```
import os
import pandas as pd

# Đường dâ˜n đê´n folder chứa các file CSV
list_folder_malware = ['./CSVS/SMSmalware-CSVs/SMSmalware/Biige',
'./CSVS/SMSmalware-CSVs/SMSmalware/Fakeinst', './CSVS/SMSmalware-
CSVs/SMSmalware/FakeNotify',
                       './CSVS/SMSmalware-CSVs/SMSmalware/Mazarbot',
'./CSVS/SMSmalware-CSVs/SMSmalware/Jifake',
'./CSVS/SMSmalware-CSVs/SMSmalware/Nandrobox',
                       './CSVS/SMSmalware-CSVs/SMSmalware/Plankton',
'./CSVS/SMSmalware-CSVs/SMSmalware/Zsone']

list_folder_benign = ['./CSVS/Benign-CSVs/Benign/Benign2017']


# Tạo một DataFrame để chứa dữ liệu từ tâ´t cả các file CSV
list_df = []
all_data = pd.DataFrame()
#Đọc từng file CSV và nô´i dữ liệu vào DataFrame chung
i = 0
# Đọc benign
for folder in list_folder_benign:
     file_list =  os.listdir(folder)
     csv_files = [file for file in file_list if file.endswith('.csv')]
     for csv in csv_files:
            file_path = os.path.join(folder, csv)
            data = pd.read_csv(file_path, header=0)
            data.columns = data.columns.str.replace(' ', '')
            data = data.sort_values(by='Timestamp')
            data['Label'] = 1
            list_df.append(data)
            all_data = pd.concat([all_data, data], ignore_index=True)
            i = i + 1
# Đọc malware
for folder in list_folder_malware:
     file_list =  os.listdir(folder)
     csv_files = [file for file in file_list if file.endswith('.csv')]
     for csv in csv_files:
        file_path = os.path.join(folder, csv)
        data = pd.read_csv(file_path, header=0)
        data.columns = data.columns.str.replace(' ', '')
        data = data.sort_values(by='Timestamp')
        data['Label'] = 0
        list_df.append(data)
        all_data = pd.concat([all_data, data], ignore_index=True)
```

```
# Hiển thị DataFrame chứa dữ liệu từ tất cả các file CSV
print(all_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 591574 entries, 0 to 591573
Data columns (total 85 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   FlowID                    591570 non-null  object
 1   SourceIP                  591574 non-null  object
 2   SourcePort                591574 non-null  float64
 3   DestinationIP             591574 non-null  object
 4   DestinationPort           591574 non-null  float64
 5   Protocol                  591574 non-null  float64
 6   Timestamp                 591574 non-null  object
 7   FlowDuration              591574 non-null  float64
 8   TotalFwdPackets           591574 non-null  float64
 9   TotalBackwardPackets      591574 non-null  float64
 10  TotalLengthofFwdPackets   591574 non-null  float64
 11  TotalLengthofBwdPackets   591574 non-null  float64
 12  FwdPacketLengthMax        591574 non-null  float64
 13  FwdPacketLengthMin        591574 non-null  float64
 14  FwdPacketLengthMean       591574 non-null  float64
 15  FwdPacketLengthStd        591574 non-null  float64
 16  BwdPacketLengthMax        591574 non-null  float64
 17  BwdPacketLengthMin        591574 non-null  float64
 18  BwdPacketLengthMean       591574 non-null  float64
 19  BwdPacketLengthStd        591574 non-null  float64
 20  FlowBytes/s               591574 non-null  float64
 21  FlowPackets/s             591574 non-null  float64
 22  FlowIATMean               591573 non-null  float64
 23  FlowIATStd                591573 non-null  float64
 24  FlowIATMax                591573 non-null  float64
 25  FlowIATMin                591573 non-null  float64
 26  FwdIATTotal               591573 non-null  float64
 27  FwdIATMean                591573 non-null  float64
 28  FwdIATStd                 591573 non-null  float64
 29  FwdIATMax                 591573 non-null  float64
 30  FwdIATMin                 591573 non-null  float64
 31  BwdIATTotal               591573 non-null  float64
 32  BwdIATMean                591573 non-null  float64
