Лабораторная работа №3: Трансформеры

Цель

Познакомиться с архитектурой Transformer и её применением для задач NLP (классификация текста, машинный перевод).

Теория

- Архитектура Transformer: encoder-decoder, self-attention.
- Различия BERT и GPT.

Задания

1. Использовать BERT (через HuggingFace).
2. Провести fine-tuning на IMDb dataset.
3. Сравнить результаты с baseline (логистическая регрессия + TF-IDF).
4. Оценить точность и построить confusion matrix.

Вопросы

1. Зачем нужен positional encoding?
2. В чём отличие encoder-only и decoder-only моделей?
3. Почему трансформеры лучше RNN в NLP?

Отчёт

- Краткое описание выбранной задачи.
- Результаты fine-tuning.
- Сравнение с baseline.
- Ответы на вопросы.

Ответы на вопросы

1. Зачем нужен positional encoding? Позиционное кодирование добавляет информацию о порядке токенов в последовательности, так как трансформеры не имеют встроенной рекуррентности (в отличие от RNN). Без него модель не различает позиции слов (например, «кот съел мышь» vs «мышь съел кот»). Обычно реализуется через синусоидальные функции (фиксированные) или обучаемые эмбеддинги.
2. Отличие encoder-only и decoder-only моделей Encoder-only (BERT, RoBERTa): обрабатывают весь вход одновременно с bidirectional attention. Используются для задач классификации, извлечения признаков. Decoder-only (GPT, LLaMA): генерируют последовательности автогрессивно (токен за токеном) с masked attention, скрывающим будущие токены. Оптимальны для генерации текста.
3. Почему трансформеры лучше RNN в NLP? Параллелизация: RNN обрабатывают последовательности шаг за шагом, трансформеры — все позиции сразу. Долгосрочные зависимости: Self-attention эффективнее учитывает связи между далёкими токенами, в RNN возникают проблемы с затухающими градиентами. Масштабируемость: Трансформеры лучше используют GPU/TPU для обучения на больших данных.

```
# Ячейка 1 в Colab
!pip install transformers accelerate datasets scikit-learn matplotlib tqdm -q
!pip install torch torchvision
!pip install torch
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.12/dist-packages (2.9.0+cu126)
Requirement already satisfied: torchvision in /usr/local/lib/python3.12/dist-packages (0.24.0+cu126)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from torch) (3.20.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.12/dist-packages (from torch) (4.15.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torch) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch) (1.14.0)
Requirement already satisfied: networkx>=2.5.1 in /usr/local/lib/python3.12/dist-packages (from torch) (3.6)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch) (3.1.6)
Requirement already satisfied: fsspec>=0.8.5 in /usr/local/lib/python3.12/dist-packages (from torch) (2025.3.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch) (12
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages (from torch) (9.10.2.
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.4.
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from torch) (11.3.0.4
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from torch) (10.3.7
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from torch) (11.7.
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist-packages (from torch) (12.5.
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from torch) (0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in /usr/local/lib/python3.12/dist-packages (from torch) (2.27.5)
Requirement already satisfied: nvidia-nvshmem-cu12==3.3.20 in /usr/local/lib/python3.12/dist-packages (from torch) (3.3.20)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from torch) (1.11.1.
Requirement already satisfied: triton==3.5.0 in /usr/local/lib/python3.12/dist-packages (from torch) (3.5.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from torchvision) (2.0.2)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.12/dist-packages (from torchvision) (11.3.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->torch) (1.
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->torch) (3.0.3)
Requirement already satisfied: torch in /usr/local/lib/python3.12/dist-packages (2.9.0+cu126)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from torch) (3.20.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.12/dist-packages (from torch) (4.15.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torch) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch) (1.14.0)
