Training FPT Hackathon buổi 10

Tổng quan Quy trình

Chúng ta sẽ chia dự án thành 4 giai đoạn chính:

1. **Giai đoạn 1: Chuẩn bị Dữ liệu (Foundation)** - Bước quan trọng nhất, quyết định chất lượng của model.
2. **Giai đoạn 2: Huấn luyện Mô hình (Training)** - Dạy cho máy tính cách nhận biết các biển báo từ dữ liệu đã chuẩn bị.
3. **Giai đoạn 3: Kiểm thử và Tinh chỉnh (Validation)** - Đánh giá và đảm bảo model hoạt động tốt.
4. **Giai đoạn 4: Triển khai (Deployment)** - Xây dựng một hệ thống Client-Server thực tế để sử dụng model.

Giai đoạn 1: Chuẩn bị Dữ liệu

Mục tiêu: Tạo ra một bộ dữ liệu (dataset) chất lượng cao chứa các hình ảnh và thông tin vị trí của các biển báo.

Bước 1.1: Lựa chọn Mô hình và Hiểu về Yêu cầu Dữ liệu

- **Lựa chọn mô hình:** Chúng ta chọn **YOLO (You Only Look Once)**, cụ thể là phiên bản **YOLOv8**.
    - **Tại sao là YOLO?** Vì bài toán của chúng ta là **Nhận diện vật thể (Object Detection)** - tức là vừa phải xác định **vật thể là gì** (ví dụ: `re_phai`), vừa phải cho biết **nó ở đâu** trong ảnh (bằng một hộp bao - bounding box). Các mô hình như MobileNetV2 chỉ làm nhiệm vụ phân loại (classification), tức là chỉ trả lời được câu hỏi "cả bức ảnh này chứa gì?", không phù hợp với bài toán này. YOLOv8 là một mô hình hiện đại, cân bằng tốt giữa tốc độ và độ chính xác, và rất dễ sử dụng.
- **Yêu cầu dữ liệu của YOLO:** Để huấn luyện YOLO, mỗi ảnh (`image.jpg`) cần có một file văn bản (`image.txt`) đi kèm. File `.txt` này chứa thông tin về tất cả các vật thể trong ảnh, mỗi vật thể một dòng, theo định dạng:
  `<class_id> <x_center> <y_center> <width> <height>`
    - Tất cả các giá trị tọa độ và kích thước đều được **chuẩn hóa** (giá trị từ 0 đến 1) bằng cách chia cho chiều rộng và chiều cao của ảnh.

Bước 1.2: Thu thập Hình ảnh

- **Mục tiêu:** Thu thập càng nhiều hình ảnh đa dạng chứa 6 loại biển báo: `rẽ phải`, `rẽ trái`, `đi thẳng`, `cấm rẽ phải`, `cấm rẽ trái`, `cấm đi thẳng`.
- **Nguyên tắc "Vàng":** Đa dạng là chìa khóa! Cần ảnh ở nhiều điều kiện:
    - **Ánh sáng:** Ngày, đêm, bình minh, hoàng hôn, trời u ám.
    - **Thời tiết:** Nắng, mưa, sương mù.
    - **Góc chụp:** Chụp thẳng, chụp chéo, từ dưới lên, từ trên xuống.
    - **Tình trạng:** Biển báo mới, cũ, bị mờ, bị che khuất một phần.
- **Nguồn:** Tự chụp, tìm kiếm trên Google Images, tải từ các bộ dữ liệu công khai (Kaggle, Roboflow Universe).

