ДОДАТОК А

Вихідний код програмної реалізації

model.py:

```
import torch.nn as nn

class Classifier(nn.Module):
    def __init__(self, vocab_size, emb_dim=128):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, emb_dim,
padding_idx=0)
        self.classifier = nn.Sequential(
            nn.Linear(emb_dim, 64),
            nn.ReLU(),
            nn.Linear(64, 2)
        )

    def forward(self, input_ids):
        embedded = self.embedding(input_ids)
        pooled = embedded.mean(dim=1)
        return self.classifier(pooled)
```

prepare_text.py:

```
from datasets import load_dataset

dataset =
load_dataset("imdb")["train"].shuffle(seed=42).select(range(25
000))

with open("train_text.txt", "w", encoding="utf-8") as f:
    for example in dataset:
```

```
        f.write(example["text"].replace("<br /><br />", "
").replace("\n", " ").replace("  ", " ") + "\n")
```

train_tokenizers.py:

```
from tokenizers import Tokenizer
from tokenizers.normalizers import NFD, Lowercase,
StripAccents, Sequence
from tokenizers.pre_tokenizers import Whitespace
from tokenizers.trainers import BpeTrainer, WordPieceTrainer,
UnigramTrainer
from tokenizers.models import BPE, WordPiece, Unigram

def train_tokenizer(model_type, files, vocab_size=8000):
    special_tokens = ["[PAD]", "[UNK]", "[CLS]", "[SEP]",
"[MASK]"]

    if model_type == "bpe":
        model = BPE(unk_token="[UNK]")
        tokenizer = Tokenizer(model)
        trainer = BpeTrainer(vocab_size=vocab_size,
special_tokens=special_tokens)

    elif model_type == "wordpiece":
        model = WordPiece(unk_token="[UNK]")
        tokenizer = Tokenizer(model)
        trainer = WordPieceTrainer(vocab_size=vocab_size,
special_tokens=special_tokens)

    elif model_type == "unigram":
        model = Unigram()
        tokenizer = Tokenizer(model)
        trainer = UnigramTrainer(vocab_size=vocab_size,
special_tokens=special_tokens, unk_token="[UNK]")
```

```
    else:
        raise ValueError("Unsupported model type")

    tokenizer.normalizer = Sequence([NFD(), Lowercase(),
StripAccents()])
    tokenizer.pre_tokenizer = Whitespace()

    tokenizer.train(files, trainer)
    tokenizer.save(f"{model_type}_tokenizer.json")

if __name__ == "__main__":
    for model in ["bpe", "wordpiece", "unigram"]:
        train_tokenizer(model, ["train_text.txt"])
```

train.py:

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from transformers import PreTrainedTokenizerFast
from datasets import load_dataset
from model import Classifier
from tqdm import tqdm

def tokenize_dataset(dataset, tokenizer, max_length=128):
    def tokenize(example):
        return tokenizer(example['text'],
padding='max_length', truncation=True, max_length=max_length)

    return dataset.map(tokenize, batched=True)

def collate_fn(batch):
    input_ids = torch.tensor([item['input_ids'] for item in
batch])
    labels = torch.tensor([item['label'] for item in batch])
```

```
    return input_ids, labels

def train_model(model, dataloader, optimizer, criterion,
device):
    model.train()
    total_loss = 0
    for input_ids, labels in tqdm(dataloader):
        input_ids, labels = input_ids.to(device),
labels.to(device)
        optimizer.zero_grad()
        outputs = model(input_ids)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    return total_loss / len(dataloader)

def evaluate_model(model, dataloader, device):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for input_ids, labels in dataloader:
            input_ids, labels = input_ids.to(device),
labels.to(device)
            outputs = model(input_ids)
            predictions = torch.argmax(outputs, dim=1)
            correct += (predictions == labels).sum().item()
            total += labels.size(0)
    return correct / total

def main(model_type):
    tokenizer =
PreTrainedTokenizerFast(tokenizer_file=f"{model_type}_tokenize
r.json",
        unk_token="[UNK]", pad_token="[PAD]",
```

```
        cls_token="[CLS]", sep_token="[SEP]")

    raw_dataset = load_dataset("imdb")
    small_train =
raw_dataset["train"].shuffle(seed=42).select(range(2000))
    small_test =
raw_dataset["test"].shuffle(seed=42).select(range(500))

    tokenized_train = tokenize_dataset(small_train, tokenizer)
    tokenized_test = tokenize_dataset(small_test, tokenizer)

    train_loader = DataLoader(tokenized_train, batch_size=32,
shuffle=True, collate_fn=collate_fn)
    test_loader = DataLoader(tokenized_test, batch_size=32,
shuffle=False, collate_fn=collate_fn)

    device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
    vocab_size = tokenizer.vocab_size
    model = Classifier(vocab_size).to(device)

    optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
    criterion = nn.CrossEntropyLoss()

    print(f"Training model with {model_type} tokenizer...")
    for epoch in range(15):
        loss = train_model(model, train_loader, optimizer,
criterion, device)
        acc = evaluate_model(model, test_loader, device)
        print(f"Epoch {epoch+1}: Loss = {loss:.4f}, Accuracy =
{acc*100:.2f}%")

if __name__ == '__main__':
    for model_type in ["bpe", "wordpiece", "unigram"]:
        main(model_type)
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	73 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата	
Розробив		Майданович О.В.			Дослідження методів токенізації в задачах продуктивності NLP-моделей
Перевірив		Кудрявцева М.С.			
Н.контр.		Малєєва І.А.			
Затв.		Золотухін О.В.			

Шифр групи	Код напр./спец.
ІТШІ-21-2	122
ХНУРЕ кафедра ШІ	