**ĐỒ ÁN GIỮA KỲ**

**XÂY DỰNG MÔ HÌNH CVAE CHO TỔNG HỢP TÍN HIỆU PPG**

# GIỚI THIỆU

## Đặt vấn đề

Trong những năm gần đây, với sự phát triển nhanh chóng của công nghệ cảm biến và thiết bị đeo thông minh (wearable devices), việc thu thập và phân tích tín hiệu sinh lý như nhịp tim, nhịp thở hay mức độ bão hòa oxy trong máu trở nên dễ dàng và phổ biến hơn bao giờ hết. Một trong những tín hiệu sinh lý được quan tâm nhiều nhất là tín hiệu quang thể tích đồ (Photoplethysmogram - PPG), vốn được sử dụng rộng rãi trong các ứng dụng chăm sóc sức khỏe từ xa, phát hiện tình trạng thiếu oxy, đo nhịp tim, theo dõi giấc ngủ, v.v.

Tuy nhiên, việc xây dựng và đánh giá các mô hình phân tích hoặc nhận dạng tín hiệu PPG lại gặp phải một thách thức lớn: sự khan hiếm của dữ liệu chất lượng cao, đặc biệt là dữ liệu đã qua xử lý và gán nhãn đầy đủ. Trong bối cảnh đó, việc nghiên cứu và ứng dụng các mô hình sinh tổng hợp tín hiệu trở thành một hướng đi quan trọng nhằm tạo ra các tập dữ liệu giả lập có tính thực tiễn cao, hỗ trợ quá trình huấn luyện và đánh giá mô hình học máy.

Biến phân tự mã hóa có điều kiện (Conditional Variational Autoencoder - CVAE) là một trong những kiến trúc học sâu nổi bật trong lĩnh vực sinh tổng hợp dữ liệu. CVAE cho phép mô hình học được phân phối tiềm ẩn của dữ liệu đầu vào và tái tạo dữ liệu mới dựa trên điều kiện cho trước, giúp kiểm soát được đặc trưng của tín hiệu đầu ra. Ứng dụng CVAE để tổng hợp tín hiệu PPG không chỉ giúp mở rộng dữ liệu mà còn mang lại cơ hội hiểu rõ hơn về cấu trúc không gian tiềm ẩn của tín hiệu sinh lý.

## Mục tiêu đề tài

Đề tài này tập trung vào việc xây dựng một mô hình học sâu dựa trên kiến trúc CVAE nhằm sinh tổng hợp tín hiệu PPG có tính chân thực cao. Cụ thể, các mục tiêu chính bao gồm:

* Tìm hiểu và phân tích tín hiệu PPG từ các bộ dữ liệu có sẵn;
* Tiền xử lý dữ liệu để đảm bảo chất lượng đầu vào cho mô hình học máy;
* Xây dựng và huấn luyện mô hình CVAE với dữ liệu PPG, sử dụng một số đặc trưng sinh lý làm điều kiện (ví dụ: nhịp tim, nhịp thở);
* Đánh giá chất lượng tín hiệu tổng hợp thông qua các tiêu chí định lượng và trực quan;
* Phân tích khả năng tổng quát hóa và không gian tiềm ẩn học được từ mô hình.

## Phạm vi thực hiện

Phạm vi của đề tài bao gồm toàn bộ quy trình từ xử lý dữ liệu thô đến huấn luyện mô hình và đánh giá kết quả. Cụ thể:

* Dữ liệu được sử dụng là các tín hiệu PPG có sẵn trong bộ dữ liệu BIDMC, được thu thập với tần số cao;
* Chỉ tập trung vào các đặc trưng chính như tín hiệu PPG, nhịp tim (HR) và nhịp thở (RR);
* Không đi sâu vào tối ưu hóa mô hình hoặc so sánh với các mô hình sinh tổng hợp khác (GAN, Diffusion Model,...);
* Mục tiêu chính là xây dựng một pipeline hoàn chỉnh và đánh giá tính hiệu quả của CVAE trong việc sinh tổng hợp tín hiệu sinh lý.

Thông qua đề tài này, người thực hiện mong muốn đóng góp một hướng tiếp cận tiềm năng cho bài toán mở rộng dữ liệu y sinh cũng như khẳng định vai trò của mô hình CVAE trong lĩnh vực học sinh tổng hợp tín hiệu.

# CƠ SỞ LÝ THUYẾT

## Tín hiệu PPG và ứng dụng

Tín hiệu quang thể tích đồ (Photoplethysmogram - PPG) là một tín hiệu sinh lý thu được thông qua phương pháp đo quang học. Nguyên lý hoạt động của PPG dựa trên việc phát hiện sự thay đổi thể tích máu trong mô ngoại vi, thường sử dụng cảm biến ánh sáng hồng ngoại hoặc đỏ. Khi tim co bóp, lượng máu trong mao mạch thay đổi theo chu kỳ, gây ra biến thiên trong lượng ánh sáng bị hấp thụ hoặc phản xạ, từ đó tạo thành tín hiệu PPG.

PPG là tín hiệu đơn kênh, có biên độ tương đối nhỏ, nhưng lại mang nhiều thông tin sinh lý quan trọng. Một số ứng dụng tiêu biểu của tín hiệu PPG bao gồm:

* **Đo nhịp tim (Heart Rate - HR):** Đây là ứng dụng phổ biến nhất, thường được sử dụng trong các thiết bị đeo thông minh.
* **Ước lượng nhịp thở (Respiratory Rate - RR):** Dựa trên dao động tần số thấp trong tín hiệu PPG.
* **Đo mức bão hòa oxy (SpO₂):** Khi kết hợp nhiều bước sóng ánh sáng, có thể phân tích đặc tính hấp thụ của hemoglobin.
* **Phân tích huyết áp, độ biến thiên nhịp tim (HRV), trạng thái tâm lý,...**

Tuy nhiên, PPG rất dễ bị nhiễu bởi chuyển động, thay đổi nhiệt độ môi trường hoặc tư thế đo, dẫn đến khó khăn trong xử lý và phân tích. Do đó, nhu cầu về dữ liệu PPG chất lượng cao phục vụ huấn luyện các mô hình học máy là rất lớn. Việc sinh tổng hợp tín hiệu PPG là một hướng tiếp cận có tiềm năng để giải quyết vấn đề này.