Bước 1.3: Gán nhãn (Annotation)

- **Mục tiêu:** Vẽ các hộp bao quanh biển báo và gán tên lớp cho chúng.
- **Công cụ:** Sử dụng **LabelImg**. Đây là công cụ offline, miễn phí và tạo ra định dạng YOLO một cách trực tiếp.
- **Quy trình với LabelImg:**
    1. Cài đặt: `pip install labelimg`.
    2. Mở LabelImg, chọn thư mục chứa ảnh và thư mục để lưu nhãn.
    3. **Quan trọng:** Chuyển định dạng lưu sang **YOLO**.
    4. Vẽ hộp bao (bounding box) thật khít quanh từng biển báo.
    5. Nhập tên lớp (class name). Hãy thống nhất tên không dấu, không khoảng trắng, ví dụ: `re_phai`, `cam_re_trai`.
    6. Lưu lại, LabelImg sẽ tự động tạo file `.txt` tương ứng.

Tổng quan quy trình:

1. **Gán nhãn** bằng LabelImg.
2. **Tăng cường dữ liệu** bằng một kịch bản (script) Python. Đây là bước mà LabelImg không tự làm được.
3. **Tổ chức và phân chia** dữ liệu đã tăng cường vào các thư mục `train` và `val`.

1.1. Chuẩn bị thư mục

Tạo một cấu trúc thư mục đơn giản để bắt đầu:

```
project/
├── original_images/     # Đặt 24 ảnh gốc của bạn vào đây
└── original_labels/     # LabelImg sẽ lưu file .txt vào đây
```

1.3. Gán nhãn

1. **Mở LabelImg:** Gõ `labelimg` trong terminal của bạn.
2. **Chọn Thư mục ảnh:** Nhấn nút "Open Dir" và chọn thư mục `original_images`.
3. **Chọn Thư mục lưu nhãn:** Nhấn nút "Change Save Dir" và chọn thư mục `original_labels`.
4. **CHỌN ĐỊNH DẠNG YOLO:** Đây là bước quan trọng nhất! Ở thanh công cụ bên trái, hãy đảm bảo bạn đã chọn định dạng là **"YOLO"**. Nếu đang là "PascalVOC", hãy nhấp vào để chuyển đổi.
5. **Bắt đầu vẽ:**
    - Nhấn phím `W` (hoặc nút "Create RectBox") để bắt đầu vẽ.
    - Kéo chuột để vẽ một hộp bao vừa khít quanh biển báo.
    - Khi bạn thả chuột, một hộp thoại sẽ hiện ra. Nhập tên lớp (ví dụ: `re_phai`, `cam_re_trai`). Nhấn OK.
    - **Lưu ý:** Lần đầu tiên bạn nhập một tên lớp mới, nó sẽ được thêm vào danh sách.
6. **Lưu lại:** Nhấn `Ctrl + S` để lưu file nhãn `.txt`. LabelImg sẽ tự động tạo file có tên tương ứng với file ảnh trong thư mục `original_labels`.
7. **Lặp lại:** Nhấn phím `D` để qua ảnh tiếp theo và lặp lại quá trình cho đến hết 24 ảnh.

Sau khi hoàn thành, trong thư mục `original_labels` sẽ có 24 file `.txt`. LabelImg cũng sẽ tạo một file `classes.txt` liệt kê tất cả các lớp của bạn. Hãy giữ file này, nó rất hữu ích.

Bước 1.4: Tăng cường và Phân chia Dữ liệu

- **Vấn đề:** Số lượng ảnh gốc thường không đủ để model học tốt.
- **Giải pháp: Tăng cường dữ liệu (Data Augmentation)** - tạo ra các phiên bản ảnh mới từ ảnh gốc bằng cách áp dụng các phép biến đổi.
- **Quy trình:**
    1. Sử dụng một script Python với thư viện `albumentations` để tự động hóa.
    2. **Các phép tăng cường phù hợp:**
        - **Geometric:** Xoay (Rotation), Cắt cúp (Crop), Biến dạng (Shear) -> Mô phỏng các góc nhìn khác nhau.

- **Color/Brightness:** Thay đổi độ sáng, độ tương phản -> Mô phỏng các điề̀u kiện ánh sáng khác nhau.
- **Noise/Blur:** Thêm nhiễu, làm mờ -> Mô phỏng camera chấ́t lượng thấ́p hoặc chuyển động.