 33  BwdIATStd                 591573 non-null  float64
 34  BwdIATMax                 591572 non-null  float64
 35  BwdIATMin                 591572 non-null  float64
 36  FwdPSHFlags               591572 non-null  float64
 37  BwdPSHFlags               591572 non-null  float64
 38  FwdURGFlags               591572 non-null  float64
 39  BwdURGFlags               591572 non-null  float64
 40  FwdHeaderLength           591572 non-null  float64
 41  BwdHeaderLength           591572 non-null  float64
 42  FwdPackets/s             591572 non-null  float64
 43  BwdPackets/s             591571 non-null  float64
 44  MinPacketLength          591571 non-null  float64
 45  MaxPacketLength          591571 non-null  float64
 46  PacketLengthMean         591571 non-null  float64
 47  PacketLengthStd          591571 non-null  object
 48  PacketLengthVariance     591570 non-null  float64
 49  FINFlagCount             591570 non-null  float64
 50  SYNFlagCount             591570 non-null  float64
 51  RSTFlagCount             591570 non-null  float64
 52  PSHFlagCount             591570 non-null  float64
 53  ACKFlagCount             591570 non-null  float64
 54  URGFlagCount             591570 non-null  float64
 55  CWEFlagCount             591570 non-null  object
 56  ECEFlagCount             591569 non-null  float64
 57  Down/UpRatio             591569 non-null  object
 58  AveragePacketSize        591568 non-null  float64
 59  AvgFwdSegmentSize        591568 non-null  float64
 60  AvgBwdSegmentSize        591568 non-null  float64
 61  FwdHeaderLength.1        591568 non-null  float64
 62  FwdAvgBytes/Bulk         591568 non-null  float64
 63  FwdAvgPackets/Bulk       591568 non-null  float64
 64  FwdAvgBulkRate           591568 non-null  float64
 65  BwdAvgBytes/Bulk         591568 non-null  float64
 66  BwdAvgPackets/Bulk       591568 non-null  float64
 67  BwdAvgBulkRate           591568 non-null  float64
 68  SubflowFwdPackets        591568 non-null  float64
 69  SubflowFwdBytes          591568 non-null  float64
 70  SubflowBwdPackets        591568 non-null  float64
 71  SubflowBwdBytes          591568 non-null  float64
 72  Init_Win_bytes_forward   591568 non-null  float64
 73  Init_Win_bytes_backward  591568 non-null  float64
 74  act_data_pkt_fwd         591568 non-null  float64
 75  min_seg_size_forward     591568 non-null  float64
 76  ActiveMean               591568 non-null  float64
 77  ActiveStd                591567 non-null  float64
 78  ActiveMax                591567 non-null  float64
 79  ActiveMin                591567 non-null  float64
 80  IdleMean                 591567 non-null  float64
 81  IdleStd                  591567 non-null  float64
 82  IdleMax                  591567 non-null  float64
 83  IdleMin                  591567 non-null  float64
 84  Label                    591574 non-null  int64
dtypes: float64(77), int64(1), object(7)
memory usage: 383.6+ MB
None

all_data['Label'].value_counts()
```

```
Label
1    410548
0    181026
Name: count, dtype: int64
```

```
import seaborn as sns
import matplotlib.pyplot as plt
columns_drop=['FlowID', 'SourceIP', 'DestinationIP', 'SourcePort',
'DestinationPort', 'Timestamp',
                               'PacketLengthStd', 'CWEFlagCount',
'Down/UpRatio','FwdAvgPackets/Bulk', 'FwdAvgBulkRate',
                               'BwdAvgBytes/Bulk',
'BwdAvgPackets/Bulk', 'BwdAvgBulkRate', 'FwdURGFlags', 'BwdURGFlags',
                               'RSTFlagCount', 'ECEFlagCount',
'BwdPSHFlags', 'FwdAvgBytes/Bulk']
# Tính ma trận tương quan
data = all_data.drop(columns=columns_drop)
corr_matrix = data.corr()
corr_matrix['Label'].sort_values(ascending=False)
```

```
Label                         1.000000
ACKFlagCount                  0.026583
URGFlagCount                  0.022652
Init_Win_bytes_backward       0.012253
SubflowFwdPackets             0.004621
                                ...
