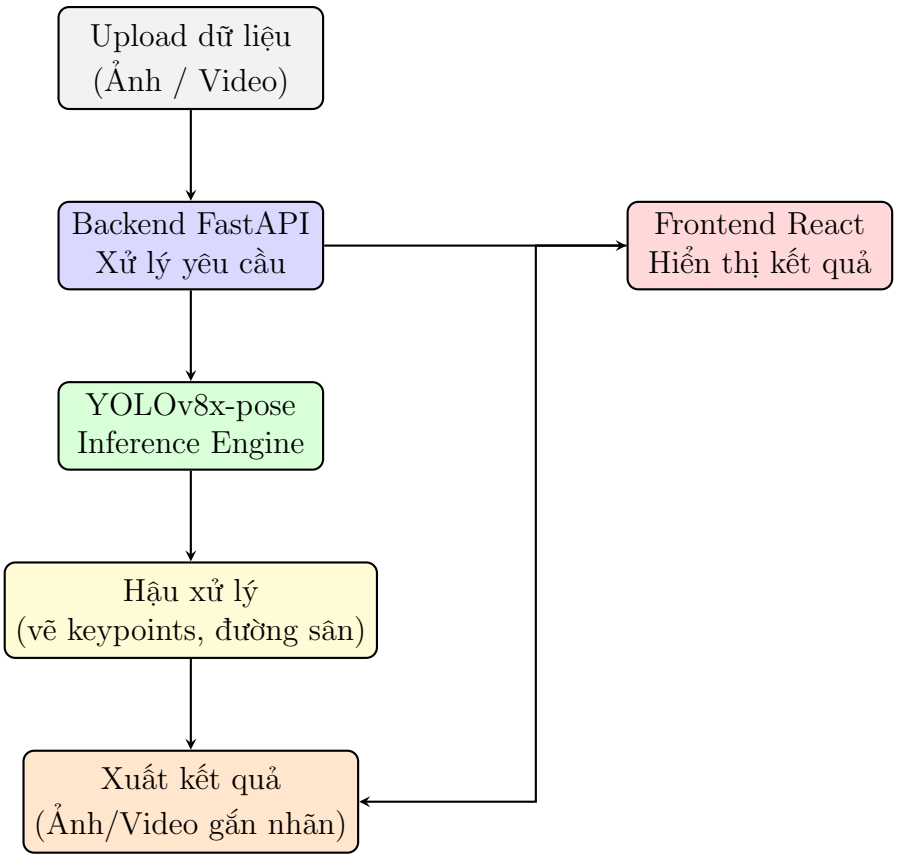
### 2.2.1 Mô tả hệ thống

Hệ thống được thiết kế nhằm thực hiện ba nhiệm vụ chính: (i) huấn luyện mô hình YOLOv8-pose trên bộ dữ liệu lớn, (ii) suy luận keypoints trên ảnh và video sân bóng với tốc độ cao, và (iii) triển khai ứng dụng web phục vụ người dùng cuối. Kiến trúc tổng thể của hệ thống gồm ba tầng chính:

- **Tầng mô hình (Model Layer):** Chứa mô hình YOLOv8x-pose đã huấn luyện. Pipeline suy luận bao gồm: nạp ảnh/video, tiền xử lý, chạy mô hình, hậu xử lý keypoints và sinh kết quả.
- **Tầng xử lý backend (Backend API Layer):** Được xây dựng bằng FastAPI, cung cấp API REST để nhận tệp tin từ người dùng, thực hiện inference và trả về ảnh/video đã gắn nhãn cùng thông tin keypoints. Backend hỗ trợ xử lý bất đồng bộ nhằm giảm độ trễ.
- **Tầng giao diện web (Frontend Layer):** Được xây dựng bằng React kết hợp Vite và TailwindCSS. Cho phép người dùng tải ảnh/video, xem trực quan các điểm keypoints được gán nhãn và xem thống kê hiệu suất suy luận.

Luồng xử lý chính của hệ thống được mô tả như sau:

**Sơ đồ kiến trúc hệ thống nhận diện keypoints sân bóng**




Hình 2.2: Kiến trúc tổng thể của hệ thống YOLOv8-pose cho nhận diện keypoints sân bóng đá.

Thiết kế này bảo đảm hệ thống linh hoạt, có thể mở rộng để bổ sung các chức năng như nhận diện cầu thủ, tracking nhiều đối tượng hoặc triển khai dạng dịch vụ trên cloud.

### 2.2.2 Công cụ và công nghệ sử dụng

Việc lựa chọn công nghệ được dựa trên các tiêu chí: hiệu năng, khả năng mở rộng, dễ sử dụng và tính cộng đồng hỗ trợ. Phần sau mô tả chi tiết từng nhóm công nghệ chính và vai trò cụ thể của chúng trong dự án.