3. **Phân chia:** Sau khi có một bộ dữ liệu lớn (gồ̀m cả ảnh gố́c và ảnh tăng cường), dùng một script Python khác để chia chúng ra thành 2 tập:
    - **Tập huấ́n luyện (train):** Khoảng 80% dữ liệu, dùng để "dạy" model.
    - **Tập kiểm định (validation/val):** Khoảng 20% dữ liệu, dùng để đánh giá hiệu năng của model trong quá trình huấ́n luyện mà không "gian lận" (model chưa từng thấ́y dữ liệu này).
4. Tổ chức vào cấ́u trúc thư mục chuẩn của YOLO.

- **Chi tiế́t**

Bước 1: Cài đặt thư viện cần thiết

```
pip install albumentations opencv-python
```

Bước 2: Tăng cường Dữ liệu bằng Script Python

Chúng ta sẽ dùng thư viện `albumentations`, một thư viện cực kỳ mạnh mẽ cho việc tăng cường dữ liệu.

2.2. Viết Script tăng cường

Tạo file Python `augment_data.py`

2.3. Chạy Script

1. Đảm bảo bạn đã ở trong thư mục `project/`.
2. Chạy lệnh: `python augment_data.py`

Sau khi chạy xong, bạn sẽ có hai thư mục mới là `augmented_images` và `augmented_labels`, chứa hàng trăm cặp file ảnh và nhãn đã được tăng cường.

Bước 3: Tổ chức và Phân chia Dữ liệu

Bây giờ bạn cần tạo cấu trúc thư mục cuối cùng cho YOLO và chia dữ liệu ra.

1. **Tạo cấu trúc thư mục cuối cùng:**

```
dataset/
├── images/
│   ├── train/
│   └── val/
└── labels/
    ├── train/
    └── val/
```

2. **Trộn và chia dữ liệu:**

   - Bạn sẽ gộp cả ảnh/nhãn **gốc** và ảnh/nhãn **đã tăng cường** lại với nhau.
   - **Ví dụ:** Bạn có 24 ảnh gốc + 24 * 10 = 240 ảnh tăng cường = **264 ảnh tổng cộng**.
   - **Chia theo tỷ lệ 80/20:**
     - Tập Train (80%): Khoảng 211 ảnh.
     - Tập Validation (20%): Khoảng 53 ảnh.
   - **Cách làm:**
     - Di chuyển (Move) 211 cặp file (ảnh và nhãn tương ứng) từ `original_images`, `original_labels`, `augmented_images`, `augmented_labels` vào các thư mục `dataset/images/train` và `dataset/labels/train`.
     - Di chuyển 53 cặp file còn lại vào các thư mục `dataset/images/val` và `dataset/labels/val`.

- **Mẹo:** Để đảm bảo tính ngẫu nhiên, chúng ta nên viết một script nhỏ để tự động chia file hoặc đơn giản là chọn thủ công một cách tương đối (split_data.py).

Script này sẽ:

1. Đọc tất cả các file ảnh từ một thư mục nguồn.

2. Xáo trộn (shuffle) danh sách các file một cách ngẫu nhiên.
3. Chia danh sách đó thành hai phần: training và validation theo tỷ lệ bạn chỉ định.
4. Di chuyển các file ảnh và file nhãn `.txt` tương ứng vào đúng cấu trúc thư mục `train` và `val` mà YOLO yêu cầu.

Chuẩn bị

1. **Cấu trúc thư mục ban đầu:**
   Script này giả định rằng bạn đã có tất cả các ảnh và nhãn (cả gốc và đã tăng cường) nằm chung trong hai thư mục.

```
project/
├── all_images/      # Chứa TẤT CẢ các file ảnh (.jpg, .png...)
├── all_labels/      # Chứa TẤT CẢ các file nhãn (.txt)
├── split_data.py    # Script chúng ta sẽ tạo
└── dataset/         # Thư mục này sẽ được script tự động điền vào
```

   - **Quan trọng:** Trước khi chạy script, hãy di chuyển tất cả các file từ `original_images`, `augmented_images` vào `all_images`. Tương tự, di chuyển tất cả file từ `original_labels`, `augmented_labels` vào `all_labels`.