FlowIATMax                   -0.041337
FlowDuration                 -0.044689
FwdIATStd                    -0.047722
FwdIATTotal                  -0.054961
FwdIATMax                    -0.055962
Name: Label, Length: 65, dtype: float64
```

```
# def is_valid_format(timestamp, format='%d/%m/%Y %H:%M:%S'):
#     try:
#         pd.to_datetime(timestamp, format='%d/%m/%Y %H:%M:%S')
#         return True
#     except ValueError:
#         return False

# list_df_5_minutes = []
# #Tạo list graph
# for df in list_df:
#     i = 0
#     timeStamp = df['Timestamp'].iloc[i]
#     # Chuyển đổi timestamp thành đối tượng datetime và cộng 5
phút
```

```
#     while not is_valid_format(timeStamp, format='%d/%m/%Y %H:%M:%S')
:
#         i = i + 1
#         timeStamp = df['Timestamp'].iloc[i]

#     datetime = pd.to_datetime(timeStamp, format='%d/%m/%Y %H:%M:%S')
+ pd.Timedelta(minutes=5)
#     new_df = pd.DataFrame(columns=all_data.columns)
#     for row in df.itertuples(index=False):
#         try:
#             date_time_of_row = pd.to_datetime(row.Timestamp,
format='%d/%m/%Y %H:%M:%S')
#         except ValueError:
#             # Bỏ qua row nếu không thể chuyển đổi thành đối
tượng datetime
#             continue
#         if(date_time_of_row <= datetime) :
#             new_df.loc[len(new_df)] = list(row)
#         elif not new_df.empty:
#             list_df_5_minutes.append(new_df)
#             new_df = pd.DataFrame(columns=all_data.columns)
#             datetime = datetime + pd.Timedelta(minutes=5)

# len(list_df_5_minutes)

def convert_ip_label(df):
    le_columns = ['SourceIP', 'DestinationIP']
    ip_list = list(df['SourceIP'].unique()) +
list(df['DestinationIP'].unique())
    ip_set = list(set(ip_list))

    for column in le_columns:
        list_unique = list(df[column].unique())
        for val in list_unique:
            df.loc[df[column] == val, column] = ip_set.index(val)

    df['DestinationIP'] = df['DestinationIP'].astype(int)
    df['SourceIP'] = df['SourceIP'].astype(int)
    return df

import networkx as nx
import matplotlib.pyplot as plt
list_graph = []
list_label = []
for df in list_df:
    df = convert_ip_label(df)
    graph = nx.from_pandas_edgelist(df, 'SourceIP', 'DestinationIP',
                                     create_using=nx.MultiDiGraph(),
edge_attr=df.drop(columns=columns_drop).columns.values.tolist())
    list_graph.append(graph)
    count_value = len(df['Label'].unique())
    if count_value == 1:
        label = df['Label'].unique()[0]
    else:
        label = 0
    list_label.append(label)


import torch
from torch_geometric.data import Data, DataLoader
from torch_geometric.loader import  DataLoader
import numpy as np
from sklearn.model_selection import train_test_split
def graph_to_pyg_data(graph, label):
    # Khởi tạo ma trận B_in với tất cả các phần tử bằng 0
    num_nodes = graph.number_of_nodes()
    num_edges = graph.number_of_edges()
    B_in = torch.zeros((num_nodes, num_edges), dtype=torch.float32)
    B_out = torch.zeros((num_nodes, num_edges), dtype=torch.float32)
    for i, node in enumerate(graph.nodes):
        for j, edge in enumerate(graph.edges):
            if node == edge[1]:
                B_in[i, j] = 1
            if node == edge[0]:
                B_out[i, j] = 1
    Y = torch.tensor(label, dtype=torch.long)