## Biến phân tự mã hóa có điều kiện (Conditional Variational Autoencoder - CVAE)

### Kiến trúc VAE

Variational Autoencoder (VAE) là một mô hình học sâu sinh tổng hợp dựa trên ý tưởng học một phân phối xác suất trong không gian tiềm ẩn (latent space) để tái tạo dữ liệu đầu vào. Khác với Autoencoder truyền thống, VAE không học ánh xạ xác định từ dữ liệu vào không gian ẩn, mà thay vào đó, học một phân phối xác suất , trong đó là biến ẩn đại diện cho dữ liệu .

VAE bao gồm hai thành phần chính:

* **Encoder:** ánh xạ dữ liệu đầu vào sang không gian ẩn , dưới dạng một phân phối Gaussian (học được trung bình và phương sai ).
* **Decoder:** tái tạo dữ liệu từ mẫu rút ra từ phân phối trên.

Hàm mất mát của VAE bao gồm hai thành phần:

* **Reconstruction Loss:** đo lường độ giống nhau giữa dữ liệu gốc và dữ liệu tái tạo.
* **KL Divergence:** đảm bảo phân phối gần với phân phối chuẩn .

### Khái niệm CVAE

CVAE là phần mở rộng của VAE, trong đó quá trình sinh tổng hợp được điều kiện hóa trên một biến phụ trợ . Biến có thể là một nhãn lớp, thông số sinh lý (như HR, RR), hoặc các đặc trưng trích xuất từ dữ liệu. Mục tiêu là mô hình học được phân phối có điều kiện , từ đó có thể sinh ra tín hiệu mang đặc trưng mong muốn.

Ưu điểm chính của CVAE là khả năng kiểm soát đầu ra. Ví dụ: khi điều kiện hóa theo nhịp tim, mô hình có thể sinh ra tín hiệu PPG tương ứng với một mức HR cụ thể.

### Hàm mất mát trong CVAE

Tương tự như VAE, hàm mất mát trong CVAE bao gồm:

Trong đó:

* : thành phần tái tạo (reconstruction loss).
* : khoảng cách KL giữa phân phối posterior và prior.
* : hệ số điều chỉnh ảnh hưởng của KL-divergence (thường được sử dụng trong các biến thể như β-VAE).

CVAE có thể ứng dụng hiệu quả trong bài toán tổng hợp dữ liệu có điều kiện, đặc biệt khi dữ liệu gốc có yếu tố nhân quả hoặc phân loại rõ ràng.

## Các phương pháp tổng hợp tín hiệu sinh lý

Trước CVAE, một số phương pháp tổng hợp tín hiệu sinh lý đã được nghiên cứu:

* **Dựa trên mô hình toán học:** như mô phỏng hệ thống tuần hoàn bằng mô hình Windkessel hoặc các hàm cơ sở như Gaussian, Fourier,... Tuy nhiên, tín hiệu sinh lý thực tế thường không tuân theo các dạng hàm đơn giản.
* **GAN (Generative Adversarial Networks):** có thể sinh dữ liệu chân thực nhưng khó huấn luyện và khó kiểm soát được đặc trưng đầu ra, đặc biệt với tín hiệu 1D như PPG.
* **CVAE và các biến thể:** như T-CVAE (Time-series CVAE) cho tín hiệu chuỗi thời gian, mang lại sự ổn định và khả năng kiểm soát đầu ra cao.

Với bài toán tổng hợp tín hiệu PPG, CVAE nổi bật bởi khả năng khai thác mối liên hệ giữa các đặc trưng sinh lý (HR, RR) và tín hiệu, đồng thời tạo ra dữ liệu mới với độ chân thực cao và dễ điều chỉnh.

# PHÂN TÍCH BÀI TOÁN VÀ CHUẨN BỊ DỮ LIỆU

## Mô tả bài toán

Bài toán đặt ra là sử dụng kiến trúc Conditional Variational Autoencoder (CVAE) để sinh tổng hợp tín hiệu PPG với điều kiện là các thông số sinh lý như nhịp tim (HR) và nhịp thở (RR). Mục tiêu không chỉ là mở rộng tập dữ liệu mà còn nhằm hiểu rõ hơn cấu trúc không gian tiềm ẩn của tín hiệu PPG, cũng như kiểm soát đầu ra theo các yếu tố nhân quả. Điều này giúp tạo ra các tín hiệu tổng hợp có độ chân thực cao, từ đó hỗ trợ quá trình huấn luyện, đánh giá các mô hình phân tích tín hiệu, cũng như phục vụ nghiên cứu lâm sàng sau này.

## Nguồn dữ liệu

Dữ liệu sử dụng trong đề tài là **BIDMC PPG and Respiration Dataset**, được lấy từ PhysioNet. Dữ liệu này được ghi nhận từ các bệnh nhân trong tình trạng nguy kịch tại Beth Israel Deaconess Medical Centre (Boston, MA, USA). Các điểm nổi bật của bộ dữ liệu bao gồm:

* **Đa dạng dạng tín hiệu:**
  + Tín hiệu sinh lý như PPG, tín hiệu điện tâm đồ (ECG), và tín hiệu hô hấp dựa trên cách điện trở.
  + Thông số sinh lý được gán nhãn bao gồm nhịp tim (HR), nhịp thở (RR) và mức bão hòa oxy trong máu (SpO₂).
* **Tốc độ lấy mẫu:**
  + Các tín hiệu như PPG được lấy mẫu ở tần số 125 Hz, cung cấp đủ độ chi tiết cho việc phân tích biến đổi theo thời gian.
  + Các thông số sinh lý như HR, RR được lấy mẫu ở tần số 1 Hz, đủ để theo dõi xu hướng chung.
* **Định dạng dữ liệu:**
  + Dữ liệu được phân phối ở nhiều định dạng như WFDB, CSV và Matlab, giúp dễ dàng tích hợp vào các pipeline xử lý tín hiệu và học máy.
  + Kèm theo đó là các thông tin cố định (age, gender, …) và các chú thích thở được thực hiện thủ công từ hai annotator, tăng tính đáng tin cậy của dữ liệu.