2. **Tạo script `split_data.py`:**
   Tạo một file Python tên là `split_data.py` trong thư mục gốc `project/` và dán nội dung sau vào.

Cách sử dụng

1. **Chuẩn bị:** Đảm bảo bạn đã tạo và đặt các file/thư mục đúng như cấu trúc ở phần "Chuẩn bị" phía trên.
2. **Chạy Script:** Mở terminal hoặc command prompt, điều hướng đến thư mục `project/` và chạy lệnh:

```
python split_data.py
```

3. **Kiểm tra kết quả:** Script sẽ chạy và in ra tiến trình. Sau khi hoàn tất, thư mục `dataset/` của bạn sẽ có đầy đủ các file ảnh và nhãn đã được phân chia vào `train` và `val`. Đồng thời, các thư mục `all_images` và `all_labels` sẽ trống vì các file đã được *di chuyển* (`shutil.move`).

Giai đoạn 2: Huấn luyện Mô hình

Mục tiêu: Sử dụng bộ dữ liệu đã chuẩn bị để tạo ra một file model (`.pt`) có khả năng nhận diện biển báo.

Bước 2.1: Cài đặt YOLOv8

- Tạo một môi trường ảo (ví dụ với Anaconda: `conda create -n yolo_env python=3.9`) và kích hoạt nó.
- Cài đặt thư viện `ultralytics` :

```
pip install ultralytics
```

Bước 2.2: Tạo file Cấu hình Dataset (`.yaml`)

- Tạo một file, ví dụ `traffic_signs.yaml` . File này là "bản đồ" chỉ cho YOLO biết dữ liệu của bạn ở đâu và gồm những lớp nào.

```
# Đường dẫn đến thư mục ảnh train và val
train: /path/to/your/dataset/images/train/
val: /path/to/your/dataset/images/val/

# Số lượng lớp
nc: 6

# Tên các lớp (thứ tự PHẢI khớp với class_id)
names:
  - 're_phai'       # class_id = 0
  - 're_trai'       # class_id = 1
  - 'di_thang'      # class_id = 2
  - 'cam_re_phai'   # class_id = 3
  - 'cam_re_trai'   # class_id = 4
  - 'cam_di_thang'  # class_id = 5
```

Lưu ý: Thứ tự trong `names` phải **khớp chính xác** với thứ tự trong file `classes.txt` mà LabelImg đã tạo ra.

Bước 2.3: Chạy lệnh Huấn luyện

- Mở terminal, kích hoạt môi trường và chạy lệnh:

```
yolo detect train data=traffic_signs.yaml model=yolov8s.pt epochs=100 imgsz=640
```

    - `data` : Trỏ đến file `.yaml` của bạn.
    - `model=yolov8s.pt` : Sử dụng **học chuyển giao (transfer learning)**. Chúng ta không dạy model từ đầu mà bắt đầu từ một model `yolov8s` đã rất thông minh (được huấn luyện trên bộ dữ liệu COCO khổng lồ). Điều này giúp model học nhanh hơn và chính xác hơn rất nhiều.
    - `epochs=100` : Số lần model "ôn bài" trên toàn bộ dữ liệu training.
    - `imgsz=640` : Kích thước ảnh đầu vào.

Giai đoạn 3: Kiểm thử và Tinh chỉnh

Mục tiêu: Đảm bảo model hoạt động như mong đợi trước khi triển khai.