    X = torch.tensor([list(graph.edges[edge].values()) for edge in
graph.edges], dtype=torch.float)
    return Data(B_in = B_in, B_out=B_out, X=X, Y=Y)

list_graph_labeled = list(zip(list_graph, list_label))
list_data = []
for graph, label in list_graph_labeled:
    list_data.append(graph_to_pyg_data(graph, label))

import torch
import torch.nn as nn
import torch.nn.functional as F

class GraphNeuralNetwork(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(GraphNeuralNetwork, self).__init__()
        self.fc0 = nn.Linear(input_dim, hidden_dim)
        self.fc1 = nn.Linear(2*hidden_dim, hidden_dim)
        self.fc2 = nn.Linear(3*hidden_dim, hidden_dim)
        self.fc3 = nn.Linear(hidden_dim, output_dim)
    def forward(self, B_in, B_out, X):
        # Tính E_0
        E_0 = F.relu(self.fc0(X))
        # Tính H0
        matrix1 = torch.matmul(B_in, E_0)
        matrix2 = torch.matmul(B_out, E_0)
        result_matrix_0 = torch.cat((matrix1, matrix2), dim=1)
        H_0 = F.relu(self.fc1(result_matrix_0))
        # Tính E1
        matrix3 = torch.matmul(B_in.t(), H_0)
        matrix4 = torch.matmul(B_out.t(), H_0)
        result_matrix_1 = torch.cat((matrix3, matrix4, E_0), dim=1)
        E_1 = F.relu(self.fc2(result_matrix_1))
        # Tính H1
        matrix5 = torch.matmul(B_in, E_1)
        matrix6 = torch.matmul(B_out, E_1)
        result_matrix_2 = torch.cat((matrix5, matrix6, H_0), dim=1)
        H_1 = F.relu(self.fc2(result_matrix_2))
        # Pooling mean cho H_1
        H_1_mean = torch.mean(H_1, dim=0, keepdim=True)
        # Softmax cho đầu ra của H_1_mean
        output = F.softmax(self.fc3(H_1_mean), dim=1)
        return output

# Chia dữ liệu thành tập huấn luyện và tập kiểm tra
train_data, test_data = train_test_split(list_data, test_size=0.2,
random_state=42)

# from torch.utils.data import Dataset, DataLoader
# class MyDataset(Dataset):
#     def __init__(self, list_data):
#         self.list_data = list_data

#     def __len__(self):
#         return len(self.list_data)

#     def __getitem__(self, idx):
#         if isinstance(idx, torch.Tensor):
#             idx = idx.tolist()
#         B_in = self.list_data[idx].B_in
#         B_out = self.list_data[idx].B_out
#         X = self.list_data[idx].X
#         Y = self.list_data[idx].Y

#         return [B_in, B_out, X, Y]

# def my_collate(batch):
#     inputs, labels = zip(*batch)
#     return inputs, labels

# trainset = MyDataset(train_data)
# testset = MyDataset(test_data)
# train_loader = DataLoader(trainset, batch_size=64, shuffle=True,
collate_fn=my_collate)
```

```
# test_loader = DataLoader(testset, batch_size=64, shuffle=False,
collate_fn=my_collate)

import torch
import torch.nn as nn
import torch.optim as optim
input_dim = 65 # Thay thế bằng kích thước của vectơ đặc trưng đỉnh
hidden_dim = 64
output_dim = 1
lr = 0.001
# Use gpu if available
device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")
model = GraphNeuralNetwork(input_dim, hidden_dim,
output_dim).to(device)
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=lr)

import torch
import torch.nn as nn
import torch.optim as optim
# Huấn luyện mô hình
def train(model, train_loader, criterion, optimizer):
    model.train()
    for data in train_loader:
        optimizer.zero_grad()
        output = model(data.B_in, data.B_out, data.X)