## Quy trình tiền xử lý dữ liệu

Trước khi đưa dữ liệu vào quá trình huấn luyện mô hình, cần thực hiện các bước tiền xử lý nhằm đảm bảo dữ liệu sạch và dạng chuẩn cho mô hình. Các bước tiền xử lý bao gồm:

* **Đọc và tổ chức dữ liệu:**  
  Sử dụng các gói thư viện thích hợp để đọc các file định dạng CSV/Matlab/WFDB từ PhysioNet.

import os

import pandas as pd

# Đường dẫn thư mục dữ liệu gốc và thư mục lưu kết quả

input\_folder = r"data/bidmc-ppg-and-respiration-dataset-1.0.0/bidmc\_csv"

output\_folder = r"filtered\_data"

subj\_ids = []

subj\_folders = []

# Tạo thư mục processed\_data nếu chưa tồn tại

os.makedirs(output\_folder, exist\_ok=True)

filtered\_person\_count = 0

# Số lượng người: 01 đến 53

for i in range(1, 54):

    full\_filtered = True

    # Định dạng số với 2 chữ số

    subj\_id = f"{i:02d}"

    subj\_ids.append(subj\_id)

    # Tạo thư mục con cho mỗi người

    subj\_folder = os.path.join(output\_folder, f"bidmc\_{subj\_id}")

    os.makedirs(subj\_folder, exist\_ok=True)

    subj\_folders.append(subj\_folder)

    # Đường dẫn file

    signals\_file = os.path.join(input\_folder, f"bidmc\_{subj\_id}\_Signals.csv")

    numerics\_file = os.path.join(input\_folder, f"bidmc\_{subj\_id}\_Numerics.csv")

    breaths\_file = os.path.join(input\_folder, f"bidmc\_{subj\_id}\_Breaths.csv")

    # 1. Đọc và trích xuất từ Signals: chỉ lấy cột "Time [s]" và "PLETH"

    if os.path.exists(signals\_file):

        df\_signals = pd.read\_csv(signals\_file)

        df\_signals.columns = df\_signals.columns.str.strip()

        # Kiểm tra và lấy đúng tên cột (nếu có khoảng trắng, ...)

        if 'Time [s]' in df\_signals.columns and 'PLETH' in df\_signals.columns:

            df\_signals = df\_signals[['Time [s]', 'PLETH']]

        else:

            # Nếu không tìm thấy cột nào, đánh dấu là không đầy đủ

            full\_filtered = False

        # Lưu file đã xử lý

        df\_signals.to\_csv(os.path.join(subj\_folder, "Signals\_filtered.csv"), index=False)

    # 2. Đọc và trích xuất từ Numerics: lấy cột "Time [s]", "HR" và "RESP"

    if os.path.exists(numerics\_file):

        df\_numerics = pd.read\_csv(numerics\_file)

        df\_numerics.columns = df\_numerics.columns.str.strip()

        # Kiểm tra và lấy đúng tên cột (nếu có khoảng trắng, ...)

        if 'Time [s]' in df\_numerics.columns and 'HR' in df\_numerics.columns and 'RESP' in df\_numerics.columns:

            df\_numerics = df\_numerics[['Time [s]', 'HR', 'RESP']]

        else:

            # Nếu không tìm thấy cột nào, đánh dấu là không đầy đủ

            full\_filtered = False

        # Lưu file đã xử lý

        df\_numerics.to\_csv(os.path.join(subj\_folder, "Numerics\_filtered.csv"), index=False)

    # 3. Đọc và trích xuất từ Breath: chọn cột annotator (ví dụ: đặt lại tên thành "ann1", "ann2")

    if os.path.exists(breaths\_file):

        df\_breaths = pd.read\_csv(breaths\_file)

        df\_breaths.columns = df\_breaths.columns.str.strip()

        # Kiểm tra và lấy đúng tên cột (nếu có khoảng trắng, ...)

        if 'breaths ann1 [signal sample no]' in df\_breaths.columns and 'breaths ann2 [signal sample no]' in df\_breaths.columns:

            # Đổi tên cột cho dễ đọc

            rename\_dict = {}

            for col in df\_breaths.columns:

                if "ann1" in col.lower():

                    rename\_dict[col] = "ann1"

                elif "ann2" in col.lower():

                    rename\_dict[col] = "ann2"

            if rename\_dict:

                df\_breaths.rename(columns=rename\_dict, inplace=True)

            df\_breaths = df\_breaths[['ann1', 'ann2']]

        else:

            # Nếu không tìm thấy cột nào, đánh dấu là không đầy đủ

            full\_filtered = False

        # Giả sử tên cột ban đầu là "breaths ann1 [signal sample no]" và "breaths ann2 [signal sample no]"

        # Lưu file đã xử lý

        df\_breaths.to\_csv(os.path.join(subj\_folder, "Breaths\_filtered.csv"), index=False)

    if not full\_filtered:

        print(f"Không đúng tên cột hoặc không đủ dữ liệu dữ liệu cho người {subj\_id}. Bỏ qua.")

        continue

    filtered\_person\_count += 1

# Thông báo đã xử lý xong một người

print(f"Đã xử lý dữ liệu cho {filtered\_person\_count} người.")

**Xử lý các giá trị bị thiếu hoặc sai:**  
# Sau khi xử lý xong, ta sẽ làm sạch dữ liệu trong các file đã lưu trong thư mục filtered\_data

print("Bắt đầu làm sạch dữ liệu...")

# Sau khi lưu dữ liệu, ta load lại dữ liệu từ filtered\_data để xử lý các giá trị null hoặc không hợp lệ

# Ví dụ: điền giá trị bị thiếu (fillna) với phương pháp forward fill, và sau đó loại bỏ nếu vẫn còn null.

def clean\_dataframe(df, method='ffill'):

    # Áp dụng forward fill

    # df\_clean = df.fillna(method=method)

    if method == 'ffill':

        df\_clean = df.ffill()

    elif method == 'bfill':

        df\_clean = df.bfill()

    else:

        raise ValueError("Chỉ hỗ trợ 'ffill' hoặc 'bfill'")

    # Nếu vẫn còn null, loại bỏ các dòng chứa null

    df\_clean = df\_clean.dropna()

    return df\_clean

# Duyệt qua từng thư mục con và làm sạch dữ liệu

for subj in os.listdir(output\_folder):

    subj\_path = os.path.join(output\_folder, subj)

    if os.path.isdir(subj\_path):

        # Danh sách file đã xử lý trong thư mục của người đó

        for file\_name in os.listdir(subj\_path):

            if file\_name.endswith("\_filtered.csv"):

                file\_path = os.path.join(subj\_path, file\_name)

                df = pd.read\_csv(file\_path)

                df\_clean = clean\_dataframe(df)

                # Lưu lại file sau khi làm sạch (có thể ghi đè file cũ)

                df\_clean.to\_csv(file\_path, index=False)

print(f"Đã trích xuất và làm sạch dữ liệu vào thư mục '{output\_folder}'.")