- **Kết quả training:** Sau khi huấn luyện xong, mọi thứ sẽ được lưu trong thư mục `runs/detect/train/` .
- **File model tốt nhất:** Model bạn sẽ sử dụng nằm ở `runs/detect/train/weights/best.pt` .
- **Kiểm tra nhanh:** Dùng lệnh `yolo predict` để test trên ảnh, video hoặc webcam mới:

```
# Test trên ảnh
yolo detect predict model=path/to/best.pt source=my_test_image.jpg

# Test trên video
yolo detect predict model=path/to/best.pt source=my_test_video.mp4
```

- **Đánh giá:** Xem các biểu đồ trong thư mục kết quả (ví dụ: `confusion_matrix.png` , `results.png`) để biết model có hay nhầm lẫn giữa các lớp không và độ chính xác tổng thể (mAP) là bao nhiêu.
- **Vòng lặp cải tiến:** Nếu kết quả chưa tốt, hãy quay lại Giai đoạn 1: thu thập thêm dữ liệu cho những trường hợp model hay nhận diện sai, gán nhãn, và huấn luyện lại.

Giai đoạn 4: Triển khai với Kiến trúc Client-Server (MQTT)

Mục tiêu: Xây dựng một hệ thống thực tế nơi một thiết bị client (có thể yếu) gửi dữ liệu đến một server mạnh để nhận diện.

Đầu tiên là chọn sử dụng Rest Api (HTTP) hay MQTT?

Bảng so sánh trực quan: REST API (HTTP) vs. MQTT

Tiêu chí	REST API (sử dụng HTTP)	MQTT (Publish/Subscribe)
Mô hình giao tiếp	**Request-Response (Hỏi-Đáp):** Client phải chủ động gửi yêu cầu, server xử lý và trả về phản hồi. Kết nối được đóng sau mỗi lần hỏi-đáp.	**Publish-Subscribe (Phát hành-Đăng ký):** Client và Server đều kết nối đến một "bưu điện" trung gian (Broker). Giao tiếp gián tiếp qua các "chủ đề" (topic).
Hiệu quả kết nối	**Kém hiệu quả cho dữ liệu liên tục:** Mỗi request (kể cả cho một khung hình video) đều mang theo phần "mào đầu" (header) của HTTP, gây ra overhead lớn.	**Rất hiệu quả:** Client và Server chỉ cần thiết lập kết nối một lần và giữ nó mở. Dữ liệu truyền đi rất nhẹ.
Độ phức tạp	**Đơn giản hơn để bắt đầu:** Không cần thành phần trung gian. Các framework như Flask, FastAPI rất phổ biến và dễ học.	**Phức tạp hơn một chút:** Cần cài đặt và chạy một MQTT Broker (ví dụ: Mosquitto) làm thành phần thứ ba.
Độ trễ	**Cao hơn:** Thời gian thiết lập kết nối cho mỗi request làm tăng độ trễ tổng thể, đặc biệt rõ rệt với video.	**Rất thấp:** Được thiết kế cho các ứng dụng thời gian thực và IoT, độ trễ gần như chỉ là thời gian truyền dữ liệu qua mạng.
Tính linh hoạt	**Rất linh hoạt cho web:** Dễ dàng tích hợp với trình duyệt, ứng dụng di động, và các dịch vụ web khác.	**Rất linh hoạt cho IoT và hệ thống phân tán:** Client và Server không cần biết về sự tồn tại của nhau, dễ dàng thêm/bớt các thành phần.

Vậy, khi nào bạn NÊN dùng HTTP?

HTTP (thông qua một REST API) sẽ là lựa chọn **TỐT HƠN** MQTT trong các trường hợp sau:

1. **Ứng dụng của bạn CHỈ xử lý từng ảnh đơn lẻ, không có video:**
    - **Ví dụ:** Bạn xây dựng một trang web cho phép người dùng tải lên một bức ảnh để nhận diện. Người dùng nhấn nút "Upload", client gửi một yêu cầu POST duy nhất, server xử lý và trả về kết quả JSON.
    - **Lợi ích:** Cực kỳ đơn giản để triển khai với Flask, không cần cài đặt Broker.
2. **Cần tích hợp vào một hệ thống Web hoặc API có sẵn:**

- **Ví dụ:** Công ty của bạn đã có một hệ thống API lớn. Việc thêm một endpoint mới `/api/v1/detect_sign` sẽ tự nhiên và dễ dàng hơn là giới thiệu một giao thức hoàn toàn mới như MQTT.