        if torch.isnan(output).any():
            continue
        loss = criterion(output.squeeze(), data.Y)
        loss.backward()
        optimizer.step()

# Đánh giá mô hình
def evaluate(model, loader, criterion):
    model.eval()
    total_loss = 0.0
    correct = 0
    total_samples = 0

    with torch.no_grad():
        for data in loader:
            output = model(data.B_in, data.B_out, data.X)
            if torch.isnan(output).any():
                continue
            loss = criterion(output, data.Y)
            total_loss += loss.item()
            _, predicted = torch.max(output, 1)
            correct += (predicted == data.Y).sum().item()
            total_samples += 1
    accuracy = correct / total_samples
    average_loss = total_loss / len(loader)

    return accuracy, average_loss

# Thiết lập các tham số
input_dim = 65 # Thay thế bằng kích thước của vectơ đặc trưng đỉnh
hidden_dim = 64
output_dim = 2
lr = 0.001
epochs = 50

model = GraphNeuralNetwork(input_dim, hidden_dim, output_dim)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=lr)

# Huấn luyện và kiểm thử mô hình
for epoch in range(epochs):
    train(model, train_data, criterion, optimizer)
    train_accuracy, train_loss = evaluate(model, train_data,
criterion)
    test_accuracy, test_loss = evaluate(model, test_data, criterion)
    print(f'Epoch {epoch + 1}/{epochs}, Test Loss: {test_loss:.4f},
Test Acc: {test_accuracy * 100:.2f}%')

# import torch
# import torch.nn as nn
# import torch.optim as optim
# # Huấn luyện mô hình
# def train(model, train_loader, criterion, optimizer, device):
#     model.train()
#     for data in train_loader:
#         data = data.to(device)
#         optimizer.zero_grad()
#         output = model(data)
#         loss = criterion(output, data.y)
#         loss.backward()
#         optimizer.step()

# # Đánh giá mô hình
# def evaluate(model, loader, criterion, device):
#     model.eval()
#     total_loss = 0.0
#     correct = 0
#     total_samples = 0

#     with torch.no_grad():
#         for data in loader:
#             data = data.to(device)
#             output = model(data)
#             loss = criterion(output, data.y)
#             total_loss += loss.item()
#             _, predicted = torch.max(output, 1)
#             correct += (predicted == data.y).sum().item()
#             total_samples += data.y.size(0)

#     accuracy = correct / total_samples
#     average_loss = total_loss / len(loader)

#     return accuracy, average_loss

# # Thiết lập các tham số
# input_dim = 75 # Thay thế bằng kích thước của vectơ đặc trưng
đỉnh
# hidden_dim = 64
# output_dim = 2
# lr = 0.001
# epochs = 100

# # Tạo mô hình và các thành phần khác
# device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
# model = GraphNeuralNetwork(input_dim, hidden_dim,
output_dim).to(device)
# criterion = nn.CrossEntropyLoss()
# optimizer = optim.Adam(model.parameters(), lr=lr)

# # Huấn luyện và kiểm thử mô hình
# for epoch in range(epochs):
#     train(model, train_loader, criterion, optimizer, device)
#     train_accuracy, train_loss = evaluate(model, train_loader,
criterion, device)
#     test_accuracy, test_loss = evaluate(model, test_loader,
criterion, device)
#     print(f'Epoch {epoch + 1}/{epochs}, Train Loss:
{train_loss:.4f}, Test Acc: {test_accuracy * 100:.2f}%')
```