## Công nghệ mô hình hoá và huấn luyện

– **PyTorch:**

PyTorch là framework deep learning chính được sử dụng trong dự án. PyTorch cung cấp:

- **Dynamic computational graph** thuận tiện cho việc debug và phát triển các kiến trúc mới.
- **API phong phú** cho việc xây dựng model, tối ưu hoá và quản lý checkpoint.

- **Hỗ trợ tốt cho GPU** (CUDA) và tích hợp dễ dàng với các thư viện ecosystem (TorchVision, Apex, v.v.).

**Vai trò trong dự án:** Huấn luyện và inference cho YOLOv8x-pose khi sử dụng weights gốc của Ultralytics (đã được implement trên PyTorch).

**Ví dụ lệnh khởi tạo training (PyTorch / script):**

```
python train_yolo.py --img 1024 --batch 16 --epochs 200 --device 0
```

– **Ultralytics YOLOv8**

Ultralytics YOLOv8 là thư viện/implement chính để sử dụng YOLOv8-pose:

- Cung cấp sẵn các biến thể (n, s, m, l, x) và biến thể `-pose` hỗ trợ keypoints.
- Tích hợp pipeline training/validation, logging, export (ONNX/TensorRT), và inference CLI.
- Cộng đồng lớn, tài liệu chi tiết và weights pre-trained có sẵn.

**Vai trò trong dự án:** làm khung chính để huấn luyện YOLOv8x-pose, tinh chỉnh head pose, và export sang định dạng triển khai.
**Ví dụ CLI (Ultralytics):**

```
yolo pose train model=yolov8x-pose.pt data=pitch-keypoints.yaml imgsz=1024 epochs=200
```

**OpenCV** OpenCV được dùng trong toàn bộ pipeline xử lý media:

- Đọc/ghi ảnh và video, resize/crop, convert color space.
- Các phép tiền xử lý (Gaussian blur, histogram equalization, morphological ops).
- Hiển thị và vẽ kết quả (bounding boxes, keypoints, đường biên sân).

**Vai trò trong dự án:** tiền xử lý input, vẽ overlay kết quả, và xử lý video frame-by-frame khi chạy inference.
**Ví dụ snippet (Python & OpenCV):**

```
import cv2
img = cv2.imread('frame.jpg')
img = cv2.resize(img, (1024,1024))
# vẽ keypoint
cv2.circle(img, (x,y), 3, (0,255,0), -1)
```

## Môi trường thực nghiệm

– **Google Colab**

- **Mục đích:** prototyping nhanh, kiểm tra tham số trên GPU T4/TPU, chia sẻ notebook với nhóm.
- **Ưu điểm:** miễn phí (phiên bản cơ bản), dễ kết nối Google Drive, khởi tạo nhanh.

- **Hạn chế:** phiên làm việc có timeout, GPU không luôn sẵn có, không phù hợp cho huấn luyện dài ngày.

**Lưu ý sử dụng:** mount Google Drive để lưu checkpoint, dùng ngắt/restore checkpoint khi phiên bị kill.

– **Kaggle Notebooks**

- **Mục đích:** huấn luyện dài hơn so với Colab, lưu trữ dataset public/private tiện lợi.
- **Ưu điểm:** GPU miễn phí (một số hạn mức), tích hợp datasets/competitions, thuận tiện cho reproducibility.
- **Hạn chế:** giới hạn RAM và tùy chỉnh môi trường ít linh hoạt hơn; kernels công khai có thể lộ mã.

**Lưu ý sử dụng:** tận dụng storage của Kaggle và caches để tránh tải lại dữ liệu lớn nhiều lần.

– **Lightning AI Studio (PyTorch Lightning)**

- **Mục đích:** huấn luyện quy mô lớn, quản lý experiment, hỗ trợ multi-GPU/distributed training.
- **Ưu điểm:** giảm boilerplate code, dễ tích hợp logging (WandB/TensorBoard), scale lên cluster.
- **Hạn chế:** cần học cú pháp PyTorch Lightning, tài nguyên mở rộng yêu cầu trả phí.

**Vai trò trong dự án:** so sánh nhiều run, training phân tán cho dataset lớn (66k ảnh).

## Công nghệ triển khai ứng dụng

– **FastAPI**

FastAPI là framework backend chính cho ứng dụng web:

- **Ưu điểm:** hiệu năng cao (asynchronous), tự động sinh OpenAPI docs, dễ tích hợp với uvicorn/gunicorn.
- **Vai trò:** làm REST API nhận file upload, gọi inference, trả về JSON và đường dẫn file kết quả.