3. **Ưu tiên sự đơn giản tuyệt đối và không yêu cầu hiệu năng thời gian thực:**
    - Nếu mục tiêu của bạn chỉ là một "proof-of-concept" nhanh chóng cho việc xử lý ảnh, việc dựng một server Flask trong 5 phút sẽ nhanh hơn việc cài đặt và cấu hình Mosquitto.

Tại sao chúng ta đã chọn MQTT cho hệ thống này?

Lý do chúng ta đã đi theo con đường MQTT cho hệ thống tổng hợp cuối cùng là vì nó giải quyết được những điểm yếu của HTTP, đặc biệt là khi xử lý video:

1. **Xử lý Video Stream hiệu quả:** Đây là lý do lớn nhất. Hãy tưởng tượng bạn gửi 20 khung hình mỗi giây qua HTTP. Điều đó có nghĩa là 20 lần thiết lập kết nối TCP, gửi HTTP request header, nhận response, rồi lại đóng kết nối. Nó cực kỳ lãng phí và chậm chạp. Với MQTT, bạn chỉ cần một kết nối duy nhất và gửi đi 20 gói dữ liệu nhỏ một cách nhanh chóng.
2. **Giao tiếp bất đồng bộ:** Client chỉ cần "bắn" các khung hình lên Broker mà không cần chờ server trả lời cho từng cái. Điều này giúp client hoạt động mượt mà, không bị "đứng hình" chờ server.
3. **Khả năng mở rộng:** Với MQTT, bạn có thể dễ dàng cho nhiều server cùng lắng nghe trên topic `yolo/detect/request` để cân bằng tải. Hoặc bạn có thể có hàng trăm client (camera) cùng gửi dữ liệu lên mà không cần thay đổi gì ở phía server.

MQTT (Message Queuing Telemetry Transport) là một giao thức Publish/Subscribe rất nhẹ và nhanh. Nó hoàn hảo cho việc truyền dữ liệu ảnh/video liên tục, đặc biệt là video stream, hiệu quả hơn nhiều so với REST API (HTTP) cho tác vụ này.

Bước 4.1: Thiết kế Kiến trúc

1. **MQTT Broker:** Một "bưu điện" trung gian. Chúng ta dùng **Mosquitto**. Client và Server đều kết nối đến đây.
2. **Server (Publisher/Subscriber):**
    - Tải model `best.pt` một lần duy nhất.
    - **Subscribe (Lắng nghe)** trên một topic chung, ví dụ: `yolo/detect/request`.
    - Khi nhận được yêu cầu, nó sẽ xử lý và **Publish (Gửi)** kết quả về một topic riêng của client đó.
3. **Client (Publisher/Subscriber):**

- Tạo một ID duy nhất cho chính nó.
- **Subscribe** vào topic phản hồi dành riêng cho mình, ví dụ: `yolo/detect/response/CLIENT_ID`.
- **Publish** dữ liệu ảnh/video (đã được mã hóa base64) lên topic `yolo/detect/request`.
- Nhận kết quả và hiển thị cho người dùng.

