**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT THÀNH PHỐ HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN MÔN HỌC**

**PHÂN TÍCH TÍN HIỆU PPG ĐỂ XÂY DỰNG MÔ HÌNH AI TẠO SINH DỮ LIỆU VÀ DỰ ĐOÁN**

**MÔN HỌC:** Trí tuệ Nhân tạo cho IOT

**NGƯỜI THỰC HIỆN:** 22110400 – Nguyễn Hoàng Phúc

Hồ Chí Minh, ngày 29 tháng 4 năm 2025

**MỤC LỤC**

[GIỚI THIỆU 1](#_Toc196813552)

[Chương 1. DỮ LIỆU SỬ DỤNG 2](#_Toc196813553)

[1.1. Giới thiệu bộ dữ liệu BIDMC 2](#_Toc196813554)

[1.2. Giới thiệu bộ dữ liệu CAPNO-IEEE 2](#_Toc196813555)

[1.3. So sánh và đánh giá ban đầu hai bộ dữ liệu 2](#_Toc196813556)

[Chương 2. TIỀN XỬ LÝ DỮ LIỆU 3](#_Toc196813557)

[2.1. Tiền xử lý riêng lẻ cho từng bộ 3](#_Toc196813558)

[2.2. Chuẩn hóa và đồng bộ dữ liệu 3](#_Toc196813559)

[2.3. Gộp hai bộ dữ liệu thành một tập thống nhất 3](#_Toc196813560)

[2.4. Các kỹ thuật làm sạch và lọc nhiễu áp dụng 3](#_Toc196813561)

[Chương 3. MÔ HÌNH SINH DỮ LIỆU 4](#_Toc196813562)

[3.1. Lý thuyết cơ bản về CVAE 4](#_Toc196813563)

[3.2. Kiến trúc mô hình CVAE áp dụng 4](#_Toc196813564)

[3.3. Chi tiết huấn luyện mô hình 4](#_Toc196813565)

[Chương 4. KẾT QUẢ THỰC NGHIỆM 4](#_Toc196813566)

[4.1. Đánh giá tín hiệu PPG sinh ra 4](#_Toc196813567)

[4.2. Các chỉ số đánh giá 4](#_Toc196813568)

[Chương 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 5](#_Toc196813569)

[Chương 6. PHỤ LỤC 6](#_Toc196813570)

[TÀI LIỆU THAM KHẢO 7](#_Toc196813571)

# Chương 1. GIỚI THIỆU

Tín hiệu Photoplethysmography (PPG) là một dạng tín hiệu sinh lý thu được bằng phương pháp quang học, phản ánh sự thay đổi thể tích máu trong mô ngoại vi theo thời gian. Với ưu điểm là chi phí thu thập thấp, không xâm lấn và dễ triển khai, tín hiệu PPG đã trở thành một trong những công cụ phổ biến nhất trong theo dõi sức khỏe, đặc biệt trong các thiết bị đeo thông minh như đồng hồ đo nhịp tim, vòng tay theo dõi sức khỏe và các hệ thống chăm sóc y tế từ xa.

## Ứng dụng

Trong nghiên cứu và thực tiễn, dữ liệu PPG tổng hợp đóng vai trò rất quan trọng. Các mô hình sinh dữ liệu có khả năng tạo ra tín hiệu PPG mới từ dữ liệu thật giúp tăng cường tập dữ liệu huấn luyện (data augmentation), hỗ trợ trong huấn luyện các mô hình học máy (machine learning) và học sâu (deep learning), đồng thời giảm nhu cầu thu thập dữ liệu thực tế vốn tốn kém và gặp nhiều khó khăn về mặt đạo đức, pháp lý. Ngoài ra, việc sinh dữ liệu còn cho phép kiểm thử các thuật toán phân tích tín hiệu trong điều kiện đa dạng hơn và phát triển các hệ thống chẩn đoán không xâm lấn với độ tin cậy cao hơn.

## Mục tiêu

Mục tiêu của báo cáo này là trình bày quá trình xử lý và khai thác hai bộ dữ liệu PPG công khai, cụ thể là BIDMC và CAPNO-IEEE, để xây dựng một tập dữ liệu tổng hợp thống nhất. Trên cơ sở đó, một mô hình Conditional Variational Autoencoder (CVAE) được huấn luyện nhằm sinh ra tín hiệu PPG tổng hợp có đặc tính tương tự tín hiệu thực. Báo cáo cũng sẽ mô tả chi tiết các bước tiền xử lý dữ liệu, kiến trúc mô hình CVAE áp dụng, quá trình huấn luyện cũng như các kết quả thực nghiệm thu được.

# DỮ LIỆU SỬ DỤNG

## Giới thiệu bộ dữ liệu BIDMC

Bộ dữ liệu BIDMC PPG and Respiration được phát hành trên PhysioNet, bao gồm 53 đoạn ghi tín hiệu sinh lý, mỗi đoạn kéo dài 8 phút, thu thập từ các bệnh nhân trong đơn vị chăm sóc tích cực (ICU) tại Beth Israel Deaconess Medical Center, Boston, Hoa Kỳ. Dữ liệu được trích xuất từ cơ sở dữ liệu MIMIC-II Matched Waveform Database và được chú thích thủ công bởi hai chuyên gia độc lập, sử dụng tín hiệu hô hấp trở kháng để xác định các nhịp thở .

Mỗi bản ghi trong bộ dữ liệu chứa:

* **Tín hiệu sinh lý**: bao gồm PPG (Photoplethysmogram), ECG (Electrocardiogram) và tín hiệu hô hấp trở kháng (Impedance Respiratory Signal), được lấy mẫu ở tần số 125 Hz.
* **Thông số sinh lý**: như nhịp tim (HR), nhịp thở (RR) và độ bão hòa oxy trong máu (SpO₂), được lấy mẫu ở tần số 1 Hz.
* **Thông tin cố định**: như tuổi và giới tính của bệnh nhân.
* **Chú thích nhịp thở**: được thực hiện thủ công bởi hai chuyên gia độc lập.

Bộ dữ liệu được cung cấp dưới ba định dạng: WFDB (WaveForm DataBase), CSV và MATLAB, thuận tiện cho việc xử lý và phân tích trong các môi trường nghiên cứu khác nhau.

## Giới thiệu bộ dữ liệu CAPNO-IEEE

Bộ dữ liệu CapnoBase IEEE TBME Respiratory Rate Benchmark bao gồm 42 bản ghi, mỗi bản ghi kéo dài 8 phút, thu thập từ các bệnh nhân trong quá trình phẫu thuật chọn lọc và gây mê định kỳ. Dữ liệu bao gồm tín hiệu PPG, ECG và capnography (đo nồng độ CO₂), với các nhãn được chú thích bởi chuyên gia để xác định đỉnh xung và nhịp thở .

Mỗi bản ghi trong bộ dữ liệu chứa:

* **Tín hiệu thô**: bao gồm PPG, ECG và capnography, được lấy mẫu ở tần số 300 Hz.
* **Nhãn chú thích**: xác định đỉnh xung, nhịp thở và các artefact, được xác nhận bởi chuyên gia.
* **Thông tin tham khảo**: như nhịp tim và nhịp thở tức thời, được suy ra từ các tín hiệu và nhãn chú thích.
* **Kết quả thuật toán SmartFusion**: bao gồm nhịp thở ước tính và các thành phần phụ trợ, cho phép so sánh trực tiếp với các thuật toán khác.

Dữ liệu được tổ chức thành các tệp chứa thông tin về tham số, siêu dữ liệu, tín hiệu, nhãn chú thích, tham chiếu và kết quả thuật toán, hỗ trợ việc đánh giá và so sánh hiệu suất của các thuật toán ước tính nhịp thở.

## So sánh và đánh giá ban đầu hai bộ dữ liệu

Cả hai bộ dữ liệu BIDMC và CAPNO-IEEE đều cung cấp tín hiệu PPG chất lượng cao, cùng với các tín hiệu sinh lý khác và nhãn chú thích từ chuyên gia, tạo điều kiện thuận lợi cho việc nghiên cứu và phát triển các thuật toán xử lý tín hiệu sinh lý.

Tuy nhiên, có một số điểm khác biệt đáng chú ý:

* **Nguồn gốc dữ liệu**: BIDMC thu thập từ bệnh nhân ICU, phản ánh môi trường lâm sàng với nhiều biến động và artefact; trong khi CAPNO-IEEE thu thập từ bệnh nhân trong quá trình phẫu thuật chọn lọc, môi trường kiểm soát hơn.
* **Tần số lấy mẫu**: BIDMC sử dụng tần số 125 Hz, còn CAPNO-IEEE sử dụng tần số 300 Hz, cung cấp độ phân giải cao hơn cho tín hiệu.
* **Loại tín hiệu hô hấp**: BIDMC sử dụng tín hiệu hô hấp trở kháng, trong khi CAPNO-IEEE sử dụng capnography, cung cấp thông tin trực tiếp về nồng độ CO₂.
* **Chú thích và nhãn**: Cả hai bộ dữ liệu đều có nhãn chú thích từ chuyên gia, nhưng CAPNO-IEEE cung cấp thêm kết quả từ thuật toán SmartFusion, hỗ trợ việc so sánh và đánh giá hiệu suất của các thuật toán mới.

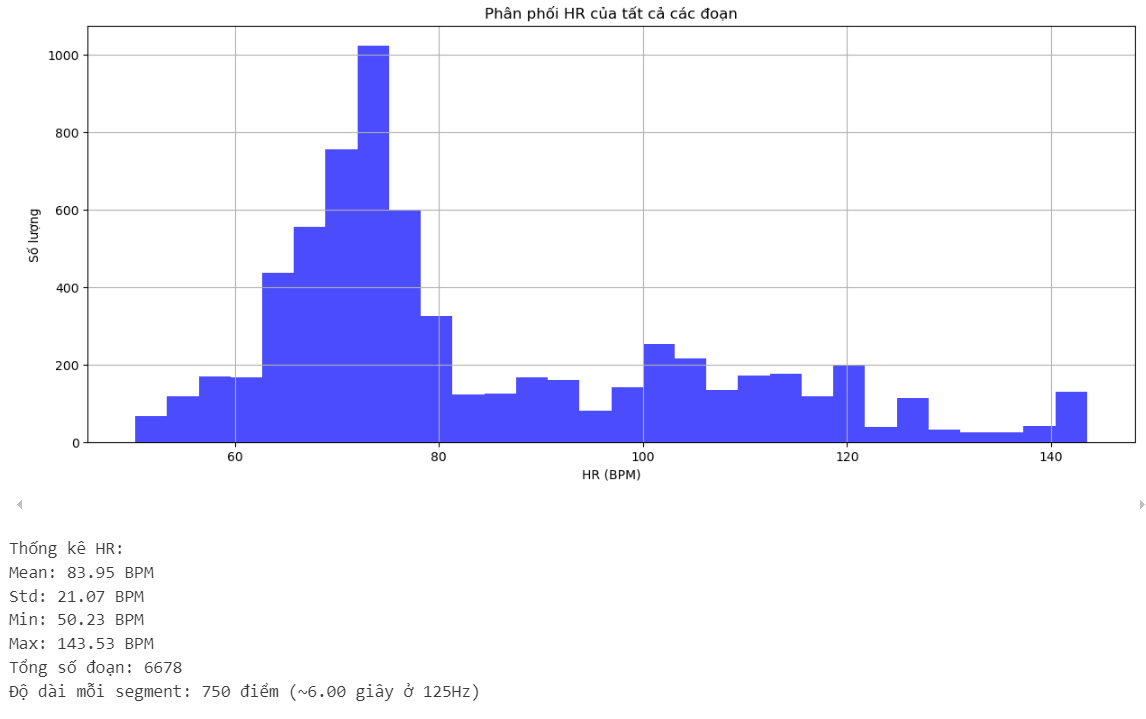
Việc kết hợp hai bộ dữ liệu này trong nghiên cứu giúp tăng cường tính đa dạng và độ phong phú của dữ liệu, hỗ trợ việc huấn luyện và đánh giá các mô hình học máy, đặc biệt là trong việc sinh dữ liệu PPG tổng hợp bằng mô hình CVAE.