**Ví dụ endpoint đơn giản:**

```
from fastapi import FastAPI, File, UploadFile
@app.post("/process")
async def process(file: UploadFile = File(...)):
    # lưu file -> gọi inference -> trả kết quả
```

– **React + Vite + TailwindCSS**

- **React:** thư viện frontend để xây dựng UI componentized.
- **Vite:** dev server / build tool nhanh, hỗ trợ HMR (hot module reload).
- **TailwindCSS:** utility-first CSS giúp phát triển giao diện nhanh và nhất quán.

**Vai trò:** frontend upload file, hiển thị ảnh/video đã gán nhãn, bảng thống kê (fps, mAP, confidence).

– **Docker**

- **Mục đích:** đóng gói backend + môi trường inference (PyTorch, Ultralytics) để deploy reproducible.
- **Ưu điểm:** portablity giữa local/server/cloud, dễ quản lý dependency.

**Ví dụ phần cuối file Dockerfile**

```
# Copy requirements first for better caching
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY . .

# Create model directory and copy model files
RUN mkdir -p model/
COPY model/ model/

# Expose port
EXPOSE 8000

# Command to run the application
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

## Tiện ích và công cụ bổ trợ

– **WandB / TensorBoard**

- **Mục đích:** logging, theo dõi metrics (loss, mAP), visualizing training curves và hyperparameter sweeps.
- **Lợi ích:** so sánh nhiều runs, lưu artifact (model weights), dễ chia sẻ kết quả thí nghiệm.

– **GitHub**

- **Vai trò:** quản lý mã nguồn, issues, CI/CD pipeline, và lưu trữ tài liệu (README, hướng dẫn).
- **Gợi ý:** dùng GitHub Actions để tự động hoá unit tests, build Docker image và deploy.

– **Roboflow**

- **Mục đích:** quản lý dataset gán nhãn, chuyển đổi format annotation, augment dataset và xuất ra các bộ dữ liệu sẵn dùng (COCO, YOLO, VOC).
- **Lợi ích:** nhanh chóng chuẩn hoá nhãn keypoints, chia train/val/test và áp dụng augmentation có kiểm soát.

Sự kết hợp giữa **Ultralytics YOLOv8 + PyTorch** cho phần mô hình, **OpenCV** cho xử lý media, các môi trường **Colab/Kaggle/Lightning** cho giai đoạn thực nghiệm, cùng **FastAPI + React + Docker** cho phần triển khai, tạo thành một hệ sinh thái đủ mạnh để phát triển, huấn luyện, tối ưu và triển khai hệ thống nhận diện 32 keypoints trên sân bóng đá một cách thực tế và có thể mở rộng.
***Chú ý:*** Các đoạn mã được sử dụng trong các ví dụ chỉ mang tính chất minh họa công cụ, không phải mã thực được sử dụng cho dự án.

### 2.2.3 Cấu trúc mã nguồn

Mã nguồn của dự án được tổ chức theo hướng module hoá nhằm bảo đảm dễ mở rộng và thuận tiện bảo trì. Cấu trúc tổng thể như sau:

```
project/
 data/                  # Datasets và file cấu hình YAML
 train/                 # Script huấn luyện YOLOv8-pose
 eval/                  # Đánh giá mAP, vẽ biểu đồ loss
 video/                 # Suy luận video + frame skipping + nội suy
 webapp/                # Ứng dụng web (backend + frontend)
 README.md
```

**1. Thư mục `data/`**

Lưu trữ các file dữ liệu và file cấu hình `pitch-keypoints.yaml` mô tả: đường dẫn train/val/test, số lượng keypoints và class đối tượng.

**2. Thư mục `train/`**

Chứa các script huấn luyện mô hình, hỗ trợ thay đổi kích thước ảnh, batch size, số epoch và kiểu dữ liệu FP16/FP32.

**3. Thư mục `eval/`**

Chứa các công cụ đánh giá mô hình như tính mAP, vẽ biểu đồ loss, so sánh phiên bản mô hình.

**4. Thư mục `video/`**

Thực hiện suy luận video với nhiều phương pháp tối ưu hoá: `frame skipping`, `batch inference`, nội suy tuyến tính keypoints và các thuật toán tăng tốc khác.

**5. Thư mục `webapp/`**

Gồm hai phần:

- **Backend (FastAPI)**: nhận file, chạy inference, trả về kết quả.
- **Frontend (React)**: giao diện người dùng, hiển thị ảnh/video đã gán nhãn và thông tin keypoints.

Cách tổ chức này giúp hệ thống rõ ràng, tách biệt frontend–backend, thuận tiện cho quá trình mở rộng và triển khai thực tế.

# Chương 3

# Kết quả & Phân tích

## 3.1 Kết quả định lượng

Sau lần tinh chỉnh cuối cùng trên tập dữ liệu $\sim$ 66k ảnh, mô hình YOLOv8x-pose đạt được các kết quả sau trên 2 tập kiểm thử:

- Mô hình YOLOv8-pose nano đạt được các chỉ số sau:

- Box mAP50: 0.995
- Box mAP50–95: 0.994
- Pose mAP50: 0.955
- Pose mAP50–95: 0.874
- Thời gian suy luận trung bình: **3.0 ms/frame**, gồm khoảng 0.3 ms tiền xử lý và 1.8 ms hậu xử lý

Nhìn chung, mô hình nano cho độ chính xác hộp giới hạn rất cao và tốc độ suy luận nhanh $\implies$ mô hình đặc biệt tối ưu cho các ứng dụng yêu cầu tốc độ cao.

- Mô hình YOLOv8-pose s đạt được các chỉ số sau:

- Box mAP50: 0.995
- Box mAP50–95: 0.986
- Pose mAP50: 0.962
- Pose mAP50–95: 0.874
- Thời gian suy luận trung bình: **4.5 ms/frame**, gồm khoảng 0.2 ms tiền xử lý và 1.6 ms hậu xử lý

So với bản nano, mô hình s cho độ chính xác keypoint cao hơn nhưng tốc độ chậm hơn.

**Đánh giá chung:**

- Cả hai mô hình đều có Box mAP50 rất cao (0.995).
- Mô hình s có Pose mAP50 cao hơn nano.
- Nano có mAP50–95 cao hơn ở phần Box.
- Nano nhanh hơn đáng kể (3.0 ms vs 4.5 ms).

$\Longrightarrow$ Nếu ưu tiên tốc độ: dùng **YOLOv8-pose nano**.
$\Longrightarrow$ Nếu ưu tiên độ chính xác keypoint: dùng **YOLOv8-pose s**.

**Xu hướng đạt độ chính xác tuyệt đối:** Dựa trên kết quả huấn luyện hiện tại, cả hai mô hình YOLOv8n-pose và YOLOv8s-pose đều cho thấy xu hướng tiếp tục cải thiện độ chính xác khi:

- Tăng kích thước tập dữ liệu chú giải chuẩn (đặc biệt ở các tư thế hiếm hoặc góc nhìn khó).
- Tăng số epoch huấn luyện kèm cơ chế early stopping để tránh overfitting.
- Điều chỉnh lại anchor, tăng kích thước ảnh đầu vào hoặc áp dụng mosaic/augment nâng cao.

Nếu tiếp tục mở rộng thêm dữ liệu và tối ưu hyperparameter, mô hình có thể tiến gần mức "độ chính xác tuyệt đối" (mAP50 $\approx$ 1.0) — đặc biệt với YOLOv8s-pose vốn có dung lượng lớn hơn và khả năng học biểu diễn tốt hơn YOLOv8n-pose.

## 3.2 Phân tích ảnh hưởng của dữ liệu và siêu tham số

### 3.2.1 Kích thước dữ liệu huấn luyện

Khi tăng dần số lượng ảnh từ vài trăm lên vài chục nghìn, có thể quan sát:

- Từ 327 $\rightarrow$ 1.418 ảnh: mAP tăng nhanh, mô hình học được các cấu trúc cơ bản của sân bóng nhưng còn kém robust với các góc quay lạ.