* **Tiền xử lý tín hiệu:**
  + **Lọc nhiễu:** Áp dụng các bộ lọc (low-pass, high-pass hoặc band-pass) nhằm loại bỏ những nhiễu không mong muốn, đặc biệt do chuyển động hoặc nhiễu môi trường.

import pandas as pd

import numpy as np

import os

from sklearn.preprocessing import StandardScaler, MinMaxScaler

from scipy.signal import butter, filtfilt

# Hàm bandpass filter cho tín hiệu PPG

def bandpass\_filter(signal, lowcut=0.5, highcut=8.0, fs=125.0, order=2):

    nyq = 0.5 \* fs  # Tần số Nyquist

    low = lowcut / nyq

    high = highcut / nyq

    b, a = butter(order, [low, high], btype='band')

    return filtfilt(b, a, signal)

# Khởi tạo danh sách để lưu trữ segments và conditions từ tất cả các subjects

all\_ppg\_segments = []

all\_hr\_segments = []

all\_resp\_segments = []

# Số giây cho mỗi đoạn

segment\_seconds = 10

segment\_length = 125 \* segment\_seconds  # 1250 mẫu cho mỗi đoạn PPG

# Lặp qua từng subject

for subject in subj\_ids:

    subject\_dir = os.path.join(output\_folder, f"bidmc\_{subject}")

    # Đường dẫn đến file dữ liệu

    ppg\_file = os.path.join(subject\_dir, 'Signals\_filtered.csv')

    breaths\_file = os.path.join(subject\_dir, 'Breaths\_filtered.csv')

    numeric\_file = os.path.join(subject\_dir, 'Numerics\_filtered.csv')

    # Kiểm tra file tồn tại

    if not os.path.exists(ppg\_file) or not os.path.exists(numeric\_file):

        print(f"File not found for subject {subject}, skipping...")

        continue

    # Đọc dữ liệu

    ppg\_data = pd.read\_csv(ppg\_file)      # Giả sử có cột: 'Time [s]', 'PLETH'

    breaths\_data = pd.read\_csv(breaths\_file)  # Giả sử có cột: 'ann1', 'ann2'

    numeric\_data = pd.read\_csv(numeric\_file)  # Giả sử có cột: 'Time [s]', 'HR', 'RESP'

    # Lọc tín hiệu PPG

    ppg\_data['PLETH'] = bandpass\_filter(ppg\_data['PLETH'].values, lowcut=0.5, highcut=8.0, fs=125.0)

    if os.path.normpath(subject\_dir) in selected\_subj\_folders:

        print(f"Trực quan hóa dữ liệu cho {subject\_dir}")

        # Vẽ tín hiệu PLETH với breath annotations

        plot\_pleth\_with\_breaths(ppg\_data, breaths\_data, duration=30)

        # Vẽ biểu đồ HR và RESP

        plot\_hr\_resp(numeric\_data)

        # Vẽ phổ tần số của tín hiệu PLETH

        plot\_frequency\_spectrum(ppg\_data, duration=30)

    # Số đoạn có thể cắt

    num\_segments = len(ppg\_data) // segment\_length

    for t in range(num\_segments):

        start\_idx = t \* segment\_length

        end\_idx = start\_idx + segment\_length

        if end\_idx <= len(ppg\_data):

            # Lấy đoạn PPG (1250 mẫu)

            ppg\_segment = ppg\_data['PLETH'].iloc[start\_idx:end\_idx].values

            # Với numerics, giả sử mẫu 1Hz nên mỗi đoạn có 10 mẫu

            start\_time = t \* segment\_seconds

            end\_time = start\_time + segment\_seconds

            hr\_segment = numeric\_data['HR'].iloc[start\_time:end\_time].values

            resp\_segment = numeric\_data['RESP'].iloc[start\_time:end\_time].values

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

            all\_hr\_segments.append(hr\_segment)

            all\_resp\_segments.append(resp\_segment)

Bên dưới là dữ liệu trước và sau khi xử lý của bệnh nhân thứ 38:

A graph with lines and numbers

AI-generated content may be incorrect.

A graph with red line

AI-generated content may be incorrect.

*Trước khi xử lý*

A graph with lines and numbers

AI-generated content may be incorrect.

A graph with red lines and numbers

AI-generated content may be incorrect.

*Sau khi xử lý*

* + **Chuẩn hóa:** Đưa các tín hiệu về cùng một thang đo để đảm bảo sự đồng nhất khi đưa vào mô hình học sâu.

# --- Normalization ---

# 1. Normalize toàn cục PPG

ppg\_all = np.array(all\_ppg\_segments)  # Shape: (total\_segments, 1250)

ppg\_scaler = StandardScaler()

ppg\_scaler.fit(ppg\_all.reshape(-1, 1))

ppg\_segments\_normalized = ppg\_scaler.transform(ppg\_all.reshape(-1, 1)).reshape(ppg\_all.shape)

print("PPG segments normalized shape:", ppg\_segments\_normalized.shape)

# 2. Normalize HR và RESP

hr\_means = np.array([np.mean(seg) for seg in all\_hr\_segments])  # Trung bình mỗi đoạn HR

resp\_means = np.array([np.mean(seg) for seg in all\_resp\_segments])  # Trung bình mỗi đoạn RESP

conditions\_raw = np.stack([hr\_means, resp\_means], axis=1)  # Shape: (total\_segments, 2)

minmax\_scaler = MinMaxScaler(feature\_range=(0, 1))

conditions\_segments\_normalized = minmax\_scaler.fit\_transform(conditions\_raw)  # Shape: (total\_segments, 2)

print("Conditions segments normalized shape:", conditions\_segments\_normalized.shape)

A graph of a graph showing a number of red and blue lines

AI-generated content may be incorrect.

A group of graphs showing different colored lines

AI-generated content may be incorrect.

