```
import os
import torch
os.environ['TORCH'] = torch.__version__
from torch import Tensor
import torch.nn as nn
from torch.nn import Linear, MultiheadAttention, Dropout, Sigmoid, ReLU, Parameter, ModuleList, BCELoss
import torch.optim as optim
from torch.optim import Adam, AdamW
from torch.optim.lr_scheduler import ReduceLROnPlateau
from torch.utils.data import TensorDataset, DataLoader
import numpy as np
import pandas as pd
import copy
from copy import deepcopy
import sklearn.metrics
from sklearn.metrics import roc_curve, auc, roc_auc_score, precision_recall_curve, accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import random
import time

emb_dim = 1536
num_heads = 32
dro = 0.3
lin_dim = 2048
```

```
num_layers = 1
output_dim = 1


class Predictor(nn.Module):
    def __init__(self, emb_dim, num_heads, dro, lin_dim):
        super(Predictor, self).__init__()
        self.mha_attention = nn.MultiheadAttention(emb_dim, num_heads, dropout=dro)
        self.linear1 = nn.Linear(emb_dim, lin_dim)
        self.linear2 = nn.Linear(lin_dim, emb_dim)
        self.relu = nn.ReLU()
        self.dropout1 = nn.Dropout(dro)
        self.dropout2 = nn.Dropout(dro)
        self.dropout3 = nn.Dropout(dro)
        self.layer_norm1 = nn.LayerNorm(emb_dim)
        self.layer_norm2 = nn.LayerNorm(emb_dim)


    def forward(self, x):
        mha_output, _ = self.mha_attention(x, x, x)
        x = self.layer_norm1(x + self.dropout1(mha_output))
        lin_output = self.linear1(x)
        lin_output = self.relu(lin_output)
        lin_output = self.dropout2(lin_output)
        lin_output = self.linear2(lin_output)
        x = self.layer_norm2(x + self.dropout3(lin_output))
        return x
```

```
class PredictorModel(nn.Module):
    def __init__(self, emb_dim, num_heads, dro, lin_dim, num_layers, output_dim):
        super(PredictorModel, self).__init__()
        self.positional_encoding = nn.Parameter(torch.zeros(1, 4, emb_dim))
        self.predictors = nn.ModuleList([
            Predictor(emb_dim, num_heads, dro, lin_dim) for _ in range(num_layers)
        ])
        self.dropout1a = nn.Dropout(dro)
        self.linear1a = nn.Linear(emb_dim, output_dim)
        self.sigmoid = nn.Sigmoid()

    def forward(self, A, B, C, D):
        x = torch.cat((A, B, C, D), dim=1)
        x = x.view(x.size(0), 4, -1)
        x = x + self.positional_encoding[:, :x.size(1), :]
        x = x.permute(1, 0, 2)
        x = self.dropout1a(x)
        for predictor in self.predictors:
            x = predictor(x)
        x = x.permute(1, 0, 2)
        x = self.linear1a(x[:, 3, :])
        x = self.sigmoid(x)
        return x
```