- Từ $\sim$ 4k $\rightarrow$ 11k ảnh: mAP tiếp tục cải thiện, đặc biệt giảm sai số với các điểm xa camera.
- Từ 11k $\rightarrow$ 66k ảnh: mAP tăng chậm hơn nhưng mô hình trở nên **ổn định** hơn khi đánh giá trên video thực, ít bị "mất keypoints" khi có nhiễu ánh sáng hoặc khán giả che khuất một phần sân.

### 3.2.2 Kích thước ảnh và số epoch

- **Ảnh 640x640:** tốc độ nhanh, phù hợp cho huấn luyện thăm dò, nhưng đôi khi khó phân giải các đường kẻ mỏng.
- **Ảnh 1024x1024:** là điểm cân bằng tốt giữa **độ chi tiết** và **chi phí tính toán**. Nhiều thí nghiệm cho thấy mAP Pose tăng rõ rệt khi chuyển từ 640 lên 1024.

- **Ảnh 1280x1280:** giúp mô hình học được chi tiết hơn, nhưng thời gian huấn luyện và suy luận tăng mạnh; việc sử dụng kích thước này chỉ phù hợp khi có GPU mạnh và thời gian huấn luyện dài.
- **Số epoch:** khoảng 150–200 epoch đã đủ cho mô hình hội tụ với dữ liệu hiện tại; tăng lên 300 epoch cho thấy lợi ích hạn chế và nguy cơ overfitting nếu không tăng thêm regularization hoặc augmentation.

### 3.2.3 Tối ưu tốc độ suy luận

Các chiến lược tối ưu chính:

- **FP16:** chuyển mô hình sang FP16 giúp tăng tốc độ suy luận, giảm sử dụng bộ nhớ GPU; kết quả so sánh hình ảnh đầu ra cho thấy sai khác rất nhỏ so với FP32.
- **TensorRT:** bước đầu thử nghiệm chuyển mô hình sang TensorRT nhưng thời gian build engine quá dài trong môi trường hạn chế nên chưa thu được lợi ích như mong đợi.
- **Dự đoán thưa trên video:** chỉ chạy mô hình trên 1 trong mỗi 5 frame và **lặp lại** kết quả hoặc **nội suy** cho các frame ở giữa. Cách làm này có thể tăng tốc lên khoảng 5 lần, nhưng chất lượng video đầu ra phụ thuộc mạnh vào chất lượng nội suy quỹ đạo chuyển động của đối tượng.

## 3.3 Thử nghiệm các phương pháp dự đoán trên video

Trong dự án này, chúng tôi tiến hành thử nghiệm bốn phương pháp dự đoán khác nhau trên cùng một tập dữ liệu video có độ dài 30 giây, tương ứng với 900 khung hình. Bốn test case được thiết kế như sau:

1. Sử dụng chuẩn tính toán FP32, đây là phương pháp truyền thống với độ chính xác cao nhưng thường tiêu tốn nhiều tài nguyên hơn
2. Sử dụng FP16, nhằm giảm tải bộ nhớ và tăng tốc độ xử lý, tuy nhiên có thể ảnh hưởng đến độ chính xác trong một số tình huống.
3. Dự đoán trên mỗi 5 khung hình và lặp lại kết quả cho các khung hình trung gian. Cách tiếp cận này giúp giảm số lần suy luận, tiết kiệm thời gian nhưng có nguy cơ làm mất đi sự mượt mà trong chuỗi khung hình
4. Dự đoán với phương pháp nội suy quang lưu giúp khung hình mượt hơn

Trong quá trình đánh giá hiệu năng và độ chính xác của các phương pháp, chúng tôi tiến hành đo lường chi tiết ba giai đoạn chính gồm tiền xử lý, suy luận và hậu xử lý. Kết quả cho thấy mỗi phương pháp có đặc điểm riêng về thời gian thực thi, phản ánh độ phức tạp của thuật toán cũng như khả năng tối ưu hóa trong từng bước.

Từ kết quả so sánh có thể thấy rằng mỗi phương pháp đều có ưu điểm và hạn chế riêng. Trong trường hợp hệ thống cần tốc độ xử lý nhanh và hiệu năng ổn định, FP32 hoặc FP16 là lựa chọn tối ưu. FP16 đặc biệt phù hợp khi muốn tiết kiệm tài nguyên bộ nhớ mà vẫn giữ được hiệu năng

| **Phương pháp** | **Preprocess (ms)** | **Inference (ms)** | **Postprocess (ms)** | **Confidence** |
|---|---|---|---|---|
| FP32 | 2.48 | 12.31 | 1.34 | 0.9671 |
| Nội suy quang lưu | 2.95 | 14.94 | 1.58 | 0.8897 |
| FP16 | 2.55 | 13.63 | 1.35 | 0.9671 |
| Dự đoán thưa (lặp) | 2.63 | 11.41 | 1.28 | 0.9670 |

Bảng 3.1: So sánh hiệu năng và độ tin cậy giữa các phương pháp