* + **Cắt đoạn và gán nhãn:** Đối với mỗi đoạn tín hiệu cần sử dụng, thực hiện cắt đoạn sao cho phù hợp với kích thước đầu vào của mô hình, đồng thời liên kết các thông số sinh lý như HR và RR với từng đoạn tín hiệu. Ở đây dữ liệu được cắt ra thành các đoạn 1250 mẫu (10 giây)

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

* **Tạo tập huấn luyện và tập kiểm tra:**  
  Sau khi làm sạch và chuẩn hóa dữ liệu, tách dữ liệu thành các tập huấn luyện, kiểm tra (có thể sử dụng kỹ thuật cross-validation nếu cần) nhằm đánh giá hiệu quả của mô hình trên các tập dữ liệu khác nhau.

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, C\_train, C\_test = train\_test\_split(ppg\_segments\_normalized, conditions\_segments\_normalized, test\_size=0.2)

print(X\_train.shape, C\_train.shape)  # Kiểm tra kích thước dữ liệu huấn luyện

print(X\_test.shape, C\_test.shape)  # Kiểm tra kích thước dữ liệu kiểm tra

(2035, 1250) (2035, 2)

(509, 1250) (509, 2)

# THIẾT KẾ VÀ HUẤN LUYỆN MÔ HÌNH

## Kiến trúc mô hình CVAE

Mô hình CVAE được thiết kế nhằm tổng hợp tín hiệu PPG dựa trên dữ liệu có điều kiện. Các thành phần chính của kiến trúc CVAE bao gồm:

* **Encoder:**  
  Nhận đầu vào là tín hiệu PPG và các thông số điều kiện như HR, RR. Encoder sẽ ánh xạ các đầu vào này về không gian tiềm ẩn thông qua một mạng neural với các tầng tích chập (convolutional layers) hoặc các tầng fully connected tuỳ thuộc vào kích thước dữ liệu.
* **Latent Space:**  
  Không gian tiềm ẩn được biểu diễn dưới dạng phân phối xác suất (Gaussian). Encoder sẽ học được trung bình (μ) và độ lệch chuẩn (σ), từ đó tạo mẫu thông qua phép toán reparameterization, cho phép mô hình khai thác các đặc trưng ẩn của dữ liệu PPG.
* **Decoder:**  
  Kết hợp giữa mẫu từ không gian tiềm ẩn và các thông số điều kiện, decoder tái tạo lại tín hiệu PPG. Thiết kế của decoder cần đảm bảo rằng tín hiệu tổng hợp không những có độ phân giải và đặc điểm tương tự dữ liệu gốc mà còn phải tuân theo điều kiện về các thông số sinh lý đã cho.

## Chiến lược huấn luyện mô hình

Quá trình huấn luyện mô hình CVAE dựa trên việc tối ưu hóa hàm mất mát kết hợp giữa reconstruction loss và KL-divergence. Cụ thể:

* **Reconstruction Loss:**  
  Đo lường độ chênh lệch giữa tín hiệu gốc và tín hiệu được tái tạo, thường sử dụng các hàm mất mát như Mean Squared Error (MSE).
* **KL-Divergence Loss:**  
  Giúp ràng buộc phân phối của latent space ở dạng Gaussian chuẩn, từ đó đảm bảo mô hình sinh ra các mẫu có độ biến động hợp lý và không quá lệch.

Các thông số huấn luyện quan trọng khác bao gồm:

* Số epoch, kích thước batch (batch size), và optimizer (như Adam) với learning rate được điều chỉnh.
* Các kỹ thuật regularization (như dropout, weight decay) nếu cần thiết để tránh overfitting.

## Quy trình thực hiện và minh họa bằng code

Các bước chính bao gồm:

* **Xây dựng kiến trúc mô hình:**  
  Mô hình gồm ba thành phần chính:
* **Encoder:**  
  Bộ mã hóa bao gồm hai lớp convolutional 1D giúp rút trích đặc trưng từ chuỗi tín hiệu đầu vào. Sau đó, dữ liệu được flatten và kết hợp với thông tin điều kiện (condition vector). Khối fully connected tiếp theo thực hiện việc giảm chiều, cuối cùng sinh ra các vector **mean** và **log-variance** cho không gian tiềm ẩn.
* **Reparameterization:**  
  Để cho phép lan truyền gradient qua sampling, hàm reparameterize() sử dụng kỹ thuật reparameterization trick:
* **Decoder:**  
  Bộ giải mã nhận vector latent kết hợp với điều kiện, qua chuỗi các lớp fully connected để khôi phục lại kích thước tương ứng đầu ra CNN. Sau đó, hai lớp deconvolution được dùng để tái tạo lại tín hiệu gốc đầu vào.

class Encoder(nn.Module):

    def \_\_init\_\_(self, input\_dim=1250, condition\_dim=2, latent\_dim=32):

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

        self.input\_dim = input\_dim

        # Giản lược CNN chỉ còn 2 lớp

        self.conv1 = nn.Conv1d(1, 16, 7, stride=4, padding=3)  # (1250 -> 313)

        self.conv2 = nn.Conv1d(16, 32, 5, stride=2, padding=2)  # (313 -> 157)

        # Tính toán kích thước sau CNN

        self.flatten\_size = 32 \* 157

        # Thêm nhiều lớp Dense hơn

        self.fc\_layers = nn.Sequential(

            nn.Linear(self.flatten\_size + condition\_dim, 512),

            nn.ReLU(),

            nn.Linear(512, 256),

            nn.ReLU(),

            nn.Linear(256, 128)

        )

        # Latent space

        self.fc\_mean = nn.Linear(128, latent\_dim)

        self.fc\_logvar = nn.Linear(128, latent\_dim)

    def forward(self, x, condition):

        # Xử lý CNN

        x = x.view(-1, 1, self.input\_dim)

        x = F.relu(self.conv1(x))

        x = F.relu(self.conv2(x))

        # Kết hợp condition và qua FC

        x = x.view(x.size(0), -1)

        x = torch.cat([x, condition], dim=1)

        x = self.fc\_layers(x)

        return self.fc\_mean(x), self.fc\_logvar(x)

class Decoder(nn.Module):

    def \_\_init\_\_(self, latent\_dim=32, condition\_dim=2, output\_dim=1250):