Bước 4.2: Triển khai Mã nguồn

- **Server (`server_mqtt.py`):**
  - **Nhiệm vụ chính:** Nhận ảnh, chạy `model(img)`, trích xuất kết quả (tọa độ, tên lớp, độ tin cậy) và gửi về dưới dạng **JSON**. Việc server luôn trả về JSON giúp hệ thống linh hoạt, client có thể quyết định cách hiển thị.
- **Client (`client_mqtt.py`):**
  - **Nhiệm vụ chính:**
    - Sử dụng `argparse` để người dùng có thể chọn chế độ (`--mode image` hoặc `--mode video`) và nguồn (`--source ...`).
    - Đảm bảo kết nối đến MQTT Broker thành công **trước khi** gửi yêu cầu đầu tiên để tránh mất tin nhắn.
    - **Nếu là ảnh:** Gửi ảnh, nhận JSON, in thông tin ra terminal, vẽ các hộp bao lên ảnh và dùng `cv2.waitKey(0)` để giữ cửa sổ mở.
    - **Nếu là video:** Bắt đầu một vòng lặp, đọc từng khung hình, gửi lên server, nhận JSON, tự vẽ kết quả lên khung hình, và hiển thị video. Đồng thời, in thông tin ra terminal một cách có kiểm soát (ví dụ: 1 giây/lần) để không làm ngập log.

Bước 4.3: Chạy Hệ thống

1. **Khởi động MQTT Broker.**
2. **Chạy Server:** Mở terminal, kích hoạt môi trường, chạy `python server_mqtt.py`. Server sẽ tải model và bắt đầu lắng nghe.
3. **Chạy Client:** Mở một terminal khác, kích hoạt môi trường và chạy client với các tham số mong muốn:
   - `python client_mqtt.py --mode image --source bien_bao_re_phai.jpg`
   - `python client_mqtt.py --mode video --source video_giao_thong.mp4`
   - `python client_mqtt.py --mode video --source 0` (cho webcam)

Lưu ý khi triển khai với Raspberry Pi

---

1. Tối ưu hóa Hiệu năng (Performance)

Raspberry Pi có tài nguyên CPU và RAM hạn chế, vì vậy mỗi mili giây đều quý giá.
Ví dụ: Khi cài opencv trên pi thì nên cài qua trình quản lý gói như apt của hệ điều hành chứ không nên cài qua pip vì pi chạy kiến trúc arm nên cài qua trình quản lý gói của hệ điều hành sẽ tương thích tốt hơn, việc cài đặt cũng sẽ nhanh hơn.

1.1. Giảm Tải cho CPU của Pi

- **Giảm Frame Rate (FPS) gửi đi:** Đây là yếu tố quan trọng nhất. Webcam có thể cung cấp 30 FPS, nhưng server của bạn có thể không xử lý kịp, và mạng cũng có thể bị nghẽn.
    - Trong file `unified_client.py`, bạn đã có dòng `frame_interval = 1 / 20` (tương đương 20 FPS). Hãy thử giảm nó xuống nữa, ví dụ: `1 / 15` (15 FPS) hoặc thậm chí `1 / 10` (10 FPS). Điều này giảm đáng kể gánh nặng cho cả CPU (đọc và nén khung hình) và mạng của Pi.
- **Giảm Chất lượng Nén JPEG:**
    - Trong dòng `cv2.imencode('.jpg', frame, [cv2.IMWRITE_JPEG_QUALITY, 85])`, con số `85` là chất lượng nén. Giảm nó xuống `75` hoặc `70` sẽ tạo ra các gói tin nhỏ hơn, gửi đi nhanh hơn và giảm tải cho mạng, nhưng sẽ làm ảnh mờ đi một chút. Hãy tìm sự cân bằng phù hợp.
- **Giảm Độ phân giải Khung hình:** Nếu có thể, hãy cấu hình camera của Pi để xuất ra video ở độ phân giải thấp hơn ngay từ đầu (ví dụ: 640x480 thay vì 1280x720). Điều này giảm tải cho mọi bước sau đó.

```
cap = cv2.VideoCapture(source)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
```

1.2. Tránh các tác vụ nặng trên Pi

- **Client chỉ gửi, không xử lý nhiều:** Đảm bảo rằng client của bạn chỉ làm các công việc nhẹ: đọc frame, nén, gửi, nhận kết quả và hiển thị. Toàn bộ logic phức tạp phải nằm ở phía server. Thiết kế hiện tại của bạn đã tuân thủ rất tốt nguyên tắc này.