gần tương đương FP32. Nếu mục tiêu chỉ là đạt tốc độ cao mà không quá chú trọng đến sự mượt mà của chuỗi khung hình, phương pháp lặp lại kết quả dự đoán trên mỗi 5 khung hình sẽ đáp ứng tốt yêu cầu này. Ngược lại, khi chất lượng hình ảnh và sự mượt mà của chuyển động được đặt lên hàng đầu, nội suy quang lưu trở thành phương án đáng cân nhắc, dù phải chấp nhận chi phí tính toán cao hơn và độ tin cậy trung bình thấp hơn. Như vậy, việc lựa chọn phương pháp nào phụ thuộc trực tiếp vào mục tiêu cụ thể của hệ thống, giữa ưu tiên về tốc độ, tài nguyên hay chất lượng hiển thị.

# Chương 4

# Kết luận

## 4.1 Kết luận & Hướng phát triển

Trong báo cáo này, em đã trình bày quá trình xây dựng và đánh giá một hệ thống nhận diện các điểm đặc trưng trên sân bóng đá sử dụng mô hình YOLOv8-pose. Từ một bộ dữ liệu nhỏ ban đầu, dự án đã mở rộng lên khoảng 66.000 ảnh, áp dụng nhiều chiến lược tăng cường dữ liệu và tinh chỉnh siêu tham số để đạt:

- Box mAP50 $\approx 0.995$, Box mAP50–95 $\approx 0.986$.
- Pose mAP50 $\approx 0.962$, Pose mAP50–95 $\approx 0.874$.
- Thời gian suy luận $\sim 4.5$ ms/frame trên GPU T4.

Các kết quả này chứng minh khả năng áp dụng thực tế của mô hình YOLOv8-pose cho bài toán nhận diện sân bóng và keypoints, đồng thời cho thấy tầm quan trọng của (i) dữ liệu phong phú và sạch, (ii) thiết kế pipeline huấn luyện hợp lý, và (iii) tối ưu tốc độ suy luận.

Trong tương lai, có thể phát triển theo các hướng:

- **Tích hợp tracking và nội suy phi tuyến**: sử dụng optical flow hoặc các mô hình tracking (SORT, DeepSORT, ByteTrack,...) để duy trì ID và quỹ đạo của các điểm, giảm số lần phải chạy mô hình trên mọi frame.
- **Mở rộng đối tượng**: bổ sung detection cho cầu thủ, bóng, trọng tài trong cùng một mô hình, từ đó xây dựng hệ thống phân tích chiến thuật.
- **Triển khai thực tế**: đóng gói mô hình dưới dạng dịch vụ (REST/gRPC) và thử nghiệm trên các server có GPU, hoặc tối ưu thêm với TensorRT khi tài nguyên cho phép.
- **Cải thiện UI/UX**: nâng cấp ứng dụng web hiện có, hỗ trợ cả video, cho phép người dùng tuỳ chọn mô hình, cài đặt ngưỡng confidence, tải xuống kết quả và thống kê chi tiết.

Từ góc độ học phần AI, dự án giúp em củng cố vững chắc các kiến thức về deep learning cho thị giác máy tính, pipeline huấn luyện mô hình, cách đọc và diễn giải các thước đo đánh giá (mAP, FPS), đồng thời rèn luyện kỹ năng triển khai thực tế (từ mô hình đến ứng dụng web demo).

# Tài liệu tham khảo


[1] Ultralytics. *YOLOv8 Documentation*. Truy cập tại: `https://docs.ultralytics.com`

[2] Roboflow. *Camera Calibration and Sports Computer Vision*. Truy cập tại: `https://blog.roboflow.com/camera-calibration-sports-computer-vision/`

[3] A. Bochkovskiy, C.-Y. Wang, H.-Y. M. Liao, *YOLOv4: Optimal Speed and Accuracy of Object Detection*. arXiv:2004.10934, 2020.

[4] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, Y. Sheikh, *OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields*. IEEE TPAMI, 2021.

[5] NVIDIA. *TensorRT Developer Guide*. Truy cập tại: `https://developer.nvidia.com/tensorrt`

[6] PyTorch Team. *PyTorch Documentation*. Truy cập tại: `https://pytorch.org/docs`

[7] FastAPI Community. *FastAPI Documentation*. Truy cập tại: `https://fastapi.tiangolo.com/`

[8] React Team. *React Official Documentation*. Truy cập tại: `https://react.dev`

[9] Vite. *Vite Documentation*. Truy cập tại: `https://vitejs.dev`

[10] TailwindCSS. *TailwindCSS Documentation*. Truy cập tại: `https://tailwindcss.com`

[11] Roboflow. *Dataset Tools and Annotation Documentation*. Truy cập tại: `https://roboflow.com`

# Phụ lục A

# Phụ lục: Hướng dẫn chạy mã nguồn

## A.1 Cấu trúc thư mục (ví dụ)