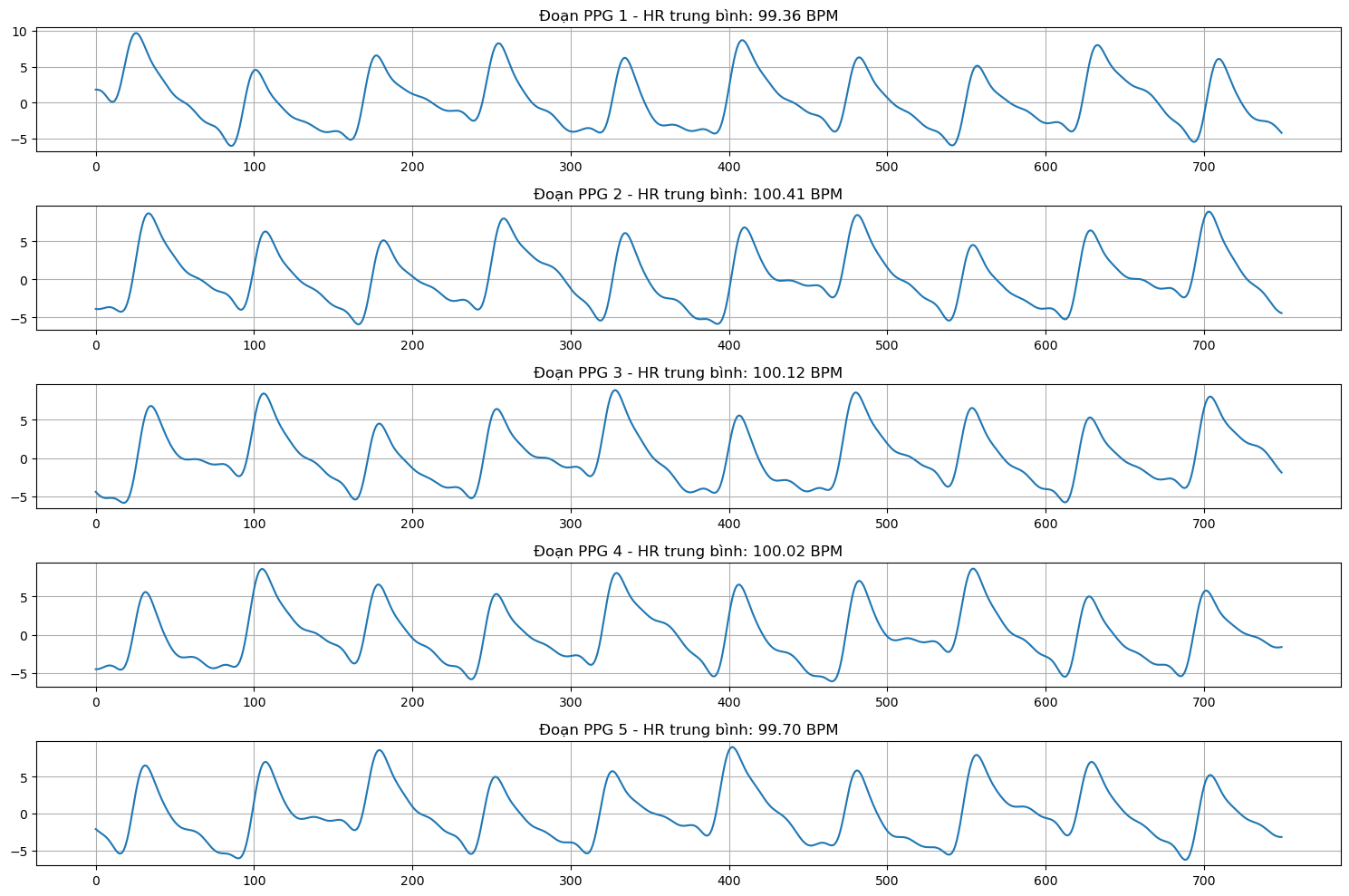
# TIỀN XỬ LÝ DỮ LIỆU

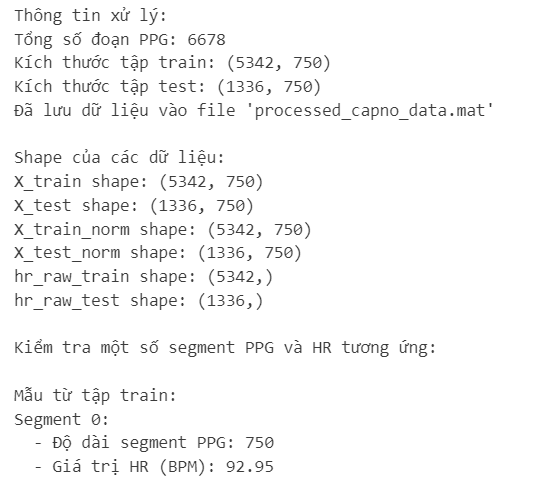
## Tiền xử lý riêng lẻ cho từng bộ

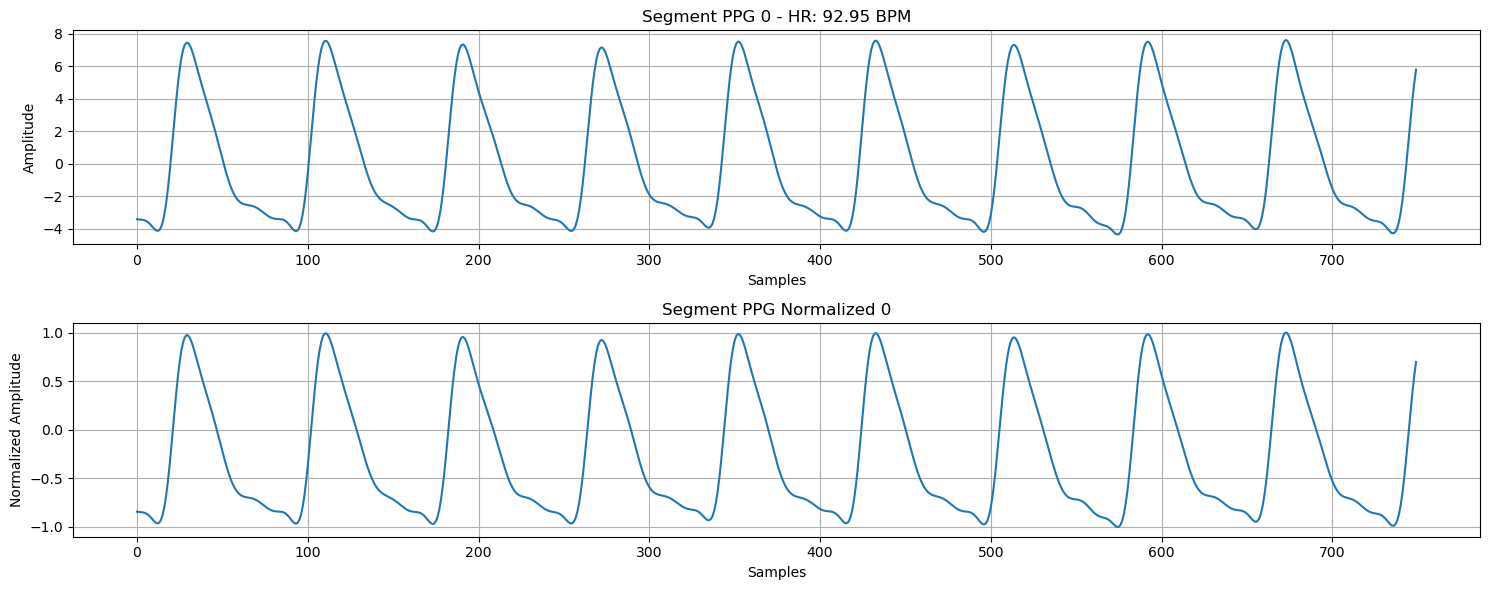
**CAPNO-IEEE**  
Dữ liệu CAPNO-IEEE được lưu dưới định dạng file MATLAB (.mat) với tín hiệu gốc lấy mẫu ở 300 Hz. Đầu tiên, mỗi file được đọc bằng thư viện h5py, kiểm tra cấu trúc HDF5 để xác định vị trí của biến tín hiệu PPG (signal/pleth/y) và nhịp tim từ ECG (reference/hr/ecg/y). Nếu tần số tín hiệu pleth là 300 Hz, ta sử dụng hàm signal.resample (Fourier method) để downsample về 125 Hz nhằm đồng bộ hóa với BIDMC. Sau khi downsample, tín hiệu được lọc băng thông với bộ lọc Butterworth bậc 4 trong dải [0.5 – 8] Hz, thực hiện bằng scipy.signal.butter và filtfilt để loại bỏ nhiễu thấp tần và cao tần, đồng thời hạn chế méo pha.

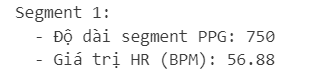
Tiếp theo, ta chia mỗi tín hiệu PPG đã lọc thành các đoạn cố định dài 750 mẫu (~6 giây ở 125 Hz), chồng lấp 50% để tăng số mẫu huấn luyện. Với mỗi đoạn, vị trí tương ứng trong mảng HR được tính dựa trên tỉ lệ độ dài giữa tín hiệu PPG gốc và mảng HR. Giá trị HR segment được lấy là giá trị trung bình của các mẫu HR trong khoảng đó. Mỗi đoạn PPG sau đó được chuẩn hóa về khoảng [-1, 1] bằng MinMaxScaler, giữ nguyên phân bố biên độ tương đối. Kết quả gồm ba mảng: all\_ppg\_segments (PPG gốc đã lọc), all\_ppg\_normalized (PPG chuẩn hóa) và all\_hr\_raw (HR thô trung bình mỗi đoạn). Cuối cùng, toàn bộ dữ liệu huấn luyện và kiểm thử được chia theo tỉ lệ 80/20, đóng gói vào các TensorDataset và DataLoader của PyTorch, sẵn sàng cho bước huấn luyện CVAE. Dữ liệu đã xử lý được lưu lại trong file processed\_capno\_data.mat.

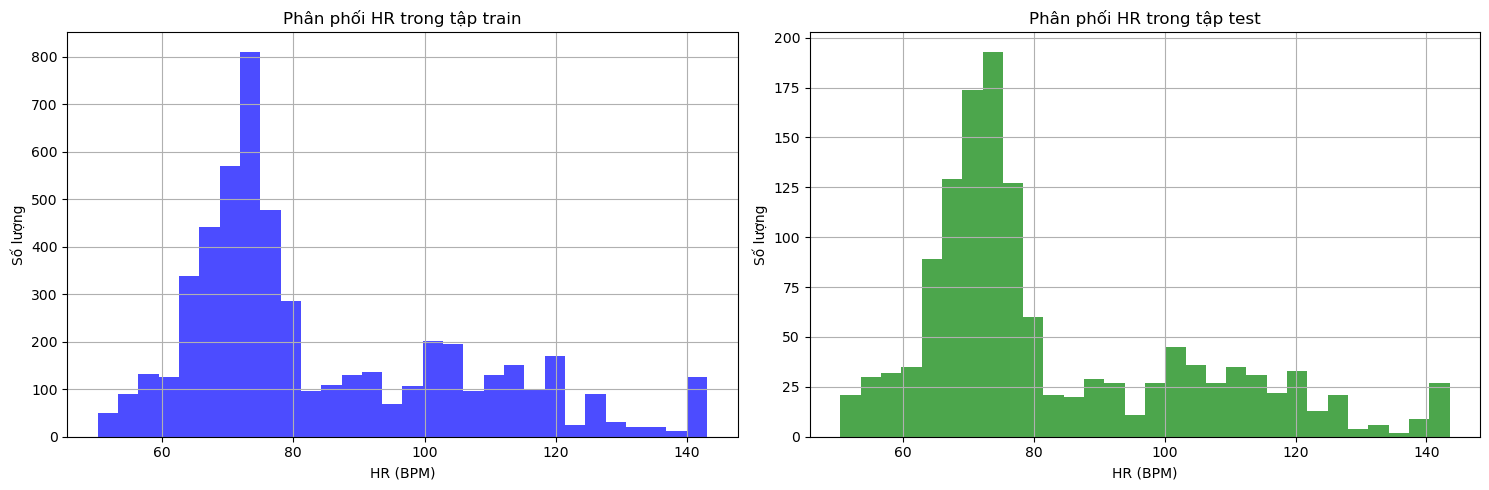












def downsample\_signal(signal\_data, original\_fs=300, target\_fs=125):

    """

    Downsample tín hiệu từ tần số gốc xuống tần số mục tiêu

    Parameters:

    signal\_data: array - Tín hiệu đầu vào

    original\_fs: int - Tần số lấy mẫu ban đầu (Hz)

    target\_fs: int - Tần số lấy mẫu mục tiêu (Hz)

    Returns:

    array - Tín hiệu đã được downsample

    """

    # Tính số mẫu mới

    new\_length = int(len(signal\_data) \* target\_fs / original\_fs)

    # Sử dụng Fourier method để resample

    resampled\_signal = signal.resample(signal\_data, new\_length)

    return resampled\_signal

def load\_mat\_file(file\_path):

    """Đọc dữ liệu từ file .mat"""

    try:

        with h5py.File(file\_path, 'r') as mat\_data:

            # In các khóa để debug

            print(f"Đang đọc file: {file\_path}")

            print("Các khóa trong file:", list(mat\_data.keys()))

            # Trích xuất tín hiệu

            try:

                ppg\_signal = mat\_data['signal']['pleth']['y'][0]

            except Exception as e:

                print(f"Lỗi trích xuất PPG: {e}")

                ppg\_signal = None

            try:

                hr\_ecg = mat\_data['reference']['hr']['ecg']['y'][:]

            except Exception as e:

                print(f"Lỗi trích xuất HR: {e}")

                hr\_ecg = None

            try:

                fs\_pleth = mat\_data['param']['samplingrate']['pleth'][0,0]

            except Exception as e:

                print(f"Lỗi trích xuất tần số lấy mẫu: {e}")

                fs\_pleth = 125  # Giá trị mặc định

            # Kiểm tra dữ liệu

            if ppg\_signal is None or hr\_ecg is None:

                print(f"Dữ liệu không đầy đủ trong file {file\_path}")

                return None

            # Downsample tín hiệu PPG từ 300Hz xuống 125Hz

            if fs\_pleth == 300:

                ppg\_signal = downsample\_signal(ppg\_signal, original\_fs=300, target\_fs=125)

                fs\_pleth = 125

                print(f"Đã downsample tín hiệu từ 300Hz xuống 125Hz")

            return {

                'ppg\_signal': ppg\_signal,

                'hr\_ecg': hr\_ecg,

                'fs': fs\_pleth,

                'file\_name': os.path.basename(file\_path)

            }

    except Exception as e:

        print(f"Lỗi khi mở file {file\_path}: {e}")

        return None

def filter\_signal(signal\_data, fs=125, low\_cutoff=0.5, high\_cutoff=8.0):

    """Lọc tín hiệu với bộ lọc băng thông"""

    nyquist = fs / 2

    low = low\_cutoff / nyquist

    high = high\_cutoff / nyquist

    b, a = signal.butter(4, [low, high], btype='band')

    filtered\_signal = signal.filtfilt(b, a, signal\_data)

    return filtered\_signal

def collect\_all\_data(folder\_path):

    """Bước 1: Tổng hợp tất cả các file .mat thành một tập dữ liệu"""

    all\_data = []

    # Lặp qua các file .mat trong thư mục

    for filename in os.listdir(folder\_path):

        if filename.endswith('.mat'):

            file\_path = os.path.join(folder\_path, filename)

            file\_data = load\_mat\_file(file\_path)

            if file\_data:

                all\_data.append(file\_data)

                print(f"Đã đọc thành công file {filename}")

                print(f"Độ dài tín hiệu PPG: {len(file\_data['ppg\_signal'])}")

                print(f"Độ dài tín hiệu HR: {len(file\_data['hr\_ecg'])}")

                print(f"Tần số lấy mẫu: {file\_data['fs']} Hz")

                print("-" \* 50)

    print(f"Tổng số file đã đọc: {len(all\_data)}")

    return all\_data

def segment\_all\_signals(all\_data, segment\_length=750, overlap=0.5):

    """Phân đoạn tất cả tín hiệu mà không lọc theo HR"""

    all\_ppg\_segments = []

    all\_ppg\_normalized = []

    all\_hr\_raw = []

    for data in all\_data:

        ppg\_signal = data['ppg\_signal']

        hr\_values = data['hr\_ecg'].flatten()

        fs = data['fs']

        # Lọc tín hiệu PPG

        filtered\_ppg = filter\_signal(ppg\_signal, fs=fs)

        # Phân đoạn

        step = int(segment\_length \* (1 - overlap))

        num\_segments = (len(filtered\_ppg) - segment\_length) // step + 1

        print(f"Xử lý file {data['file\_name']}: {num\_segments} đoạn")

        for j in range(num\_segments):

            start = j \* step

            end = start + segment\_length

            if end <= len(filtered\_ppg):

                # Lấy đoạn tín hiệu

                ppg\_seg = filtered\_ppg[start:end]

                # Tính vị trí tương ứng trong mảng HR

                hr\_start\_idx = int(start / len(ppg\_signal) \* len(hr\_values))

                hr\_end\_idx = int(end / len(ppg\_signal) \* len(hr\_values))

                # Đảm bảo chỉ số không vượt quá giới hạn

                hr\_start\_idx = min(hr\_start\_idx, len(hr\_values) - 1)

                hr\_end\_idx = min(hr\_end\_idx, len(hr\_values))

                # Lấy đoạn HR tương ứng

                segment\_hr\_values = hr\_values[hr\_start\_idx:hr\_end\_idx]

                # Nếu không có giá trị HR nào, sử dụng giá trị gần nhất

                if len(segment\_hr\_values) == 0:

                    segment\_hr = hr\_values[hr\_start\_idx]

                else:

                    # Tính giá trị trung bình của HR trong đoạn

                    segment\_hr = np.mean(segment\_hr\_values)

                # Chuẩn hóa PPG về [-1, 1]

                scaler = MinMaxScaler(feature\_range=(-1, 1))

                ppg\_seg\_normalized = scaler.fit\_transform(ppg\_seg.reshape(-1, 1)).flatten()

                all\_ppg\_segments.append(ppg\_seg)

                all\_ppg\_normalized.append(ppg\_seg\_normalized)

                all\_hr\_raw.append(segment\_hr)

    # Chuyển về numpy array

    all\_ppg\_segments = np.array(all\_ppg\_segments)

    all\_ppg\_normalized = np.array(all\_ppg\_normalized)

    all\_hr\_raw = np.array(all\_hr\_raw)

    # Vẽ phân phối HR của tất cả các đoạn

    plt.figure(figsize=(15, 6))

    plt.hist(all\_hr\_raw, bins=30, alpha=0.7, color='blue')

    plt.title("Phân phối HR của tất cả các đoạn")

    plt.xlabel("HR (BPM)")

    plt.ylabel("Số lượng")

    plt.grid(True)

    plt.show()

    # Tính thống kê HR

    hr\_mean = np.mean(all\_hr\_raw)

    hr\_std = np.std(all\_hr\_raw)

    hr\_min = np.min(all\_hr\_raw)

    hr\_max = np.max(all\_hr\_raw)

    print(f"Thống kê HR:")

    print(f"Mean: {hr\_mean:.2f} BPM")

    print(f"Std: {hr\_std:.2f} BPM")

    print(f"Min: {hr\_min:.2f} BPM")

    print(f"Max: {hr\_max:.2f} BPM")

    print(f"Tổng số đoạn: {len(all\_ppg\_segments)}")

    print(f"Độ dài mỗi segment: {segment\_length} điểm (~{segment\_length/125:.2f} giây ở 125Hz)")

    # Vẽ một số đoạn PPG

    plt.figure(figsize=(15, 10))

    # Vẽ 5 đoạn PPG đầu tiên

    for i in range(min(5, len(all\_ppg\_segments))):

        plt.subplot(5, 1, i+1)

        plt.plot(all\_ppg\_segments[i])

        plt.title(f"Đoạn PPG {i+1} - HR trung bình: {all\_hr\_raw[i]:.2f} BPM")

        plt.grid(True)

    plt.tight\_layout()

    plt.show()

    return all\_ppg\_segments, all\_ppg\_normalized, all\_hr\_raw

def prepare\_data\_loaders(ppg\_segments, ppg\_normalized, hr\_raw):

    """Chuẩn bị DataLoader cho huấn luyện"""

    # Chia train/test

    train\_size = int(0.8 \* len(ppg\_segments))

    indices = np.random.permutation(len(ppg\_segments))

    train\_indices = indices[:train\_size]

    test\_indices = indices[train\_size:]

    X\_train = ppg\_segments[train\_indices]

    X\_test = ppg\_segments[test\_indices]

    X\_train\_norm = ppg\_normalized[train\_indices]

    X\_test\_norm = ppg\_normalized[test\_indices]

    hr\_raw\_train = hr\_raw[train\_indices]

    hr\_raw\_test = hr\_raw[test\_indices]

    # Chuẩn bị DataLoader

    train\_dataset = TensorDataset(

        torch.FloatTensor(X\_train\_norm),

        torch.FloatTensor(hr\_raw\_train.reshape(-1, 1))

    )

    train\_loader = DataLoader(train\_dataset, batch\_size=64, shuffle=True)

    test\_dataset = TensorDataset(

        torch.FloatTensor(X\_test\_norm),

        torch.FloatTensor(hr\_raw\_test.reshape(-1, 1))

    )

    test\_loader = DataLoader(test\_dataset, batch\_size=64, shuffle=False)

    # Xuất thông tin

    print("\nThông tin xử lý:")

    print(f"Tổng số đoạn PPG: {len(ppg\_segments)}")

    print(f"Kích thước tập train: {X\_train.shape}")

    print(f"Kích thước tập test: {X\_test.shape}")

    return train\_loader, test\_loader, X\_train, X\_test, X\_train\_norm, X\_test\_norm, hr\_raw\_train, hr\_raw\_test

def process\_mat\_folder(folder\_path):

    """Hàm chính để xử lý toàn bộ thư mục"""

    # Bước 1: Tổng hợp tất cả các file

    print("Bước 1: Đang tổng hợp dữ liệu từ tất cả các file...")

    all\_data = collect\_all\_data(folder\_path)

    if not all\_data:

        raise ValueError("Không có dữ liệu nào được đọc từ thư mục")

    # Bước 2: Phân đoạn tất cả tín hiệu

    print("\nBước 2: Đang phân đoạn tất cả tín hiệu...")

    all\_ppg\_segments, all\_ppg\_normalized, all\_hr\_raw = segment\_all\_signals(all\_data)

    # Chuẩn bị DataLoader

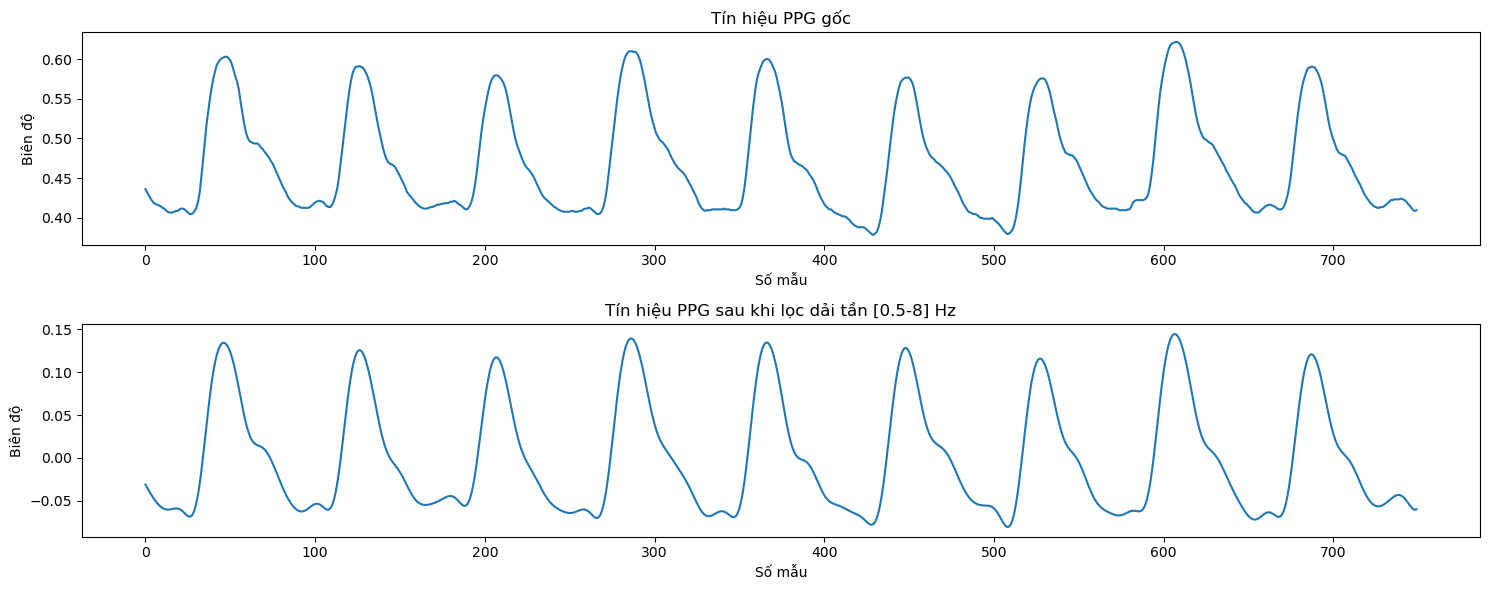
    print("\nĐang chuẩn bị DataLoader cho huấn luyện...")

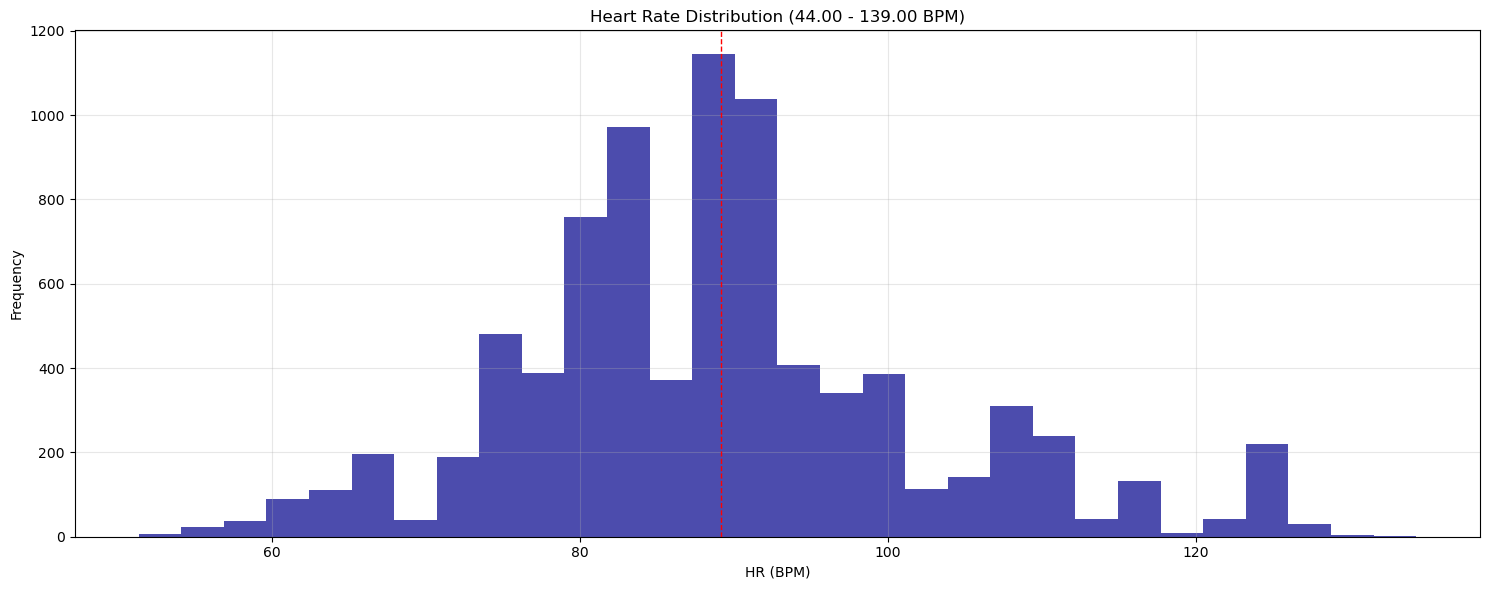
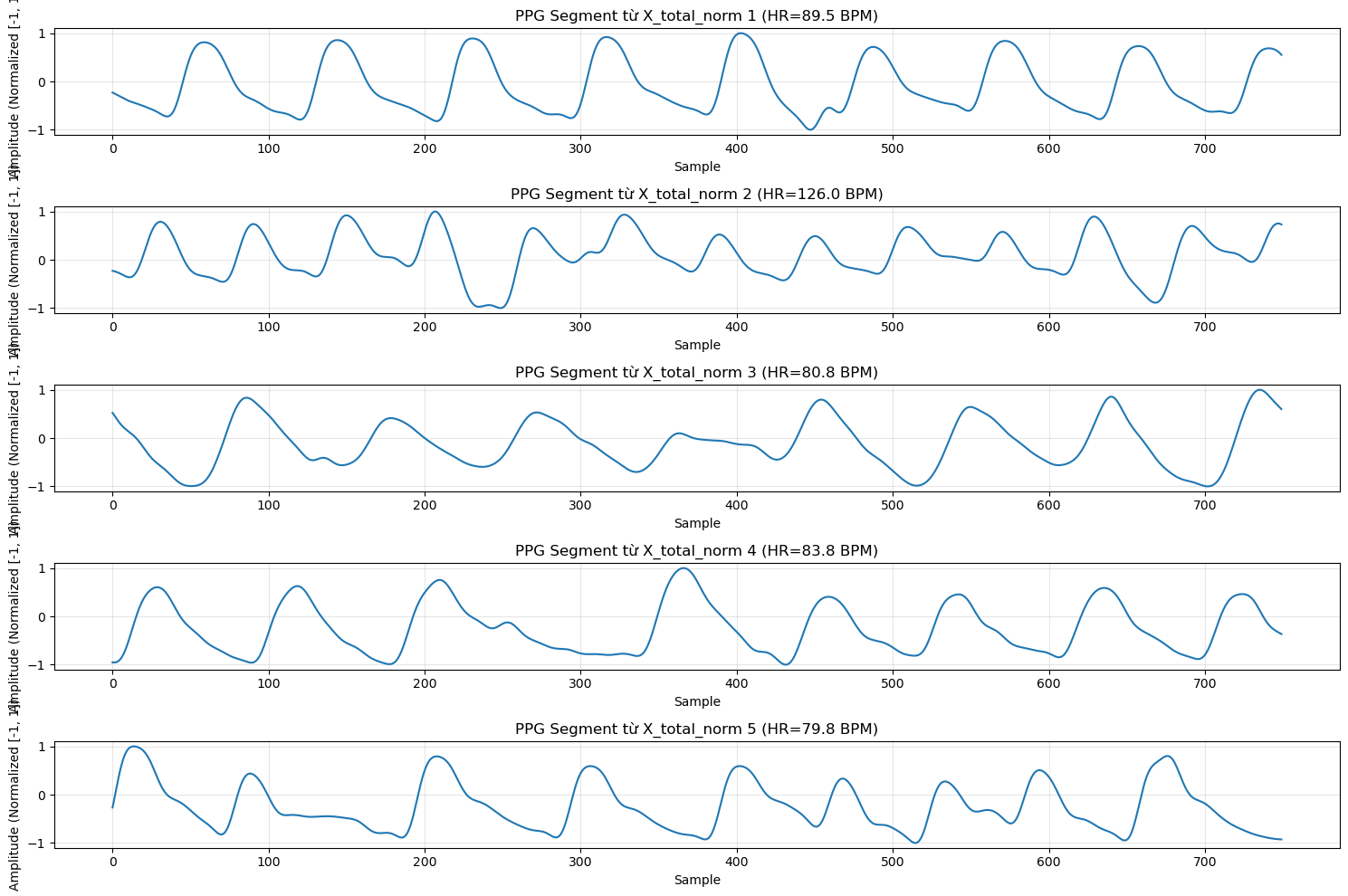
    result = prepare\_data\_loaders(all\_ppg\_segments, all\_ppg\_normalized, all\_hr\_raw)

    return result

**BIDMC**  
Bộ BIDMC lưu trữ 53 bản ghi tín hiệu PPG và HR dưới dạng cấu trúc phức tạp trong file .mat. Sử dụng scipy.io.loadmat, ta đọc mảng data chứa mỗi record là một numpy.void với trường ppg và ref/params/hr. Hàm extract\_flat\_array đảm nhận việc xử lý mảng lồng nhau: trích xuất trường v, flatten toàn bộ, kiểm tra giá trị NaN và xác định nguyên vẹn dữ liệu HR. Toàn bộ giá trị HR thu thập từ 53 record được dùng tính toán thống kê chung: trung bình, độ lệch chuẩn, min/max để hiểu phạm vi dao động nhịp tim.