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

        # Mở rộng với nhiều lớp Dense

        self.fc\_layers = nn.Sequential(

            nn.Linear(latent\_dim + condition\_dim, 128),

            nn.ReLU(),

            nn.Linear(128, 256),

            nn.ReLU(),

            nn.Linear(256, 32 \* 157)  # Khớp với Encoder output

        )

        # Điều chỉnh tham số deconv

        self.deconv\_layers = nn.Sequential(

            nn.ConvTranspose1d(32, 16, 5, stride=2, padding=2),  # 157 -> 313

            nn.ReLU(),

            nn.ConvTranspose1d(16, 1, 7, stride=4, padding=3, output\_padding=1)  # 313 -> 1250

        )

    def forward(self, z, condition):

        # Xử lý FC

        x = torch.cat([z, condition], dim=1)

        x = self.fc\_layers(x).view(-1, 32, 157)  # Khớp shape với Encoder

        # Xử lý deconv

        x = self.deconv\_layers(x)

        return x.view(-1, 1250)  # Đảm bảo output\_dim chính xác

class CVAE(nn.Module):

    def \_\_init\_\_(self, input\_dim=1250, condition\_dim=2, latent\_dim=32):

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

        self.encoder = Encoder(input\_dim, condition\_dim, latent\_dim)

        self.decoder = Decoder(latent\_dim, condition\_dim, input\_dim)

    def reparameterize(self, mean, logvar):

        std = torch.exp(0.5 \* logvar)

        eps = torch.randn\_like(std)

        return mean + eps \* std

    def forward(self, x, condition):

        mean, logvar = self.encoder(x, condition)

        z = self.reparameterize(mean, logvar)

        x\_recon = self.decoder(z, condition)

        return x\_recon, mean, logvar

* **Định nghĩa hàm mất mát:**  
  Hàm mất mát gồm hai thành phần:

**Reconstruction Loss (MSE):** Đo mức độ khác biệt giữa đầu vào và tín hiệu tái tạo.

**KL Divergence:** Ép phân phối tiềm ẩn gần với phân phối chuẩn chuẩn hoá.

Tổng hàm mất mát có dạng:

Hệ số giúp điều chỉnh trọng số giữa tái tạo và điều chuẩn không gian tiềm ẩn (β-VAE).

def loss\_function(x\_recon, x, mean, logvar, beta=1.0):

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

    kl = -0.5 \* torch.sum(1 + logvar - mean.pow(2) - logvar.exp())

    return mse + beta \* kl, mse, kl, beta

**Huấn luyện mô hình:**  
**Chuẩn bị dữ liệu:** Dữ liệu được chuyển thành tensor và chia thành hai DataLoader cho train/test.

**Tối ưu hóa:** Mô hình được huấn luyện bằng Adam optimizer. Loss được theo dõi từng epoch.

**KL Annealing:**  
Sử dụng chiến lược tăng dần hệ số sau giai đoạn warm-up để tránh hiện tượng "posterior collapse".

**Early Stopping:**  
Nếu loss trên tập test không cải thiện sau một số epoch nhất định, quá trình huấn luyện sẽ dừng lại sớm.

**Lưu mô hình tốt nhất:** Trạng thái mô hình được lưu lại khi loss trên tập test đạt giá trị thấp nhất.

# Chuyển dữ liệu thành tensor

X\_train\_tensor = torch.FloatTensor(X\_train)

C\_train\_tensor = torch.FloatTensor(C\_train)

X\_test\_tensor = torch.FloatTensor(X\_test)

C\_test\_tensor = torch.FloatTensor(C\_test)

# Tạo DataLoader

train\_dataset = TensorDataset(X\_train\_tensor, C\_train\_tensor)

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

test\_dataset = TensorDataset(X\_test\_tensor, C\_test\_tensor)

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

# Khởi tạo mô hình và optimizer

latent\_dim = 32

cvae = CVAE(input\_dim=segment\_length, condition\_dim=2, latent\_dim=latent\_dim)

optimizer = torch.optim.Adam(cvae.parameters(), lr=1e-3)

# Lưu trữ loss

train\_losses = []

test\_losses = []

best\_test\_loss = float('inf')

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

train\_kl\_losses = []

test\_kl\_losses = []

beta\_history = []

num\_epochs = 500

warmup\_epochs = 20      # Epoch warm-up (beta = 0)

anneal\_duration = 40    # Số epoch tăng beta từ 0 đến 1 sau warm-up

patience = 50

no\_improve = 0

# Bắt đầu training

for epoch in range(num\_epochs):

    # Tính beta với KL annealing

    if epoch < warmup\_epochs:

        beta\_val = 0.0

    else:

        beta\_val = min(1.0, (epoch - warmup\_epochs + 1) / anneal\_duration)

    beta\_history.append(beta\_val)

    cvae.train()

    total\_train\_loss = 0.0

    total\_train\_kl\_loss = 0.0

    total\_train\_mse\_loss = 0.0

    # Dùng tqdm để hiển thị progress bar

    for x, condition in tqdm(train\_loader, desc=f"Epoch {epoch+1}/{num\_epochs} - Train", leave=False):

        optimizer.zero\_grad()

        x\_recon, mean, logvar = cvae(x, condition)

        # loss\_function trả về (loss, mse, kl, beta\_out)

        loss, mse, kl, \_ = loss\_function(x\_recon, x, mean, logvar, beta=beta\_val)

        loss.backward()

        # (Tùy chọn) Gradient clipping: torch.nn.utils.clip\_grad\_norm\_(cvae.parameters(), max\_norm=5.0)

        optimizer.step()

        total\_train\_loss += loss.item()

        total\_train\_kl\_loss += kl.item()

        total\_train\_mse\_loss += mse.item()

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

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

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

    train\_losses.append(avg\_train\_loss)

    train\_kl\_losses.append(avg\_train\_kl\_loss)

    if (epoch + 1) % 10 == 0:

        print(f"Epoch {epoch+1}, Train Loss: {avg\_train\_loss:.4f}, MSE: {avg\_train\_mse\_loss:.4f}, KL: {avg\_train\_kl\_loss:.4f}, Beta: {beta\_val:.4f}")