```
project/
     data/
     train/
     eval/
     video/
     webapp/
     README.md
```

## A.2 Ví dụ lệnh huấn luyện YOLOv8-pose

```
yolo pose train \
    model=yolov8x-pose.pt \
    data=data/pitch-keypoints.yaml \
    imgsz=1024 \
    epochs=200 \
    batch=16 \
    device=0 \
    name=pitch-yolov8x-1024
```

## A.3 Ví dụ lệnh suy luận trên video

```
yolo pose predict \
    model=runs/pose/pitch-yolov8x-1024/weights/best.pt \
    source=video/test_30s.mp4 \
    imgsz=1024 \
    save=True \
    device=0
```

# Phụ lục B

# Triển khai chương trình trên Web Application

Trong phát triển ứng dụng web, Python có nhiều framework hỗ trợ lập trình viên như Django, Flask, Tornado,... Tuy nhiên, nhóm chúng em lựa chọn FastAPI vì framework này có hiệu năng cao nhờ cơ chế bất đồng bộ, cú pháp gọn gàng và dễ sử dụng. FastAPI còn tự động sinh tài liệu API, hỗ trợ tốt cho việc tích hợp với frontend. Đặc biệt, FastAPI rất phù hợp để triển khai các mô hình deep learning vì dễ dàng xử lý dữ liệu ảnh/video và kết nối với các thư viện như PyTorch, TensorFlow. Nhờ đó, nhóm có thể vừa xây dựng giao diện web, vừa triển khai mô hình nhận diện đối tượng một cách nhanh chóng và hiệu quả.

Về phần frontend, nhóm lựa chọn React kết hợp Vite và Tailwind CSS. React là thư viện phổ biến, hỗ trợ xây dựng giao diện theo hướng component hóa, dễ tái sử dụng và bảo trì. Vite mang lại tốc độ phát triển nhanh nhờ cơ chế hot-reload và build tối ưu. Tailwind CSS giúp thiết kế giao diện hiện đại, responsive và dễ tùy biến. Bộ công cụ này không chỉ giúp nhóm phát triển nhanh chóng mà còn đảm bảo hiệu năng và trải nghiệm người dùng tốt khi tích hợp với backend FastAPI.

## B.1 Cấu trúc thư mục