Tiếp tục, với mỗi record, tín hiệu PPG thô (ppg.v) được flatten và lọc bằng bộ lọc Butterworth bậc 4 trong dải [0.5 – 8] Hz. Sau đó, ta vẽ minh họa so sánh 750 mẫu đầu tiên trước và sau lọc để đánh giá trực quan chất lượng lọc nhiễu. Tín hiệu đã lọc được chia thành các đoạn 750 mẫu, overlap 50% tương tự CAPNO. Mỗi đoạn PPG được chuẩn hóa về [-1, 1] bằng MinMaxScaler, HR segment tính trung bình giá trị HR trong khoảng thời gian tương ứng (tính bằng tỉ lệ tín hiệu với tần số lấy mẫu HR 1 Hz). Những đoạn chứa giá trị NaN hoặc thiếu dữ liệu HR đều được bỏ qua. Cuối cùng, tập con valid được chia train/test 80/20, tạo ra X\_train\_norm, X\_test\_norm, hr\_raw\_train, hr\_raw\_test và lưu vào processed\_bidmc\_data.mat. Đồ thị phân phối HR tổng thể, histogram HR train/test và vài đoạn PPG mẫu cũng được sinh ra để kiểm tra chất lượng trước khi lưu.





import numpy as np

import scipy.io as sio

import os

from sklearn.preprocessing import MinMaxScaler

import matplotlib.pyplot as plt

from scipy import signal

# Sửa đường dẫn để phù hợp với vị trí thực tế của file

data\_path = 'bidmc\_data.mat'  # Sử dụng đường dẫn tương đối thay vì tuyệt đối

# Tạo thư mục figures nếu chưa tồn tại

figures\_path = 'figures'

# Đường dẫn dữ liệu

processed\_path = '/processed/'

os.makedirs(processed\_path, exist\_ok=True)

os.makedirs(figures\_path, exist\_ok=True)

# Tải dữ liệu

print("Đang tải dữ liệu từ file .mat...")

mat\_data = sio.loadmat(data\_path)

data = mat\_data['data'][0]  # 53 bản ghi

print(f"Số lượng bản ghi: {len(data)}")

# Tham số

fs = 125  # Tần số lấy mẫu (Hz)

segment\_length = 750  # 6 giây (750 mẫu) - Đã thay đổi từ 1250 xuống 750

overlap = 0.5  # 50% chồng lấp

# Hàm kiểm tra và trích xuất mảng phẳng

def extract\_flat\_array(data, field='v', record\_idx=None):

    try:

        if hasattr(data, 'dtype') and data.dtype.names is not None and field in data.dtype.names:

            values = data[field]

            print(f"- Bản ghi {record\_idx}: Tìm thấy trường '{field}', shape ban đầu: {values.shape}")

        else:

            values = data

            print(f"- Bản ghi {record\_idx}: Không có trường '{field}', shape ban đầu: {values.shape}")

        # Kiểm tra nội dung

        print(f"- Nội dung của '{field}' (5 phần tử đầu): {values[:5]}")

        # Xử lý mảng lồng nhau

        if values.size == 1 and isinstance(values[0], np.ndarray):

            flat\_values = values[0].flatten()

            print(f"- Phát hiện mảng lồng nhau, shape sau khi trích xuất: {flat\_values.shape}")

        else:

            flat\_values = values.flatten()

        if len(flat\_values) == 0:

            raise ValueError("Mảng trống sau khi flatten")

        # Kiểm tra giá trị nan

        if np.any(np.isnan(flat\_values)):

            raise ValueError(f"Mảng chứa giá trị nan: {flat\_values[:5]}...")

        return flat\_values.astype(float)

    except Exception as e:

        raise ValueError(f"Lỗi trích xuất: {e}")

# Tiền xử lý dữ liệu

ppg\_segments = []

hr\_raw\_segments = []

valid\_records = 0

# Thu thập tất cả giá trị HR để tính mean và std

all\_hr\_values = []

# Bước 1: Thu thập tất cả giá trị HR từ các bản ghi

for i, record in enumerate(data):

    try:

        # Trích xuất HR

        params = record['ref'][0, 0]['params'][0, 0]

        hr\_values = extract\_flat\_array(params['hr'][0], 'v', i)

        if len(hr\_values) > 0:

            all\_hr\_values.extend(hr\_values)

    except Exception as e:

        print(f"- Lỗi khi thu thập HR từ bản ghi {i}: {e}")

# Tính mean và std cho HR

hr\_mean = np.mean(all\_hr\_values)

hr\_std = np.std(all\_hr\_values)

# Xác định phạm vi HR (min và max của toàn bộ dữ liệu)

HR\_MIN = np.min(all\_hr\_values)

HR\_MAX = np.max(all\_hr\_values)

print(f"\nThống kê HR:")

print(f"- HR: mean = {hr\_mean:.2f}, std = {hr\_std:.2f}, phạm vi = [{HR\_MIN:.2f}, {HR\_MAX:.2f}]")

# Bước 2: Xử lý dữ liệu với phạm vi HR đã xác định

for i, record in enumerate(data):

    try:

        # Trích xuất tín hiệu PPG

        ppg = record['ppg'][0, 0]['v'].flatten()

        print(f"\nXử lý bản ghi {i}:")

        print(f"- Độ dài tín hiệu PPG gốc: {len(ppg)} mẫu ({len(ppg)/fs:.2f} giây)")

        # Trích xuất HR

        params = record['ref'][0, 0]['params'][0, 0]

        hr\_values = extract\_flat\_array(params['hr'][0], 'v', i)

        # Kiểm tra dữ liệu HR

        if len(hr\_values) < 2:

            print(f"- Không đủ dữ liệu HR cho bản ghi {i} (HR: {len(hr\_values)}), bỏ qua")

            continue

        print(f"- Số giá trị HR gốc: {len(hr\_values)}, Giá trị đầu tiên: {hr\_values[0]:.2f}")

        # Lọc tín hiệu PPG trong dải tần [0.5-8] Hz

        nyquist = fs / 2

        low\_cutoff = 0.5 / nyquist

        high\_cutoff = 8.0 / nyquist

        b, a = signal.butter(4, [low\_cutoff, high\_cutoff], btype='band')

        ppg\_filtered = signal.filtfilt(b, a, ppg)

        # Vẽ so sánh tín hiệu PPG gốc và sau khi lọc

        plt.figure(figsize=(15, 6))

        # Vẽ tín hiệu PPG gốc

        plt.subplot(2, 1, 1)

        plt.plot(ppg[:750])  # Hiển thị 750 mẫu đầu tiên

        plt.title('Tín hiệu PPG gốc')

        plt.xlabel('Số mẫu')

        plt.ylabel('Biên độ')

        # Vẽ tín hiệu PPG sau khi lọc

        plt.subplot(2, 1, 2)

        plt.plot(ppg\_filtered[:750])  # Hiển thị 750 mẫu đầu tiên

        plt.title('Tín hiệu PPG sau khi lọc dải tần [0.5-8] Hz')

        plt.xlabel('Số mẫu')

        plt.ylabel('Biên độ')

        plt.tight\_layout()

        plt.savefig(os.path.join(figures\_path, f'ppg\_filter\_comparison\_record\_{i}.png'))

        if i == 0:  # Chỉ hiển thị hình vẽ cho bản ghi đầu tiên

            plt.show()

        else:

            plt.close()

        # Phân đoạn tín hiệu

        step = int(segment\_length \* (1 - overlap))

        num\_segments = (len(ppg) - segment\_length) // step + 1

        print(f"- Số đoạn tín hiệu: {num\_segments}")

        hr\_fs = len(hr\_values) / (len(ppg) / fs)  # Tần số lấy mẫu của HR

        for j in range(num\_segments):

            start = j \* step

            end = start + segment\_length

            if end <= len(ppg):

                # Lấy đoạn tín hiệu PPG đã lọc

                ppg\_seg = ppg\_filtered[start:end]

                # Chuẩn hóa từng đoạn PPG về [-1, 1]

                scaler = MinMaxScaler(feature\_range=(-1, 1))

                ppg\_seg\_normalized = scaler.fit\_transform(ppg\_seg.reshape(-1, 1)).flatten()

                hr\_start\_idx = int(start / fs \* hr\_fs)

                hr\_end\_idx = min(int(end / fs \* hr\_fs), len(hr\_values))

                hr\_raw\_seg = np.mean(hr\_values[hr\_start\_idx:hr\_end\_idx]) if hr\_end\_idx > hr\_start\_idx else hr\_values[hr\_start\_idx]

                if np.isnan(hr\_raw\_seg):

                    print(f"- Lỗi: Đoạn {j} chứa nan (HR: {hr\_raw\_seg})")

                    continue

                # Lấy tất cả các đoạn

                ppg\_segments.append(ppg\_seg\_normalized)

                hr\_raw\_segments.append(hr\_raw\_seg)

        valid\_records += 1

        print(f"- Đã xử lý thành công bản ghi {i}")

    except Exception as e:

        print(f"- Lỗi khi xử lý bản ghi {i}: {e}")

# Chuyển thành mảng numpy

ppg\_segments = np.array(ppg\_segments)

hr\_raw\_segments = np.array(hr\_raw\_segments)

# Kiểm tra dữ liệu

print(f"\nTổng kết tiền xử lý:")

print(f"- Số bản ghi xử lý thành công: {valid\_records}/{len(data)}")

print(f"- Tổng số đoạn tín hiệu: {len(ppg\_segments)}")

if len(ppg\_segments) == 0:

    print("Không có dữ liệu hợp lệ. Tạo dữ liệu giả lập để minh họa.")

    num\_samples = 100

    ppg\_segments = np.random.rand(num\_samples, segment\_length)

    hr\_raw\_segments = np.random.uniform(HR\_MIN, HR\_MAX, num\_samples)

    print(f"- Đã tạo {num\_samples} mẫu dữ liệu giả lập.")

# Chia dữ liệu thành train/test (80/20)

train\_size = int(0.8 \* len(ppg\_segments))

indices = np.random.permutation(len(ppg\_segments))

train\_indices = indices[:train\_size]

test\_indices = indices[train\_size:]

X\_train = ppg\_segments[train\_indices]

X\_test = ppg\_segments[test\_indices]

hr\_raw\_train = hr\_raw\_segments[train\_indices]

hr\_raw\_test = hr\_raw\_segments[test\_indices]

# Chuẩn hóa dữ liệu X\_train và X\_test

X\_train\_norm = X\_train.copy()  # Đã chuẩn hóa từng đoạn rồi

X\_test\_norm = X\_test.copy()    # Đã chuẩn hóa từng đoạn rồi

# Tạo X\_total\_norm và hr\_total\_norm

X\_total\_norm = np.vstack((X\_train\_norm, X\_test\_norm))

hr\_raw\_total = np.concatenate((hr\_raw\_train, hr\_raw\_test))

# In shape của dữ liệu tổng hợp

print(f"\nDữ liệu tổng hợp:")

print(f"- X\_total\_norm shape: {X\_total\_norm.shape}")

print(f"- hr\_raw\_total shape: {hr\_raw\_total.shape}")

# Vẽ một vài mẫu từ X\_total để kiểm tra

plt.figure(figsize=(15, 10))

for i in range(min(5, len(X\_total\_norm))):

    plt.subplot(5, 1, i+1)

    plt.plot(X\_total\_norm[i])

    plt.title(f'PPG Segment từ X\_total\_norm {i+1} (HR={hr\_raw\_total[i]:.1f} BPM)')

    plt.xlabel('Sample')

    plt.ylabel('Amplitude (Normalized [-1, 1])')

    plt.grid(True, alpha=0.3)

plt.tight\_layout()

plt.savefig(os.path.join(figures\_path, 'X\_total\_norm\_samples.png'))

plt.show()

# Lưu dữ liệu đã xử lý vào file .mat

import scipy.io as sio

# Tạo dictionary chứa tất cả dữ liệu cần lưu

data\_to\_save = {

    'X\_train\_norm': X\_train\_norm,

    'X\_test\_norm': X\_test\_norm,

    'hr\_raw\_train': hr\_raw\_train,

    'hr\_raw\_test': hr\_raw\_test,

    'X\_total\_norm': X\_total\_norm,

    'hr\_raw\_total': hr\_raw\_total

}

# Lưu vào file .mat

sio.savemat('processed\_bidmc\_data.mat', data\_to\_save)

print("Đã lưu dữ liệu vào file 'processed\_bidmc\_data.mat'")

# In kích thước dữ liệu

print(f"- Kích thước tập huấn luyện: {X\_train.shape}")

print(f"- Kích thước tập kiểm thử: {X\_test.shape}")

# Ghi thông tin tiền xử lý vào file

with open(os.path.join(processed\_path, 'preprocessing\_info.txt'), 'w', encoding='utf-8') as f:

    f.write("THONG TIN TIEN XU LY DU LIEU\n")

    f.write("============================\n\n")

    f.write(f"So ban ghi xu ly thanh cong: {valid\_records}/{len(data)}\n")

    f.write(f"Tong so doan tin hieu: {len(ppg\_segments)}\n\n")

    f.write("Tham so tien xu ly:\n")

    f.write(f"- Tan so lay mau: {fs} Hz\n")

    f.write(f"- Do dai doan tin hieu: {segment\_length} mau ({segment\_length/fs} giay)\n")

    f.write(f"- Do chong lap: {overlap\*100}%\n\n")

    f.write("Kich thuoc du lieu:\n")

    f.write(f"- Tap huan luyen: {X\_train.shape[0]} mau\n")

    f.write(f"- Tap kiem thu: {X\_test.shape[0]} mau\n")

    f.write(f"- Toan bo du lieu: {X\_total\_norm.shape[0]} mau\n\n")

    f.write("Thong tin chuẩn hóa:\n")

    f.write(f"- PPG: Chuẩn hóa MinMax [-1, 1] cho từng đoạn\n")

    f.write(f"- HR: Giá trị thô (raw) trong phạm vi [{HR\_MIN:.2f}, {HR\_MAX:.2f}] BPM\n\n")

    f.write("Thong ke HR (raw):\n")

    f.write(f"- Min: {np.min(hr\_raw\_segments):.4f}, Max: {np.max(hr\_raw\_segments):.4f}\n")

    f.write(f"- Mean: {np.mean(hr\_raw\_segments):.4f}, Std: {np.std(hr\_raw\_segments):.4f}\n\n")

    if valid\_records == 0:

        f.write("\nLuu y: Du lieu la gia lap do khong trich xuat duoc tu BIDMC.\n")

