ДОДАТОК А

Вихідний код квантового трансформера

```
    !pip install pennylane torch numpy scikit-learn
    !pip install pennylane-lightning[gpu]

    import torch
    import torch.nn as nn
    import torch.optim as optim
    from transformers import AutoTokenizer, AutoModel
    import pennylane as qml
    from sklearn.datasets import load_files
    from sklearn.model_selection import
train_test_split
    from sklearn.metrics import accuracy_score,
precision_recall_fscore_support
    from torch.utils.data import DataLoader,
TensorDataset
    import numpy as np
    import urllib.request
    import tarfile
    import os
    import random

    os.environ["TOKENIZERS_PARALLELISM"] = "false"

    # Configurations
    n_qubits = 8
    q_depth = 3
    batch_size = 16
    num_epochs = 5
    learning_rate = 5e-5
    num_repeats = 5  # Число повторів

    device = torch.device('cuda' if
torch.cuda.is_available() else 'cpu')

    # === Quantum circuit & Quantum Layer ===
    qml_dev = qml.device("lightning.gpu",
wires=n_qubits)

    @qml.qnode(qml_dev, interface='torch',
diff_method="adjoint")
    def quantum_circuit(inputs, weights):
```

```
        qml.AngleEmbedding(inputs,
wires=range(n_qubits), rotation='Y')
        qml.StronglyEntanglingLayers(weights,
wires=range(n_qubits))
        return  [qml.expval(qml.PauliZ(i))  for  i  in
range(n_qubits)]

    class QuantumLayer(nn.Module):
        def __init__(self):
            super().__init__()
            weight_shapes   =   {"weights":   (q_depth,
n_qubits, 3)}
            self.qlayer                                =
qml.qnn.TorchLayer(quantum_circuit, weight_shapes)
            self.norm = nn.BatchNorm1d(n_qubits)
        def forward(self, x):
            x = nn.functional.normalize(x)
            x = self.qlayer(x)
            return self.norm(x)

    class QuantumBERT(nn.Module):
        def    __init__(self,    bert_model="bert-base-
uncased"):
            super().__init__()
            self.bert                                  =
AutoModel.from_pretrained(bert_model)
            self.dropout = nn.Dropout(0.3)
            self.quantum_layer = QuantumLayer()
            self.fc = nn.Linear(n_qubits, 2)
        def forward(self, input_ids, attention_mask):
            bert_output     =     self.bert(input_ids,
attention_mask=attention_mask).pooler_output
            bert_output  =  self.dropout(bert_output[:,
:n_qubits])
            quantum_output                             =
self.quantum_layer(bert_output)
            return self.fc(quantum_output)

    # --- Prepare Data (load only once!) ---
    tokenizer  =  AutoTokenizer.from_pretrained("bert-
base-uncased")
    url                                                =
'https://ai.stanford.edu/~amaas/data/sentiment/aclImd
b_v1.tar.gz'
    filename = 'aclImdb_v1.tar.gz'
```

```
    if not os.path.exists('aclImdb'):
        urllib.request.urlretrieve(url, filename)
        with tarfile.open(filename, 'r:gz') as tar:
            tar.extractall()
    train_data = load_files('aclImdb/train',
categories=['pos', 'neg'], encoding='utf-8')

    # --- Метрики ---
    accuracies, precisions, recalls, f1s = [], [], [],
[]
    train_losses_all, val_losses_all = [], []

    def print_mean_std(name, arr):
        print(f"{name}: mean = {np.mean(arr):.4f}, std
= {np.std(arr):.4f}")

    for repeat in range(num_repeats):
        print(f"\n=== Repeat {repeat +
1}/{num_repeats} ===")
        # --- Случайная выборка 2000 ---
        indices =
np.random.choice(len(train_data.data), 2000,
replace=False)
        texts = [train_data.data[i] for i in indices]
        labels = [train_data.target[i] for i in
indices]
        train_texts, val_texts, train_labels,
val_labels = train_test_split(
            texts, labels, test_size=0.2,
random_state=random.randint(0, 10000)
        )

        # Токенізація
        train_encodings = tokenizer(train_texts,
truncation=True, padding=True, max_length=128,
return_tensors='pt')
        val_encodings = tokenizer(val_texts,
truncation=True, padding=True, max_length=128,
return_tensors='pt')
        train_labels = torch.tensor(train_labels)
        val_labels = torch.tensor(val_labels)

        train_dataset =
TensorDataset(train_encodings['input_ids'],
train_encodings['attention_mask'], train_labels)
```

```
        val_dataset                                    =
TensorDataset(val_encodings['input_ids'],
val_encodings['attention_mask'], val_labels)
        train_loader    =    DataLoader(train_dataset,
batch_size=batch_size,  shuffle=True,  num_workers=2,
pin_memory=True)
        val_loader      =      DataLoader(val_dataset,
batch_size=batch_size,  shuffle=False,  num_workers=2,
pin_memory=True)

        # Model, Optimizer, Loss
        model = QuantumBERT().to(device)
        optimizer  =  optim.AdamW(model.parameters(),
lr=learning_rate)
        criterion = nn.CrossEntropyLoss()

        # --- Train ---
        model.train()
        for epoch in range(num_epochs):
            total_loss = 0
            for  input_ids,  attention_mask,  labels  in
train_loader:
                input_ids,  attention_mask,  labels  =
input_ids.to(device),       attention_mask.to(device),
labels.to(device)
                optimizer.zero_grad()
                outputs       =       model(input_ids,
attention_mask)
                loss = criterion(outputs, labels)
                loss.backward()
                optimizer.step()
                total_loss += loss.item()
            avg_loss = total_loss / len(train_loader)
            print(f"Epoch  {epoch  +  1}/{num_epochs},
Train Loss: {avg_loss:.4f}")
        train_losses_all.append(avg_loss)

        # --- Evaluation ---
        model.eval()
        preds, truths = [], []
        val_losses = []
        with torch.no_grad():
            for  input_ids,  attention_mask,  labels  in
val_loader:
```

```
                input_ids, attention_mask, labels =
input_ids.to(device),
                attention_mask.to(device),
labels.to(device)
                outputs      =      model(input_ids,
attention_mask)
                loss = criterion(outputs, labels)
                val_losses.append(loss.item())

preds.extend(outputs.argmax(dim=1).cpu().numpy())
                truths.extend(labels.cpu().numpy())

        val_loss_mean = np.mean(val_losses)
        val_losses_all.append(val_loss_mean)

        accuracy = accuracy_score(truths, preds)
        precision,      recall,      f1,      _      =
precision_recall_fscore_support(truths,        preds,
average='binary')
        accuracies.append(accuracy)
        precisions.append(precision)
        recalls.append(recall)
        f1s.append(f1)

        print(f'Val     Loss:     {val_loss_mean:.4f},
Accuracy: {accuracy:.4f}, Precision: {precision:.4f},
Recall: {recall:.4f}, F1-score: {f1:.4f}')

    # --- Виводимо кінцкві метрики ---
    print("\n==== Statistics over runs ====")
    print_mean_std('Train Loss', train_losses_all)
    print_mean_std('Val Loss', val_losses_all)
    print_mean_std('Accuracy', accuracies)
    print_mean_std('Precision', precisions)
    print_mean_std('Recall', recalls)
    print_mean_std('F1-score', f1s)
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	59 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Попович К.С.			Покращена архітектура трансформера за рахунок квантових обчислень	Шифр групи	Код напр./спец.
Перевірив		Шевченко О.Ю.				СШІм-23-2	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Золотухін О.В.					