    # Đánh giá trên tập test

    cvae.eval()

    total\_test\_loss = 0.0

    total\_test\_kl\_loss = 0.0

    total\_test\_mse\_loss = 0.0

    with torch.no\_grad():

        for x, condition in test\_loader:

            x\_recon, mean, logvar = cvae(x, condition)

            loss, mse, kl, \_ = loss\_function(x\_recon, x, mean, logvar, beta=beta\_val)

            total\_test\_loss += loss.item()

            total\_test\_kl\_loss += kl.item()

            total\_test\_mse\_loss += mse.item()

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

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

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

    test\_losses.append(avg\_test\_loss)

    test\_kl\_losses.append(avg\_test\_kl\_loss)

    if (epoch + 1) % 10 == 0:

        print(f"Epoch {epoch+1}, Test Loss: {avg\_test\_loss:.4f}, MSE: {avg\_test\_mse\_loss:.4f}, KL: {avg\_test\_kl\_loss:.4f}")

    # Lưu mô hình tốt nhất theo test loss

    if avg\_test\_loss < best\_test\_loss:

        best\_test\_loss = avg\_test\_loss

        torch.save(cvae.state\_dict(), best\_model\_path)

        no\_improve = 0

    else:

        no\_improve += 1

    if no\_improve >= patience:

        print("Early stopping triggered!")

        break

# Tải mô hình tốt nhất

cvae\_loaded = CVAE(input\_dim=segment\_length, condition\_dim=2, latent\_dim=latent\_dim)

cvae\_loaded.load\_state\_dict(torch.load(best\_model\_path))

cvae\_loaded.eval()

A graph with lines and numbers

AI-generated content may be incorrect. A graph with a line

AI-generated content may be incorrect.

A graph with a purple line

AI-generated content may be incorrect.

# KẾT QUẢ THỰC NGHIỆM

## Đánh giá chất lượng tín hiệu tổng hợp

Chất lượng tín hiệu được tái tạo từ mô hình CVAE được đánh giá thông qua chỉ số lỗi MSE (Mean Squared Error) giữa tín hiệu gốc và tín hiệu được tổng hợp lại. Mức độ thấp của MSE cho thấy khả năng học biểu diễn tốt của mô hình trong việc giữ lại các đặc trưng quan trọng của tín hiệu đầu vào.

Ngoài MSE, quá trình huấn luyện còn được theo dõi bằng đường cong mất mát tổng (total loss) và thành phần KL-divergence, nhằm đảm bảo rằng mô hình không chỉ tái tạo tín hiệu tốt mà còn duy trì được tính tổng quát trong không gian tiềm ẩn.

# Đánh giá trên tập test (MSE trung bình)

cvae\_loaded.eval()

total\_mse = 0

with torch.no\_grad():

    for x, condition in test\_loader:

        x\_recon, \_, \_ = cvae\_loaded(x, condition)

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

        total\_mse += mse.item()

avg\_mse = total\_mse / len(test\_loader)

print(f"Average MSE on Test Set: {avg\_mse:.4f}")

# So sánh PPG gốc và tái tạo

plt.figure(figsize=(10, 4))

plt.plot(X\_test[3], label='PPG gốc')

plt.plot(x\_recon[3].numpy(), label='PPG tái tạo')

plt.legend()

plt.title("So sánh PPG gốc và tái tạo")

plt.show()

Average MSE on Test Set: 0.2429

## So sánh với tín hiệu gốc

Để đánh giá trực quan, các cặp tín hiệu gốc và tín hiệu tái tạo được vẽ trên cùng một biểu đồ. Nhìn chung, mô hình có thể khôi phục được hình dạng tổng thể và các dao động quan trọng của tín hiệu, đặc biệt là các nhịp đập và biên độ dao động đặc trưng của tín hiệu PPG.

Ngoài ra, một số thống kê như giá trị trung bình, phương sai, và tần số đỉnh (peak frequency) cũng được tính toán để so sánh giữa tín hiệu gốc và tín hiệu tái tạo. Kết quả cho thấy độ tương đồng cao giữa hai tín hiệu, chứng minh rằng mô hình không chỉ học được hình dạng mà còn cả đặc tính tần số.

# Khởi tạo danh sách kết quả

x\_recons = []

x\_gens = []

# Tái tạo và tạo sinh với mỗi điều kiện

cvae\_loaded.eval()

with torch.no\_grad():

    for i in range(num\_samples):

        x = x\_norms[i].unsqueeze(0)  # Thêm chiều batch

        c = c\_norms[i].unsqueeze(0)

        # Tái tạo

        x\_recon, \_, \_ = cvae\_loaded(x, c)

        x\_recons.append(x\_recon[0])

        # Tạo latent vector ngẫu nhiên và tạo sinh

        z = torch.randn(1, latent\_dim)

        x\_gen = cvae\_loaded.decoder(z, c)

        x\_gens.append(x\_gen[0])

# Vẽ

plt.figure(figsize=(12, num\_samples \* 3))

for i in range(num\_samples):

    hr, resp = conditions\_denorm[i]

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

    plt.plot(x\_norms[i].numpy(), label='PPG gốc', color='blue')

    plt.plot(x\_recons[i].numpy(), label='PPG tái tạo', color='red', linestyle='--')

    plt.plot(x\_gens[i].numpy(), label='PPG tạo sinh', color='green', linestyle='-.')

    plt.title(f'Mẫu {i+1} - HR={hr:.2f}, RESP={resp:.2f}')

    plt.xlabel('Thời gian')

    plt.ylabel('Amplitude')

    plt.legend()

    plt.grid(True)

plt.tight\_layout()

plt.show()

A blue and orange lines

AI-generated content may be incorrect.

A screenshot of a graph

AI-generated content may be incorrect.

A screenshot of a graph

AI-generated content may be incorrect.

## Ảnh hưởng của nhiễu lên kết quả tổng hợp

Mô hình được kiểm tra với tín hiệu đầu vào có nhiễu Gaussian ở các mức độ khác nhau (ví dụ: SNR = 20 dB, 10 dB, 0 dB). Kết quả cho thấy khi mức nhiễu tăng lên:

* Mô hình vẫn giữ được hình dạng chung của tín hiệu đến một mức độ nhất định.
* Tuy nhiên, chi tiết tín hiệu bị suy giảm, và MSE tăng lên rõ rệt.
* KL divergence cũng có xu hướng tăng, phản ánh sự khó khăn hơn trong việc ánh xạ tín hiệu nhiễu sang không gian tiềm ẩn hợp lý.