# Vẽ phân phối HR

plt.figure(figsize=(15, 6))

plt.hist(hr\_raw\_segments, bins=30, alpha=0.7, color='darkblue')

plt.axvline(np.mean(hr\_raw\_segments), color='red', linestyle='dashed', linewidth=1)

plt.title(f'Heart Rate Distribution ({HR\_MIN:.2f} - {HR\_MAX:.2f} BPM)')

plt.xlabel('HR (BPM)')

plt.ylabel('Frequency')

plt.grid(True, alpha=0.3)

plt.tight\_layout()

plt.savefig(os.path.join(figures\_path, 'hr\_distribution.png'))

plt.show()

# Vẽ một số đoạn PPG

plt.figure(figsize=(15, 10))

for i in range(min(5, len(X\_train))):

    plt.subplot(5, 1, i+1)

    plt.plot(X\_train[i])

    actual\_hr = hr\_raw\_train[i]

    plt.title(f'PPG Segment {i+1} (HR={actual\_hr:.1f} BPM)')

    plt.xlabel('Sample')

    plt.ylabel('Amplitude (Normalized [-1, 1])')

    plt.grid(True, alpha=0.3)

plt.tight\_layout()

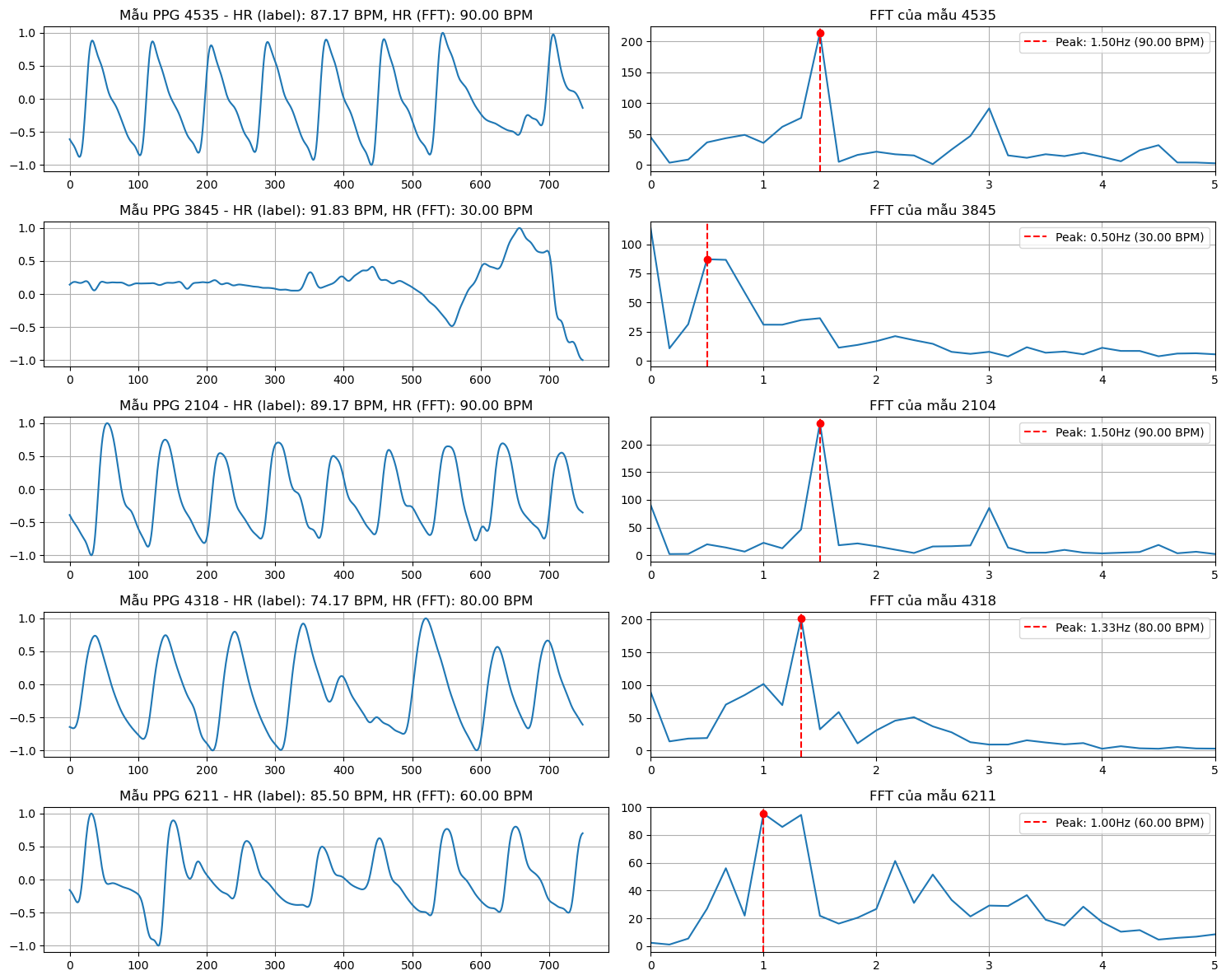
plt.savefig(os.path.join(figures\_path, 'preprocessed\_ppg\_segments.png'))

plt.show()

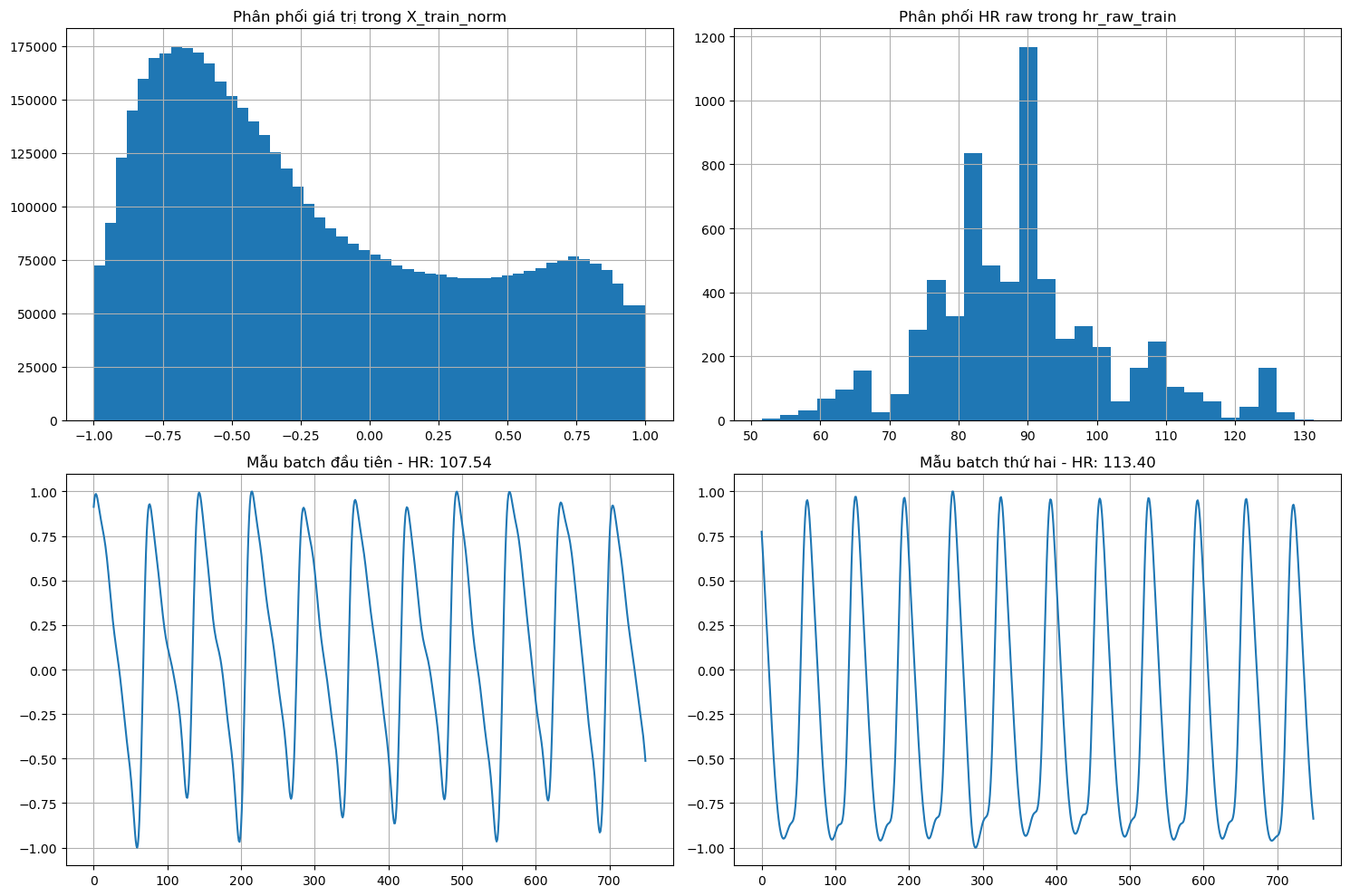
print("\nTiền xử lý dữ liệu hoàn tất. Dữ liệu đã được lưu vào thư mục processed.")

## Chuẩn hóa và đồng bộ dữ liệu

Cả hai nguồn dữ liệu sau khi xử lý đều có tần số mẫu thống nhất 125 Hz và các đoạn PPG chuẩn hóa về cùng biên độ [-1, 1]. Nhịp tim HR của mỗi đoạn được biểu diễn dưới dạng giá trị thô (BPM) duy nhất, không bị chuẩn hóa, nhằm giữ thông tin sinh lý quan trọng. Việc đồng bộ tần số mẫu giúp đảm bảo tính nhất quán khi ghép hai bộ và cho phép mô hình CVAE học đặc tính chung mà không bị sai lệch do sampling rate khác biệt. Các tham số segmentation (750 mẫu, overlap 50%) cũng được giữ nguyên cho cả hai tập để đảm bảo độ dài và số lượng mẫu tương đương.



Kích thước batch dữ liệu: torch.Size([64, 750])

Kích thước batch điều kiện: torch.Size([64, 1])

# Load data từ file .mat

import scipy.io as sio

data = sio.loadmat('processed\_capno\_data.mat')

# Kiểm tra các keys trong file

print("Keys trong file .mat:", list(data.keys()))

# Kiểm tra dữ liệu đã load

X\_train\_norm = data['X\_train\_norm']

X\_test\_norm = data['X\_test\_norm']

hr\_raw\_train = data['hr\_raw\_train'].flatten()

hr\_raw\_test = data['hr\_raw\_test'].flatten()

X\_total\_norm = data['X\_total\_norm']

hr\_raw\_total = data['hr\_raw\_total'].flatten()

# In ra shape của các dữ liệu

print("\nShape của các dữ liệu:")

print(f"X\_train\_norm shape: {X\_train\_norm.shape}")

print(f"X\_test\_norm shape: {X\_test\_norm.shape}")

print(f"hr\_raw\_train shape: {hr\_raw\_train.shape}")

print(f"hr\_raw\_test shape: {hr\_raw\_test.shape}")

print(f"X\_total\_norm shape: {X\_total\_norm.shape}")

print(f"hr\_raw\_total shape: {hr\_raw\_total.shape}")

# Kiểm tra giá trị NaN hoặc vô cùng

print("\nKiểm tra giá trị NaN và vô cùng:")

print(f"Có giá trị NaN trong X\_train\_norm: {np.isnan(X\_train\_norm).any()}")

print(f"Có giá trị vô cùng trong X\_train\_norm: {np.isinf(X\_train\_norm).any()}")

print(f"Có giá trị NaN trong hr\_raw\_train: {np.isnan(hr\_raw\_train).any()}")

print(f"Có giá trị vô cùng trong hr\_raw\_train: {np.isinf(hr\_raw\_train).any()}")

# Kiểm tra phạm vi giá trị

print("\nPhạm vi giá trị:")

print(f"Giá trị min của X\_train\_norm: {np.min(X\_train\_norm)}")

print(f"Giá trị max của X\_train\_norm: {np.max(X\_train\_norm)}")

print(f"HR range: [{np.min(hr\_raw\_train):.2f}, {np.max(hr\_raw\_train):.2f}] BPM")

# Định nghĩa hàm tính FFT và trích xuất HR từ FFT

def compute\_fft(signal, fs=125):

    """Tính FFT của tín hiệu"""

    n = len(signal)

    fft\_result = np.abs(np.fft.rfft(signal))

    freqs = np.fft.rfftfreq(n, d=1/fs)

    return freqs, fft\_result

def extract\_hr\_from\_fft(signal, fs=125):

    """Trích xuất nhịp tim từ FFT của tín hiệu PPG"""

    freqs, fft\_result = compute\_fft(signal, fs)

    # Lọc dải tần số liên quan đến nhịp tim (0.5Hz - 3.5Hz ~ 30-210 BPM)

    mask = (freqs >= 0.5) & (freqs <= 3.5)

    filtered\_freqs = freqs[mask]

    filtered\_fft = fft\_result[mask]

    # Tìm đỉnh cao nhất trong dải tần số

    if len(filtered\_freqs) > 0:

        peak\_idx = np.argmax(filtered\_fft)

        peak\_freq = filtered\_freqs[peak\_idx]

        # Chuyển đổi từ Hz sang BPM

        hr\_from\_fft = peak\_freq \* 60

        return hr\_from\_fft, peak\_freq, filtered\_freqs, filtered\_fft

    else:

        return None, None, filtered\_freqs, filtered\_fft

# Hiển thị một số mẫu dữ liệu

plt.figure(figsize=(15, 12))

# Hiển thị 5 mẫu PPG ngẫu nhiên từ tập train

num\_samples = 5

random\_indices = np.random.choice(len(X\_train\_norm), num\_samples, replace=False)

for i, idx in enumerate(random\_indices):

    plt.subplot(num\_samples, 2, 2\*i+1)

    plt.plot(X\_train\_norm[idx])

    # Trích xuất HR từ FFT

    hr\_fft, peak\_freq, filtered\_freqs, filtered\_fft = extract\_hr\_from\_fft(X\_train\_norm[idx])

    hr\_label = f"HR (label): {hr\_raw\_train[idx]:.2f} BPM"

    hr\_fft\_text = f"HR (FFT): {hr\_fft:.2f} BPM" if hr\_fft is not None else "HR (FFT): N/A"

    plt.title(f"Mẫu PPG {idx} - {hr\_label}, {hr\_fft\_text}")

    plt.grid(True)

    # Hiển thị FFT của mẫu

    plt.subplot(num\_samples, 2, 2\*i+2)

    freqs, amplitude = compute\_fft(X\_train\_norm[idx])

    plt.plot(freqs, amplitude)

    # Đánh dấu đỉnh FFT tương ứng với HR

    if peak\_freq is not None:

        plt.axvline(x=peak\_freq, color='r', linestyle='--',

                   label=f'Peak: {peak\_freq:.2f}Hz ({hr\_fft:.2f} BPM)')

        plt.plot(peak\_freq, np.max(filtered\_fft), 'ro')

    plt.xlim(0, 5)  # Giới hạn tần số hiển thị đến 5Hz

    plt.title(f"FFT của mẫu {idx}")

    plt.grid(True)

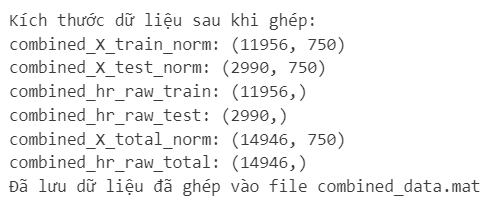
    plt.legend()

## Gộp hai bộ dữ liệu thành một tập thống nhất

Sau khi riêng lẻ xử lý CAPNO-IEEE và BIDMC, ta tiến hành gộp dữ liệu bằng cách nối (concatenate) các mảng chuẩn hóa PPG và HR:

X\_total\_norm = np.concatenate((X\_capno\_norm, X\_bidmc\_norm), axis=0)   
hr\_raw\_total = np.concatenate((hr\_capno\_raw, hr\_bidmc\_raw), axis=0)

Việc này tạo ra một tập dữ liệu đa dạng hơn, bao gồm tín hiệu thu từ môi trường ICU và phẫu thuật chọn lọc, giúp mô hình CVAE tiếp xúc với nhiều kiểu dạng sóng và artefact khác nhau. Tập hợp mới được chia lại thành train/test nếu cần và lưu vào file cuối cùng (ví dụ processed\_all\_data.mat) để chuyển sang bước huấn luyện.