```
sight-and-await/
 backend/
    main.py
    inference.py
    results/
    requirements.txt
    model/.
 frontend/
 README.md
```

## B.2 Quy trình hoạt động

- Trong hệ thống này, quy trình hoạt động được thiết kế xoay quanh việc tích hợp mô hình YOLOv8 vào backend để thực hiện nhiệm vụ nhận diện đối tượng. Người dùng bắt đầu bằng cách truy cập vào giao diện web.Tại đây, người dùng có thể tải lên một ảnh hoặc một video

thông qua form upload. Ngay sau khi file được chọn, frontend sẽ sử dụng phương thức POST để gửi dữ liệu đến API /process của backend, thông qua các thư viện như fetch hoặc axios.

- Backend, được triển khai bằng FastAPI, sẽ tiếp nhận file này và lưu tạm vào thư mục gốc để phục vụ cho quá trình xử lý. Tùy thuộc vào loại file, hệ thống sẽ gọi các hàm process_image() hoặc process_video() nằm trong file inference.py.

- Sau khi quá trình xử lý hoàn tất, backend trả về phản hồi JSON cho frontend. Phản hồi này chứa đường dẫn để hiển thị ảnh hoặc video kết quả, đường dẫn tải xuống, cùng với các thông tin thống kê như số lượng đối tượng, độ tin cậy trung bình, danh sách các đối tượng phát hiện và keypoints. Frontend sẽ sử dụng dữ liệu này để hiển thị trực tiếp ảnh hoặc video đã được gắn nhãn lên giao diện, đồng thời trình bày bảng thống kê chi tiết cho người dùng.

## B.3 Kiểm thử và đánh giá

Trong quá trình phát triển hệ thống, nhóm chúng em đã tiến hành nhiều loại kiểm thử khác nhau để đảm bảo ứng dụng hoạt động ổn định, chính xác và mang lại trải nghiệm tốt cho người dùng. Các kiểm thử này được thiết kế nhằm đánh giá cả về mặt chức năng, hiệu năng, giao diện cũng như khả năng xử lý lỗi.

- Kiểm thử chức năng tập trung vào việc xác minh các tính năng cốt lõi của hệ thống. Người dùng có thể tải lên ảnh hoặc video thông qua giao diện web, và hệ thống phải đảm bảo rằng file được gửi đến backend, mô hình YOLOv8 thực hiện suy luận, sau đó trả về kết quả đúng với dữ liệu đầu vào. Kết quả nhận diện bao gồm bounding boxes, lớp đối tượng, độ tin cậy và keypoints phải được hiển thị chính xác.

- Kiểm thử hiệu năng, nhằm đo lường tốc độ và độ trễ của hệ thống. Nhóm tiến hành đo thời gian xử lý trung bình cho một ảnh và một video, đồng thời ghi nhận độ trễ phản hồi từ khi người dùng gửi file đến khi nhận được kết quả.

- Kiểm thử giao diện là giao diện người dùng phải hiển thị đúng ảnh hoặc video kết quả sau khi xử lý, đồng thời cung cấp bảng thống kê số lượng đối tượng, độ tin cậy trung bình và danh sách các đối tượng được phát hiện

- Kiểm thử lỗi được thực hiện để đánh giá khả năng hệ thống xử lý các tình huống bất thường. Nhóm đã thử tải lên các file sai định dạng (ví dụ: PDF, TXT), các file có dung lượng quá lớn, hoặc trường hợp không có file nào được gửi.

## B.4 Bảo mật và triển khai

- Về bảo mật, backend được xây dựng bằng FastAPI đã tích hợp sẵn CORSmiddleware. Trong mã nguồn, cấu hình hiện tại cho phép tất cả các domain có thể gọi API, điều này thuận tiện cho giai đoạn phát triển và thử nghiệm. Tuy nhiên, khi triển khai thực tế, nhóm có thể giới hạn lại danh sách domain được phép truy cập, ví dụ chỉ cho phép frontend chạy tại http://localhost:5173 hoặc một domain cố định trên server. Việc giới hạn này giúp ngăn chặn các yêu cầu trái phép từ bên ngoài, đảm bảo API chỉ phục vụ đúng đối tượng người dùng.

- Về triển khai, dự án đã chuẩn bị sẵn Dockerfile trong repo.Nhờ đó, nhóm có thể dễ dàng chạy trên bất kỳ server nào mà không cần cài đặt thủ công. Docker mang lại sự nhất quán trong môi trường triển khai: ứng dụng chạy trên máy cá nhân, máy chủ nội bộ hay cloud đều có hành vi giống nhau

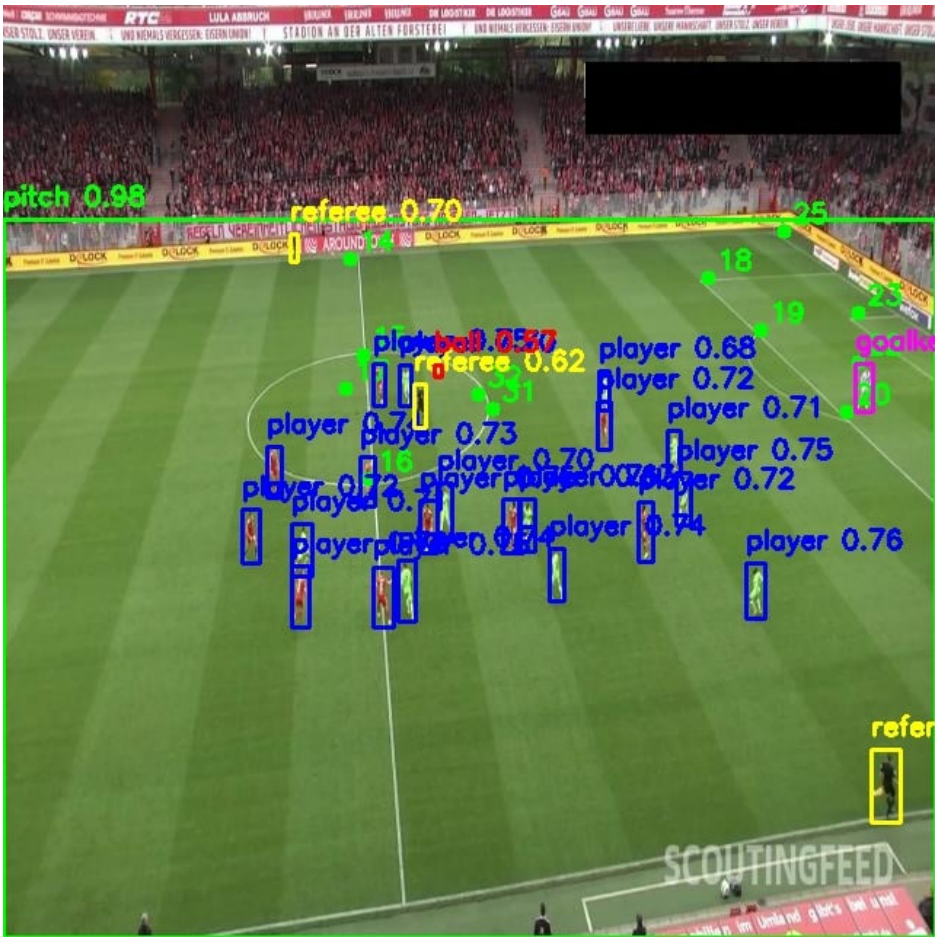
## B.5 Kết quả thực nghiệm

Ứng dụng demo mà nhóm phát triển không chỉ dừng lại ở khả năng dự đoán khung hình hay phân tích mặt sân, mà đã tiến thêm một bước quan trọng để trở thành một nền tảng phân tích video thể thao thông minh. Bằng việc tích hợp thêm mô-đun nhận diện cầu thủ, trọng tài và các đối tượng liên quan trên sân, hệ thống có thể tự động phân biệt vai trò, theo dõi chuyển động và đưa ra thông tin chi tiết về từng nhân vật trong trận đấu.

Với đầu vào là một ảnh chụp trong một trận bóng đá


Hình B.1: Ảnh sân bóng đầu vào

Sau khi đưa vào app ta được kết quả như sau:




Hình B.2: Ảnh sân bóng đầu ra


Hình B.3: Kết quả phân tích