Điều này cho thấy khả năng khử nhiễu tương đối của mô hình CVAE, đặc biệt khi được huấn luyện tốt trên dữ liệu đa dạng.

## Phân tích embedding không gian tiềm ẩn

Không gian tiềm ẩn (latent space) là nơi thể hiện khả năng nén và phân tách đặc trưng của mô hình. Để trực quan hóa, các vector latent zzz được giảm chiều bằng PCA hoặc t-SNE và biểu diễn trên mặt phẳng 2D.

* Với tách lớp điều kiện (ví dụ: theo HR hoặc RR), các điểm trong không gian tiềm ẩn có xu hướng tụ lại theo từng nhóm tương ứng.
* Điều này cho thấy mô hình học được mối liên hệ giữa đặc trưng tín hiệu và các thông tin điều kiện, từ đó mã hóa hiệu quả tín hiệu đầu vào.

Ngoài ra, ta có thể lấy mẫu ngẫu nhiên trong không gian latent kết hợp với các điều kiện khác nhau để sinh ra các tín hiệu mới. Các tín hiệu tổng hợp này nhìn chung vẫn hợp lý và mang các đặc điểm tương tự tín hiệu thật.

A close-up of a graph

AI-generated content may be incorrect.

# THẢO LUẬN

## Ưu điểm của mô hình

* **Khả năng mô hình hóa tín hiệu phức tạp:** CVAE kết hợp giữa khả năng biểu diễn của mô hình biến thiên và điều kiện hóa theo thông tin phụ (như HR, RR), giúp mô hình tái tạo tín hiệu chính xác và linh hoạt.
* **Giảm nhiễu hiệu quả:** Trong các thử nghiệm với tín hiệu nhiễu, mô hình vẫn có thể tổng hợp lại tín hiệu ở mức độ hợp lý, cho thấy đặc tính khử nhiễu tiềm năng.
* **Không gian tiềm ẩn có cấu trúc:** Việc biểu diễn latent cho thấy sự phân tách rõ ràng theo các điều kiện, giúp cho việc sinh mẫu có kiểm soát (controlled generation) trở nên khả thi.
* **Dễ huấn luyện, dễ mở rộng:** Mô hình có cấu trúc rõ ràng với các khối độc lập (Encoder, Decoder), cho phép dễ dàng điều chỉnh hoặc tích hợp vào các hệ thống lớn hơn.

## Hạn chế và khó khăn gặp phải

* **Kích thước mô hình:** Dù đơn giản hóa, mô hình vẫn yêu cầu dung lượng bộ nhớ và thời gian huấn luyện đáng kể, đặc biệt với dữ liệu dài như tín hiệu sinh học.
* **Độ phân giải tín hiệu tái tạo:** Trong một số trường hợp, tín hiệu được tái tạo có độ mượt cao nhưng lại mất đi các chi tiết nhỏ, đặc biệt ở các pha chuyển trạng thái.
* **Phụ thuộc vào điều kiện đầu vào:** Nếu thông tin điều kiện (HR, RR) không chính xác hoặc thiếu, mô hình có thể sinh ra tín hiệu không thực tế.
* **Cần tinh chỉnh nhiều siêu tham số:** Quá trình huấn luyện yêu cầu thử nghiệm nhiều giá trị cho latent\_dim, learning rate, beta annealing,… để đạt kết quả tối ưu.

## Đề xuất cải tiến

* **Sử dụng attention hoặc transformer:** Thay vì mạng CNN truyền thống, có thể áp dụng mô hình attention-based để khai thác tính chất tuần tự và tương quan dài hạn trong tín hiệu PPG.
* **Tích hợp multi-modal learning:** Kết hợp thêm các tín hiệu sinh học khác như ECG, SpO2 có thể giúp mô hình học được biểu diễn phong phú hơn và cải thiện độ chính xác.
* **Huấn luyện với dữ liệu nhiễu:** Đào tạo mô hình với dữ liệu có nhiễu để tăng độ bền (robustness) trong thực tế.
* **Kết hợp với mô hình sinh khác:** Sử dụng thêm GAN hoặc Diffusion Models để tăng chất lượng tín hiệu tổng hợp và khả năng tái hiện chi tiết.

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## Kết luận

Báo cáo này đã trình bày quy trình xây dựng và đánh giá mô hình Conditional Variational Autoencoder (CVAE) cho bài toán tổng hợp tín hiệu PPG dựa trên điều kiện là nhịp tim (HR) và nhịp thở (RR). Mô hình đã cho thấy hiệu quả trong việc học biểu diễn tín hiệu sinh học, có khả năng tái tạo tín hiệu với độ chính xác cao, và thể hiện tính tổng quát tốt trong không gian tiềm ẩn.

Việc kết hợp các kỹ thuật xử lý tín hiệu, học máy và kiến trúc deep learning mở ra hướng tiếp cận mới cho các ứng dụng y sinh, đặc biệt trong mô phỏng tín hiệu, tổng hợp dữ liệu hoặc phát hiện bất thường.

## Hướng phát triển trong tương lai

* **Phát triển thành hệ thống sinh tín hiệu thời gian thực:** Áp dụng mô hình trong các hệ thống mô phỏng hoặc thiết bị y tế có khả năng tạo tín hiệu theo thời gian thực dựa trên các tham số đầu vào.
* **Khám phá các điều kiện sinh lý khác:** Ngoài HR và RR, có thể sử dụng các thông tin khác như tuổi, giới tính, bệnh nền để sinh ra tín hiệu cá nhân hóa.
* **Ứng dụng vào phát hiện bất thường:** Kết hợp tín hiệu gốc và tín hiệu sinh để xây dựng bộ phát hiện bất thường (anomaly detection) trong dữ liệu y sinh.
* **Tối ưu hóa kiến trúc mô hình:** Tự động hóa quá trình chọn kiến trúc (AutoML) hoặc sử dụng các kỹ thuật như Neural Architecture Search để tìm ra thiết kế tốt nhất.

# PHỤ LỤC

**Mã nguồn**

[**https://github.com/phucnguyenhoank/giua\_ky\_final**](https://github.com/phucnguyenhoank/giua_ky_final)

**Tài liệu tham khảo**

IDMC PPG and Respiration Dataset – PhysioNet

Pytorch Documentation – <https://pytorch.org/docs/stable/index.html>