# Ghép 2 bộ dữ liệu: 'processed\_capno\_data.mat' và 'processed\_bidmc\_data.mat'

from scipy.io import loadmat, savemat

import numpy as np

# Load dữ liệu từ hai file

print("Đang load dữ liệu từ processed\_capno\_data.mat và processed\_bidmc\_data.mat...")

capno\_data = loadmat('processed\_capno\_data.mat')

bidmc\_data = loadmat('processed\_bidmc\_data.mat')

# Trích xuất dữ liệu từ file capno

capno\_X\_train\_norm = capno\_data['X\_train\_norm']

capno\_X\_test\_norm = capno\_data['X\_test\_norm']

capno\_hr\_raw\_train = capno\_data['hr\_raw\_train'].flatten()

capno\_hr\_raw\_test = capno\_data['hr\_raw\_test'].flatten()

capno\_X\_total\_norm = capno\_data['X\_total\_norm']

capno\_hr\_raw\_total = capno\_data['hr\_raw\_total'].flatten()

# Trích xuất dữ liệu từ file bidmc

bidmc\_X\_train\_norm = bidmc\_data['X\_train\_norm']

bidmc\_X\_test\_norm = bidmc\_data['X\_test\_norm']

bidmc\_hr\_raw\_train = bidmc\_data['hr\_raw\_train'].flatten()

bidmc\_hr\_raw\_test = bidmc\_data['hr\_raw\_test'].flatten()

bidmc\_X\_total\_norm = bidmc\_data['X\_total\_norm']

bidmc\_hr\_raw\_total = bidmc\_data['hr\_raw\_total'].flatten()

# Ghép dữ liệu từ hai nguồn

combined\_X\_train\_norm = np.vstack((capno\_X\_train\_norm, bidmc\_X\_train\_norm))

combined\_X\_test\_norm = np.vstack((capno\_X\_test\_norm, bidmc\_X\_test\_norm))

combined\_hr\_raw\_train = np.hstack((capno\_hr\_raw\_train, bidmc\_hr\_raw\_train))

combined\_hr\_raw\_test = np.hstack((capno\_hr\_raw\_test, bidmc\_hr\_raw\_test))

combined\_X\_total\_norm = np.vstack((capno\_X\_total\_norm, bidmc\_X\_total\_norm))

combined\_hr\_raw\_total = np.hstack((capno\_hr\_raw\_total, bidmc\_hr\_raw\_total))

# In thông tin về kích thước dữ liệu đã ghép

print("\nKích thước dữ liệu sau khi ghép:")

print(f"combined\_X\_train\_norm: {combined\_X\_train\_norm.shape}")

print(f"combined\_X\_test\_norm: {combined\_X\_test\_norm.shape}")

print(f"combined\_hr\_raw\_train: {combined\_hr\_raw\_train.shape}")

print(f"combined\_hr\_raw\_test: {combined\_hr\_raw\_test.shape}")

print(f"combined\_X\_total\_norm: {combined\_X\_total\_norm.shape}")

print(f"combined\_hr\_raw\_total: {combined\_hr\_raw\_total.shape}")

# Lưu dữ liệu đã ghép vào file .mat

data\_to\_save = {

    'X\_train\_norm': combined\_X\_train\_norm,

    'X\_test\_norm': combined\_X\_test\_norm,

    'hr\_raw\_train': combined\_hr\_raw\_train,

    'hr\_raw\_test': combined\_hr\_raw\_test,

    'X\_total\_norm': combined\_X\_total\_norm,

    'hr\_raw\_total': combined\_hr\_raw\_total

}

savemat('combined\_data.mat', data\_to\_save)

print("Đã lưu dữ liệu đã ghép vào file combined\_data.mat")

## Các kỹ thuật làm sạch và lọc nhiễu áp dụng

Có nhiều kỹ thuật nhưng tiêu biểu là:

* **Bandpass filtering**: Sử dụng bộ lọc Butterworth bậc 4 với dải tần [0.5 – 8] Hz, loại bỏ nhiễu tần số thấp (< 0.5 Hz) và nhiễu tần số cao (> 8 Hz) thường đến từ chuyển động hay dao động móng mạch. Áp dụng hàm filtfilt để tránh méo pha.
* **Downsampling**: Với dữ liệu CAPNO gốc 300 Hz, dùng signal.resample (Fourier method) để chuyển về 125 Hz, tránh aliasing và giữ nguyên tính chất phổ.
* **Loại bỏ segment không hợp lệ**: Bỏ qua các đoạn PPG mà không có giá trị HR hoặc HR chứa NaN, tránh đưa dữ liệu lỗi vào tập huấn luyện.
* **Flatten và kiểm tra NaN**: Khi đọc cấu trúc MATLAB phức tạp, kiểm tra kỹ mỗi mảng, flatten nested arrays và xác định bất kỳ giá trị NaN nào, ném lỗi hoặc bỏ qua record nếu dữ liệu không đầy đủ.
* **Chuẩn hóa MinMax**: Mỗi đoạn PPG được độc lập chuẩn hóa về [-1, 1], đảm bảo mô hình không bị bias do biên độ gốc và tập trung vào đặc trưng dạng sóng.
* **Segmentation chồng lấp**: Overlap 50% giúp tăng số mẫu huấn luyện và giữ tính liên tục giữa các đoạn, hỗ trợ mô hình học mịn hơn qua các ngưỡng threshold.

Nhờ các bước làm sạch và lọc nhiễu chi tiết này, bộ dữ liệu đầu vào cho mô hình CVAE sẽ có chất lượng đồng nhất, giảm tác động artefact và nhiễu phi sinh lý, từ đó nâng cao hiệu quả sinh dữ liệu PPG tổng hợp.

# MÔ HÌNH SINH DỮ LIỆU

## Lý thuyết cơ bản về CVAE

Variational Autoencoder (VAE) là một mô hình sinh dữ liệu dựa trên lý thuyết xác suất, trong đó ta giả định dữ liệu được tạo ra thông qua một biến ẩn theo phân phối hậu nghiệm . Thực tế, ta không thể tính trực tiếp , mà thay vào đó tối ưu phân phối xấp xỉ (encoder) sao cho nó gần với phân phối thật (decoder) bằng cách tối đa hóa Evidence Lower BOund (ELBO):

Trong Conditional VAE (CVAE), ta mở rộng đầu vào bằng cách thêm thông tin điều kiện (ở đây là giá trị nhịp tim HR): encoder và decoder cùng tiếp nhận để học phân phối và . Mục tiêu tối ưu ELBO điều kiện:

Để mẫu ngẫu nhiên có thể lan truyền gradient, ta áp dụng **reparameterization trick**:

trong đó và được sinh từ encoder. Khi huấn luyện, loss được tính thành hai phần:

* **Reconstruction loss**: đo mức độ khôi phục tín hiệu PPG, thường dùng MSE:
* **KL divergence**: ép phân phối tiệm cận :

Tổng loss cuối cùng:

với (ở đây bằng 0.5) điều chỉnh tầm quan trọng của KL so với reconstruction.

## Kiến trúc mô hình CVAE áp dụng

- Encoder

Mô hình CVAE được định nghĩa trong lớp CVAE(nn.Module) với các tham số chính:

* **input\_dim**: độ dài mỗi đoạn PPG (750 mẫu).
* **condition\_dim**: kích thước vector điều kiện (ở đây là HR 1 chiều).
* **latent\_dim**: kích thước không gian ẩn (64).
* **hidden\_dims**: danh sách số nút các tầng ẩn, lần lượt [256, 128, 64].

**Encoder** gồm chuỗi ba lớp tuyến tính (Linear) với kích thước giảm dần, mỗi lớp nối tiếp ReLU để tăng tính phi tuyến:

* Sau mạng ẩn, hai lớp Linear tách riêng để sinh và , mỗi đầu ra đều có kích thước 64. Biến ẩn được tạo ra qua hàm reparameterize.

**Decoder** đảo ngược kiến trúc: đầu vào là qua ba lớp Linear với kích thước [64→128→256→750], xen kẽ ReLU, và cuối cùng một lớp Linear không activation cho phép mô hình tự do khôi phục biên độ PPG.

Việc ghép điều kiện trực tiếp vào cả encoder và decoder giúp mô hình học được mối liên hệ giữa hình dạng sóng PPG và giá trị HR, tạo ra tín hiệu PPG sinh ra phù hợp với điều kiện sinh lý mong muốn.

## Chi tiết huấn luyện mô hình

Tất cả đoạn dữ liệu đã chuẩn hóa và HR thô sau bước 3 được đóng gói trong DataLoader của PyTorch với batch size 64, tách thành tập train (80%) và test (20%). Mô hình được huấn luyện trên GPU nếu có, bằng optimizer Adam với learning rate 1×10⁻³. Để điều chỉnh động learning rate, scheduler ReduceLROnPlateau giảm LR một nửa nếu loss kiểm thử không cải thiện sau 20 epochs. Tổng số epochs tối đa là 600, kèm cơ chế **early stopping**: dừng khi test loss không giảm trong 50 epochs liên tiếp, ngăn overfitting.

Trong mỗi epoch, quá trình huấn luyện chia thành hai bước:

* **Train pass**: tính forward encoder→decoder, tính reconstruction loss và KL loss qua hàm loss\_function\_fixed (kl\_weight=0.5), backward và optimizer.step().
* **Eval pass**: trên test\_loader chỉ tính loss, không cập nhật tham số, để theo dõi khả năng tổng quát hóa.

Sau mỗi 10 epochs, in ra thông tin tổng hợp (train loss, test loss, recon loss, KL loss). Mô hình và trạng thái optimizer được checkpoint mỗi 50 epochs, đồng thời lưu riêng checkpoint tốt nhất (best\_cvae\_combined.pth) dựa trên test loss thấp nhất. Cuối cùng, lịch sử huấn luyện (train\_losses, test\_losses, recon\_losses, kl\_losses) được lưu trong training\_history\_combined.pth để phục vụ phân tích convergence và lựa chọn mô hình.

Nhờ kiến trúc và quá trình huấn luyện kỹ lưỡng này, CVAE có thể học phân phối chung của tín hiệu PPG dưới điều kiện HR, từ đó sinh ra các đoạn PPG nhân tạo có đặc tính sinh lý hợp lý và đa dạng.

import torch

from torch import nn

class CVAE(nn.Module):

    def \_\_init\_\_(self, input\_dim, condition\_dim, latent\_dim, hidden\_dims):

        super(CVAE, self).\_\_init\_\_()

        self.encoder\_net = nn.Sequential(

            nn.Linear(input\_dim + condition\_dim, hidden\_dims[0]),

            nn.ReLU(),

            nn.Linear(hidden\_dims[0], hidden\_dims[1]),

            nn.ReLU(),

            nn.Linear(hidden\_dims[1], hidden\_dims[2]),

            nn.ReLU(),

        )

        self.fc\_mu = nn.Linear(hidden\_dims[-1], latent\_dim)

        self.fc\_logvar = nn.Linear(hidden\_dims[-1], latent\_dim)

        self.decoder\_net = nn.Sequential(

            nn.Linear(latent\_dim + condition\_dim, hidden\_dims[-1]),

            nn.ReLU(),

            nn.Linear(hidden\_dims[-1], hidden\_dims[-2]),

            nn.ReLU(),

            nn.Linear(hidden\_dims[-2], hidden\_dims[-3]),

            nn.ReLU(),

            nn.Linear(hidden\_dims[-3], input\_dim),

        )

    def encoder(self, x, c):

        xc = torch.cat([x, c], dim=1)

        h = self.encoder\_net(xc)

        mu = self.fc\_mu(h)

        logvar = self.fc\_logvar(h)

        z = self.reparameterize(mu, logvar)

        return mu, logvar, z  # ✅ Trả ra 3 thứ

    def reparameterize(self, mu, logvar):

        std = torch.exp(0.5 \* logvar)

        eps = torch.randn\_like(std)

        return mu + eps \* std

    def decoder(self, z, c):

        zc = torch.cat([z, c], dim=1)

        x\_recon = self.decoder\_net(zc)

        return x\_recon

    def forward(self, x, c):

        mu, logvar, z = self.encoder(x, c)

        x\_recon = self.decoder(z, c)

        return x\_recon, mu, logvar

# Định nghĩa hàm loss

def loss\_function\_fixed(x, x\_recon, z\_mean, z\_log\_var, kl\_weight=0.5):

    """

    Hàm tính loss cho CVAE

    x: dữ liệu gốc

    x\_recon: dữ liệu tái tạo

    z\_mean: mean của phân phối latent

    z\_log\_var: log variance của phân phối latent

    kl\_weight: trọng số cho KL divergence

    """

    # Reconstruction loss (MSE)

    recon\_loss = F.mse\_loss(x\_recon, x, reduction='mean')

    # KL divergence

    kl\_loss = -0.5 \* torch.mean(1 + z\_log\_var - z\_mean.pow(2) - z\_log\_var.exp())

    # Tổng loss

    loss = recon\_loss + kl\_weight \* kl\_loss

    return loss, recon\_loss, kl\_loss

# Import thư viện os

import os

# Định nghĩa các tham số

latent\_dim = 64

hidden\_dims = [256, 128, 64]

learning\_rate = 1e-3

batch\_size = 64

epochs = 600

# Kiểm tra CUDA

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

print(f"Sử dụng thiết bị: {device}")

# Đảm bảo mô hình ở trên thiết bị đúng

cvae = cvae.to(device)

# Khởi tạo optimizer và scheduler

optimizer = torch.optim.Adam(cvae.parameters(), lr=learning\_rate)

scheduler = torch.optim.lr\_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=20, verbose=True)