---

2. Độ tin cậy và Ổn định (Reliability & Stability)

Hệ thống cần phải chạy trong thời gian dài mà không bị treo hoặc mất kết nối.

2.1. Quản lý Kết nối Mạng

- **Kết nối có dây (Ethernet) > WiFi:** Nếu có thể, hãy cắm dây mạng LAN cho Raspberry Pi. Kết nối có dây luôn ổn định hơn và có độ trễ thấp hơn WiFi, đặc biệt quan trọng cho việc stream video.
- **Tự động kết nối lại MQTT:** Thư viện Paho-MQTT có cơ chế tự động kết nối lại nhưng đôi khi cần cấu hình thêm. Code hiện tại khá cơ bản. Trong một ứng dụng sản phẩm, bạn sẽ cần thêm logic trong callback `on_disconnect` để cố gắng kết nối lại sau một khoảng thời gian.
- **Heartbeat/Keepalive:** Trong lệnh `client.connect()`, tham số cuối cùng (mặc định là 60) là thời gian "keepalive". Nó đảm bảo client và broker thường xuyên kiểm tra xem kết nối có còn sống không. `60` là một giá trị tốt.

2.2. Xử lý Lỗi (Error Handling)

- **Kiểm tra `cap.read()`:** Code của bạn đã có `if not success: break`, điều này rất tốt. Nó xử lý trường hợp camera bị ngắt kết nối.
- **Sử dụng `try...except`:** Bọc các khối lệnh có thể phát sinh lỗi (như kết nối mạng, xử lý JSON) trong khối `try...except` để chương trình không bị sập hoàn toàn khi có lỗi xảy ra, mà có thể ghi log và cố gắng tiếp tục.

---

3. Tự động hóa (Automation)

Bạn sẽ không muốn mỗi lần mất điện lại phải cắm màn hình, bàn phím vào Pi để chạy lại script.

3.1. Chạy Script khi Khởi động

- **Mục tiêu:** Tự động chạy file `client_mqtt.py` mỗi khi Raspberry Pi khởi động.
- **Cách thực hiện:** Sử dụng `systemd`, là trình quản lý dịch vụ tiêu chuẩn trên hầu hết các hệ điều hành Linux.

1. Tạo một file service: `sudo nano /etc/systemd/system/yolo_client.service`

2. Dán nội dung sau vào (nhớ sửa lại đường dẫn và tên người dùng cho đúng):

```
[Unit]
Description=YOLO MQTT Client Service
After=network.target
```

```
[Service]
ExecStart=/home/pi/yolo_client_project/venv/bin/python /home/pi/yolo_client
_project/unified_client.py --mode video --source 0
WorkingDirectory=/home/pi/yolo_client_project
StandardOutput=inherit
StandardError=inherit
Restart=always
User=pi

[Install]
WantedBy=multi-user.target
```

3. Lưu và đóng file.

4. Kích hoạt và khởi động service:

```
sudo systemctl enable yolo_client.service
sudo systemctl start yolo_client.service
```

5. Để xem log của dịch vụ: `sudo journalctl -u yolo_client.service -f`

script này sẽ chạy như một dịch vụ nền và tự khởi động lại nếu bị lỗi hoặc khi Pi khởi động.

Bảng tóm tắt các lưu ý

Lĩnh vực	Lưu ý quan trọng	Hành động đề xuất
Hiệu năng	Giảm tải CPU và mạng	Giảm FPS, giảm chất lượng JPEG, giảm độ phân giải.
Độ tin cậy	Ổn định kết nối	Ưu tiên mạng dây, đảm bảo nguồn điện đủ mạnh.
Tự động hóa	Chạy khi khởi động	Tạo một service `systemd` để tự động hóa việc chạy script.