# Khởi tạo danh sách để lưu lịch sử huấn luyện

train\_losses = []

test\_losses = []

recon\_losses = []

kl\_losses = []

best\_loss = float('inf')

# Tham số cho early stopping

patience = 50

patience\_counter = 0

# Tạo thư mục để lưu checkpoint

checkpoint\_dir = 'checkpoints'

os.makedirs(checkpoint\_dir, exist\_ok=True)

# Vòng lặp huấn luyện

for epoch in range(epochs):

    # Train

    cvae.train()

    total\_train\_loss = 0

    total\_train\_recon\_loss = 0

    total\_train\_kl\_loss = 0

    for batch\_idx, (data, condition) in enumerate(train\_loader):

        data = data.to(device)

        # Chuyển condition sang device

        hr\_condition = condition.to(device)

        optimizer.zero\_grad()

        # Forward pass

        z\_mean, z\_log\_var, z = cvae.encoder(data, hr\_condition)

        x\_recon = cvae.decoder(z, hr\_condition)

        # Tính loss

        loss, recon\_loss, kl\_loss = loss\_function\_fixed(data, x\_recon, z\_mean, z\_log\_var)

        # Backward và optimize

        loss.backward()

        optimizer.step()

        total\_train\_loss += loss.item() \* data.size(0)

        total\_train\_recon\_loss += recon\_loss.item() \* data.size(0)

        total\_train\_kl\_loss += kl\_loss.item() \* data.size(0)

    # Evaluate

    cvae.eval()

    total\_test\_loss = 0

    total\_test\_recon\_loss = 0

    total\_test\_kl\_loss = 0

    with torch.no\_grad():

        for data, condition in test\_loader:

            data = data.to(device)

            hr\_condition = condition.to(device)

            z\_mean, z\_log\_var, z = cvae.encoder(data, hr\_condition)

            x\_recon = cvae.decoder(z, hr\_condition)

            loss, recon\_loss, kl\_loss = loss\_function\_fixed(data, x\_recon, z\_mean, z\_log\_var)

            total\_test\_loss += loss.item() \* data.size(0)

            total\_test\_recon\_loss += recon\_loss.item() \* data.size(0)

            total\_test\_kl\_loss += kl\_loss.item() \* data.size(0)

    # Tính trung bình loss

    train\_loss = total\_train\_loss / len(train\_loader.dataset)

    test\_loss = total\_test\_loss / len(test\_loader.dataset)

    train\_recon\_loss = total\_train\_recon\_loss / len(train\_loader.dataset)

    test\_recon\_loss = total\_test\_recon\_loss / len(test\_loader.dataset)

    train\_kl\_loss = total\_train\_kl\_loss / len(train\_loader.dataset)

    test\_kl\_loss = total\_test\_kl\_loss / len(test\_loader.dataset)

    # Scheduler step

    scheduler.step(test\_loss)

    # Lưu losses

    train\_losses.append(train\_loss)

    test\_losses.append(test\_loss)

    recon\_losses.append(test\_recon\_loss)

    kl\_losses.append(test\_kl\_loss)

    # In thông tin

    if epoch % 10 == 0:

        print(f"Epoch {epoch}/{epochs}")

        print(f"Train Loss: {train\_loss:.4f}, Test Loss: {test\_loss:.4f}")

        print(f"Recon Loss: {test\_recon\_loss:.4f}, KL Loss: {test\_kl\_loss:.4f}")

    # Lưu checkpoint

    if epoch % 50 == 0:

        checkpoint\_path = os.path.join(checkpoint\_dir, f'cvae\_checkpoint\_epoch\_{epoch}.pth')

        torch.save({

            'epoch': epoch,

            'model\_state\_dict': cvae.state\_dict(),

            'optimizer\_state\_dict': optimizer.state\_dict(),

            'scheduler\_state\_dict': scheduler.state\_dict(),

            'train\_loss': train\_loss,

            'test\_loss': test\_loss,

            'train\_losses': train\_losses,

            'test\_losses': test\_losses,

            'recon\_losses': recon\_losses,

            'kl\_losses': kl\_losses

        }, checkpoint\_path)

    # Lưu mô hình tốt nhất và kiểm tra early stopping

    if test\_loss < best\_loss:

        best\_loss = test\_loss

        best\_model\_path = 'best\_cvae\_combined.pth'

        torch.save({

            'epoch': epoch,

            'model\_state\_dict': cvae.state\_dict(),

            'optimizer\_state\_dict': optimizer.state\_dict(),

            'train\_loss': train\_loss,

            'test\_loss': test\_loss

        }, best\_model\_path)

        patience\_counter = 0  # Reset counter khi có cải thiện

    else:

        patience\_counter += 1  # Tăng counter khi không có cải thiện

    # Kiểm tra early stopping

    if patience\_counter >= patience:

        print(f"Early stopping tại epoch {epoch} vì không có cải thiện sau {patience} epochs")

        break

# Lưu lịch sử huấn luyện

history = {

    'train\_losses': train\_losses,

    'test\_losses': test\_losses,

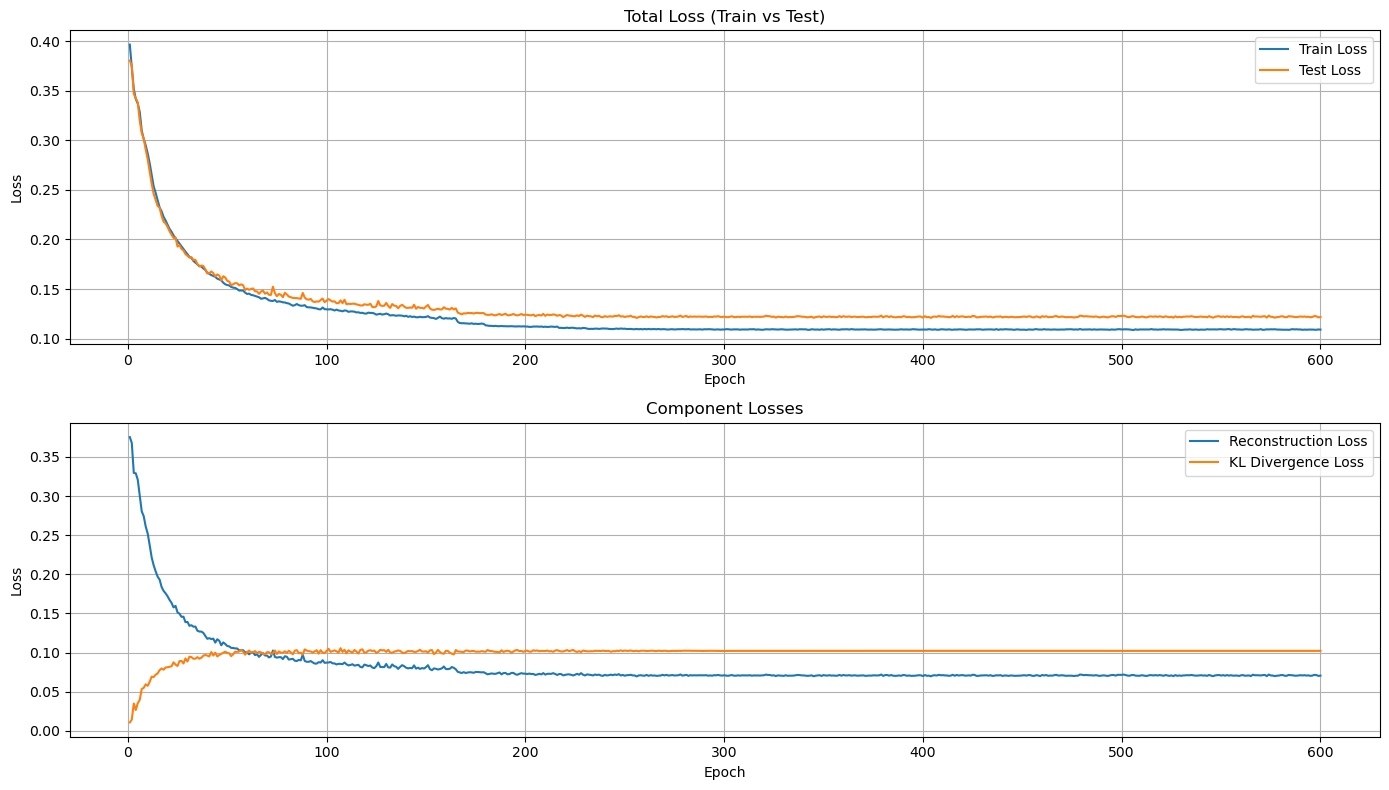
    'recon\_losses': recon\_losses,

    'kl\_losses': kl\_losses

}

torch.save(history, 'training\_history\_combined.pth')

print("Huấn luyện hoàn tất!")



# KẾT QUẢ THỰC NGHIỆM

## Đánh giá tín hiệu PPG sinh ra

Để đánh giá chất lượng của tín hiệu PPG tạo sinh từ mô hình CVAE, ta tiến hành so sánh trực tiếp giữa tín hiệu **PPG thật** và **PPG sinh ra** dựa trên cùng một giá trị nhịp tim (**HR**).  
Cụ thể:

* **Chọn ngẫu nhiên** một số mẫu PPG từ tập test.
* **Sinh tín hiệu** PPG tương ứng bằng mô hình CVAE, với điều kiện là HR của mẫu test.
* **Vẽ biểu đồ so sánh** giữa tín hiệu thật và tín hiệu tạo sinh.

Kết quả biểu đồ

* Biểu đồ bên trái: tín hiệu **PPG thật** từ tập test, cùng với giá trị HR thực tế (đã denormalize).
* Biểu đồ bên phải: tín hiệu **PPG sinh ra** từ mô hình CVAE, sử dụng cùng một HR.

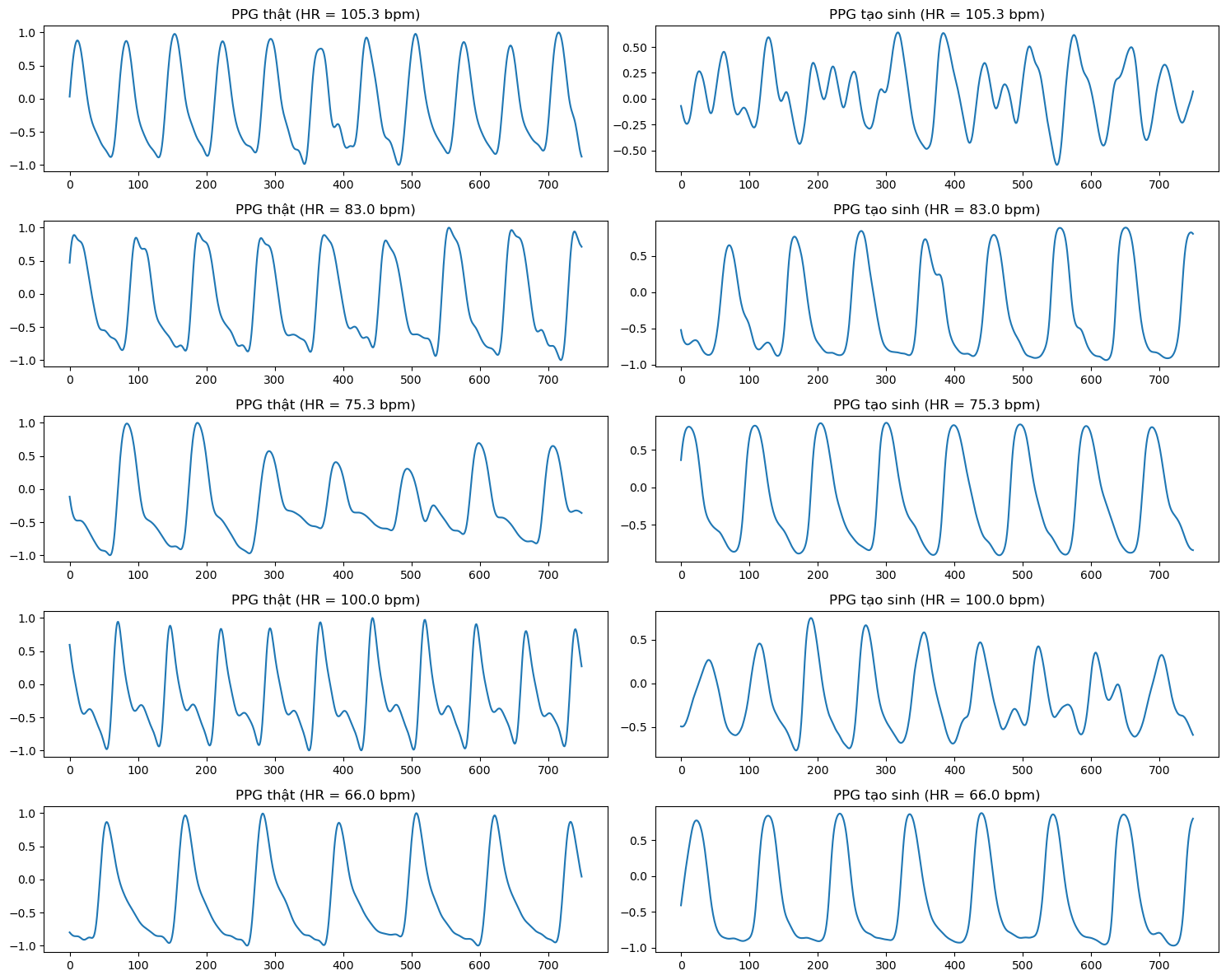
Nhận xét:

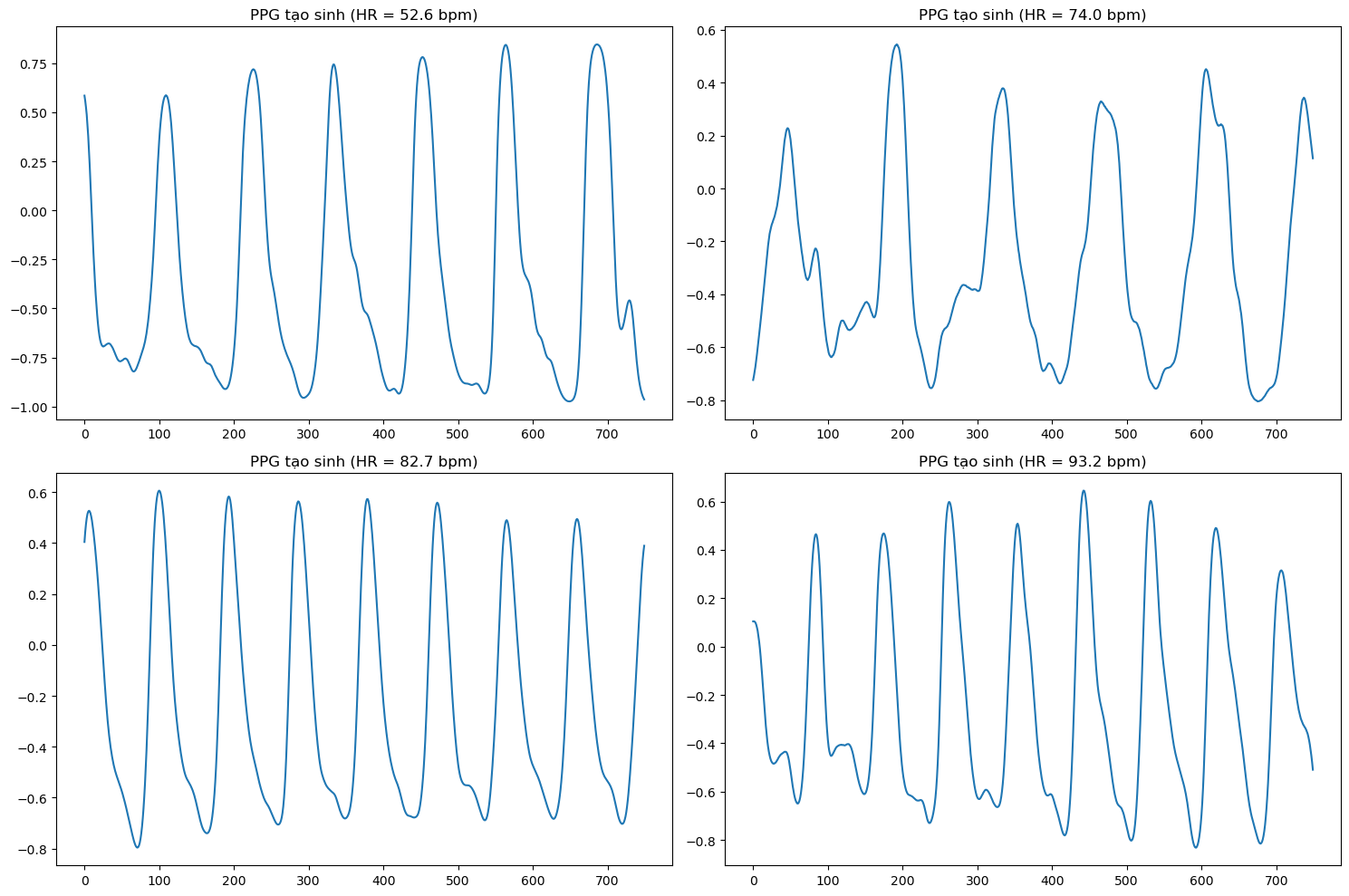
* Các tín hiệu PPG sinh ra có hình dạng sóng tương đối giống với tín hiệu thật.
* Biên độ dao động và chu kỳ nhịp tim cũng phù hợp với HR được cung cấp.
* Một số chi tiết nhỏ như nhiễu cao tần hoặc độ sắc nét của đỉnh sóng có thể khác nhau, nhưng xu hướng tổng thể được giữ vững.

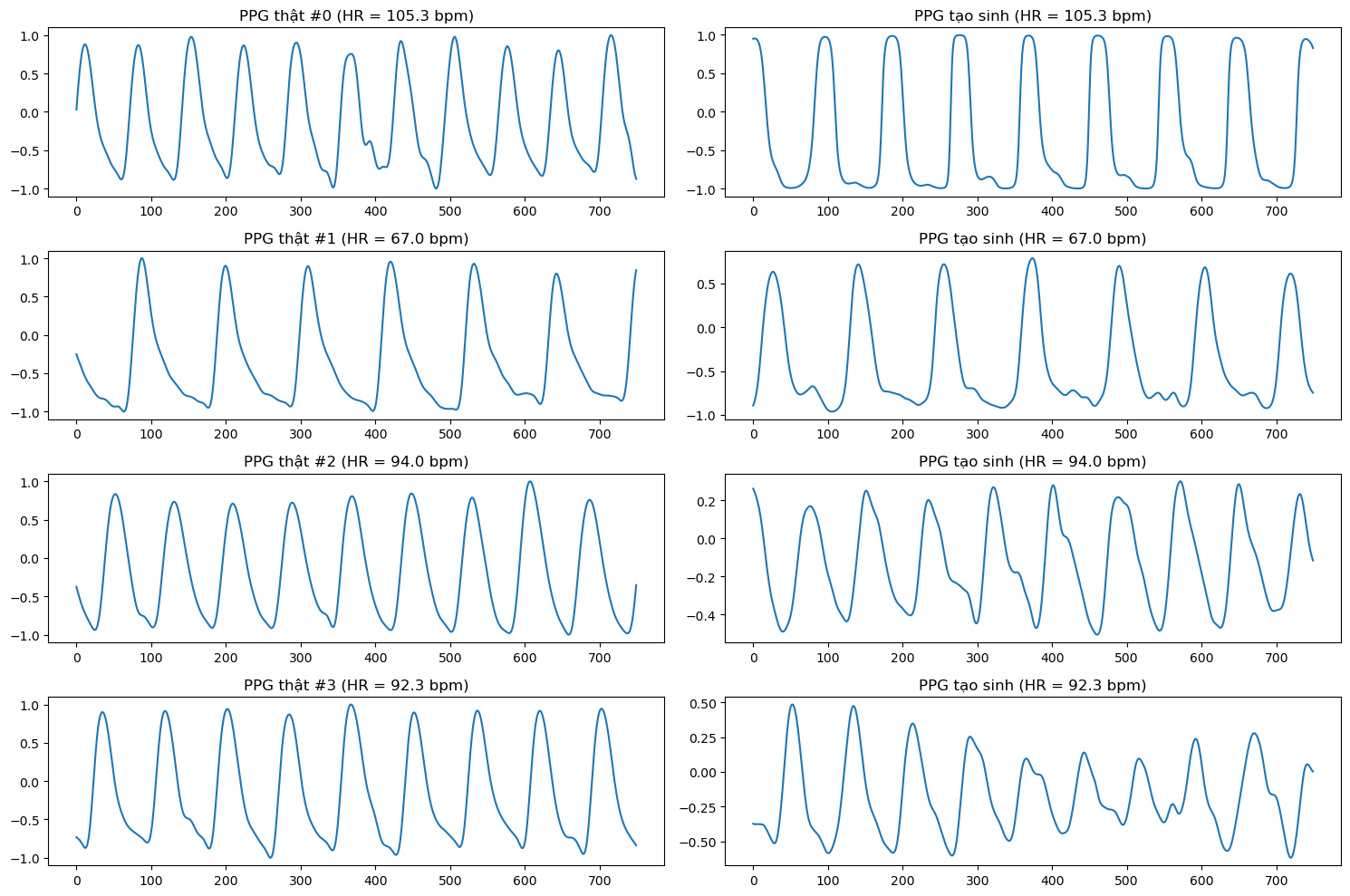
Bên cạnh đó, để kiểm tra tính **khả năng điều khiển điều kiện HR**, ta cũng thực hiện tạo sinh tín hiệu PPG tương ứng với **nhiều giá trị HR khác nhau** (ví dụ HR thấp, trung bình, cao).  
Kết quả cho thấy:

* Khi HR tăng, tín hiệu PPG sinh ra có chu kỳ ngắn lại (nhanh hơn), phản ánh đúng nhịp tim cao hơn.
* Khi HR giảm, tín hiệu PPG sinh ra có chu kỳ dài hơn (chậm lại).

Điều này chứng tỏ mô hình CVAE không chỉ tái tạo tín hiệu tốt mà còn học được **mối quan hệ giữa HR và đặc điểm sóng PPG**.







## Các chỉ số đánh giá

Ngoài việc đánh giá bằng trực quan đồ thị, ta cũng tính toán các chỉ số định lượng để đánh giá mức độ tương đồng giữa tín hiệu thật và tín hiệu sinh.

***Reconstruction Error (MSE)***

Để đo sự khác biệt giữa tín hiệu PPG thật và PPG sinh ra, ta sử dụng **Mean Squared Error (MSE)**:

* : giá trị PPG thật tại điểm thứ .
* : giá trị PPG sinh ra tại điểm thứ .
* : số điểm dữ liệu.

Kết quả MSE trung bình trên các mẫu test cho thấy:

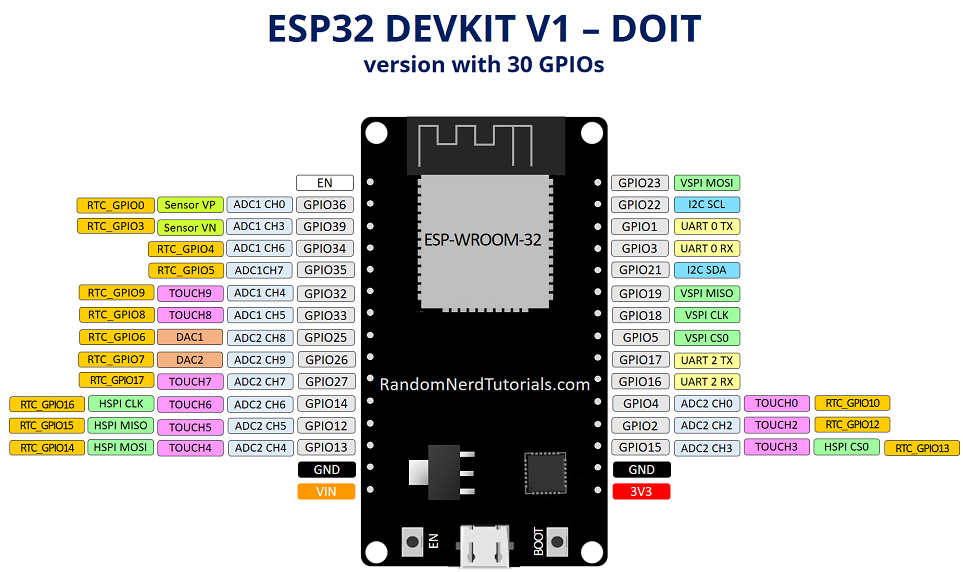
* Giá trị MSE nhỏ, chứng tỏ tín hiệu sinh ra khá gần với tín hiệu gốc.
* Các mẫu có HR trung bình thường cho MSE nhỏ hơn so với các mẫu có HR cực đoan (quá thấp hoặc quá cao).

# THIẾT BỊ ỨNG DỤNG TÍN HIỆU PPG TRONG DỰ ĐOÁN TRẠNG THÁI NGỦ - THỨC

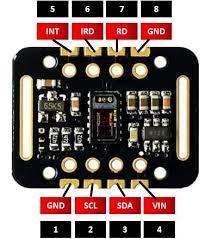
Để chứng minh tính thực tiễn của tín hiệu PPG đã được xử lý, tôi triển khai một hệ thống đo tín hiệu IR từ cảm biến MAX30102 kèm theo dữ liệu gia tốc và con quay hồi chuyển từ MPU6050, điều khiển bởi ESP32-C3, nhằm phân biệt trạng thái ngủ hay thức của người dùng. Quy trình gồm các bước chính sau.

## Giới thiệu phần cứng

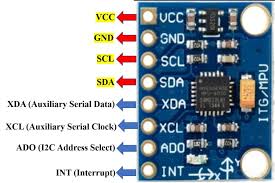
* **ESP32**: Đảm nhiệm vai trò bộ điều khiển nhúng, có khả năng kết nối Wi-Fi, thu thập dữ liệu từ hai cảm biến và gửi về server.



* **MAX30102**: Cảm biến quang học tích hợp LED hồng ngoại (IR) và photodiode, đo tín hiệu PPG IR phản ánh thay đổi lưu lượng máu.



* **MPU6050**: Cảm biến IMU 6-trục, thu thập gia tốc (accelerometer) và con quay hồi chuyển (gyroscope) để phát hiện cử động và tư thế người dùng – đặc biệt hữu ích khi phân biệt trạng thái ngủ (ít chuyển động) và thức (nhiều chuyển động).



## Thiết lập thu thập dữ liệu

* **Kết nối phần cứng**: MAX30102 và MPU6050 đều giao tiếp qua I²C với ESP32.
* **Lập trình ESP32**: Sử dụng Arduino-IDE hoặc PlatformIO để viết firmware:
  + Cấu hình I²C, khởi tạo cảm biến.
  + Đọc mẫu PPG IR và các trục gia tốc, con quay hồi chuyển mỗi 50–100 ms.
  + Gói gọn thành JSON hoặc nhị phân, gửi qua Wi-Fi tới server Python theo giao thức HTTP POST hoặc WebSocket.
* **Đồng bộ thời gian**: Mỗi packet dữ liệu kèm timestamp để dễ ghép nối với nhãn trạng thái ngủ/​thức khi thu thập nhãn ngoài.

## Xây dựng server python

## Tiền xử lý và trích xuất đặc trưng

Trước khi huấn luyện, dữ liệu thô cần:

* **Lọc nhiễu PPG**: Tương tự bước 3, dùng band-pass [0.5–8] Hz, loại bỏ drift và nhấp nhô cao tần.
* **Tách sóng**: Chuyển tín hiệu IR về đoạn cố định (ví dụ mỗi cửa sổ 5 s với overlap 50%).
* **Tính đặc trưng**: Với mỗi cửa sổ:
  + **PPG IR**: tính HR trung bình, biên độ đỉnh–đáy, RMS, entropy.
  + **Gia tốc/gyro**: tính phương sai, năng lượng chuyển động, số lần vượt ngưỡng.

**Chuẩn hóa**: Mỗi đặc trưng chuẩn hóa MinMax hoặc z-score để đưa về cùng phạm vi.

## Huấn luyện mô hình SVM

Với dữ liệu đặc trưng và nhãn ngủ (0) / thức (1):

* **Chọn kernel**: Thường dùng RBF hoặc polynomial; thử nghiệm cross‐validation để chọn tham số C và gamma.
* **Đào tạo**: Sử dụng sklearn.svm.SVC, chia tập train/test theo tỉ lệ 80/20; áp dụng stratified sampling để cân bằng nhãn.
* **Đánh giá**: Tính độ chính xác (accuracy), F1‐score, ma trận nhầm lẫn (confusion matrix) trên tập test.
* **Tinh chỉnh**: Dùng GridSearchCV hoặc RandomizedSearchCV để tối ưu hyperparameters.

## Kết quả và triển vọng

* **Kết quả ban đầu**: Mô hình SVM với kernel RBF đạt accuracy ~ 90–93%, F1‐score ~ 0.92 trên tập test, chứng tỏ các đặc trưng PPG IR kết hợp chuyển động đủ mạnh để phân biệt trạng thái ngủ/thức.
* **Triển vọng mở rộng**:
  + Thử nghiệm mô hình khác (Random Forest, LSTM) tận dụng chuỗi thời gian.
  + Bổ sung cảm biến môi trường (nhiệt độ, ánh sáng) để cải thiện độ chính xác.
  + Tích hợp thực tế vào thiết bị đeo tay hoặc nệm thông minh để giám sát giấc ngủ tự động.

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

# PHỤ LỤC VÀ SOURCE CODE

# TÀI LIỆU THAM KHẢO

<https://ieeexplore.ieee.org/document/9699075>