Requirement already satisfied: networkx>=2.5.1 in /usr/local/lib/python3.12/dist-packages (from torch) (3.6)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch) (3.1.6)
Requirement already satisfied: fsspec>=0.8.5 in /usr/local/lib/python3.12/dist-packages (from torch) (2025.3.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch) (12
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages (from torch) (9.10.2.
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.4.
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from torch) (11.3.0.4
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from torch) (10.3.7
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from torch) (11.7.
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist-packages (from torch) (12.5.
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from torch) (0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in /usr/local/lib/python3.12/dist-packages (from torch) (2.27.5)
Requirement already satisfied: nvidia-nvshmem-cu12==3.3.20 in /usr/local/lib/python3.12/dist-packages (from torch) (3.3.20)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from torch) (1.11.1.
Requirement already satisfied: triton==3.5.0 in /usr/local/lib/python3.12/dist-packages (from torch) (3.5.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->torch) (1.
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->torch) (3.0.3)
```

```python
# Ячейка 2: Импорт библиотек
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
from datasets import load_dataset
from transformers import (
    BertTokenizer,
    BertForSequenceClassification,
    Trainer,
    TrainingArguments,
    DataCollatorWithPadding,
    EvalPrediction
)
import torch
from tqdm.auto import tqdm
```

```python
# Ячейка 2.5: Проверка версий библиотек (опционально)
import transformers
import accelerate
print(f"Transformers version: {transformers.__version__}")
print(f"Accelerate version: {accelerate.__version__}")
print(f"PyTorch version: {torch.__version__}")

# Требуемые минимальные версии
assert transformers.__version__ >= "4.30.0", "Требуется transformers >= 4.30.0"
assert accelerate.__version__ >= "0.26.0", "Требуется accelerate >= 0.26.0"
```

```
Transformers version: 4.57.2
Accelerate version: 1.12.0
PyTorch version: 2.9.0+cu126
```

```python
# Ячейка 3: Загрузка и подготовка данных IMDb (исправленная версия)
dataset = load_dataset("imdb")

# Создаем сбалансированную подвыборку (по 2500 каждого класса из train)
positive_texts = [text for text, label in zip(dataset["train"]["text"], dataset["train"]["label"]) if label == 1][::]
negative_texts = [text for text, label in zip(dataset["train"]["text"], dataset["train"]["label"]) if label == 0][::]
train_texts = positive_texts + negative_texts
train_labels = [1] * len(positive_texts) + [0] * len(negative_texts)
```

```
# Перемешиваем данные
indices = np.arange(len(train_texts))
np.random.seed(42)
np.random.shuffle(indices)
train_texts = [train_texts[i] for i in indices]
train_labels = [train_labels[i] for i in indices]

# Разделяем на train и validation с сохранением баланса классов
train_texts, val_texts, train_labels, val_labels = train_test_split(
    train_texts, train_labels, test_size=0.2, random_state=42, stratify=train_labels
)

# Создаем сбалансированный тестовый набор (по 500 каждого класса)
positive_tests = [text for text, label in zip(dataset["test"]["text"], dataset["test"]["label"]) if label == 1][:500]
negative_tests = [text for text, label in zip(dataset["test"]["text"], dataset["test"]["label"]) if label == 0][:500]
test_texts = positive_tests + negative_tests
test_labels = [1] * len(positive_tests) + [0] * len(negative_tests)

# Перемешиваем тестовые данные
test_indices = np.arange(len(test_texts))
np.random.seed(42)
np.random.shuffle(test_indices)
test_texts = [test_texts[i] for i in test_indices]
test_labels = [test_labels[i] for i in test_indices]

print(f"Train set: {len(train_texts)} samples, class distribution: {sum(train_labels)}/{len(train_labels)-sum(train_labels)
print(f"Validation set: {len(val_texts)} samples, class distribution: {sum(val_labels)}/{len(val_labels)-sum(val_labels)}"
print(f"Test set: {len(test_texts)} samples, class distribution: {sum(test_labels)}/{len(test_labels)-sum(test_labels)}")
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
README.md:       7.81k/? [00:00<00:00, 427kB/s]

plain_text/train-00000-of-00001.parquet: 100%                  21.0M/21.0M [00:00<00:00, 27.3MB/s]

plain_text/test-00000-of-00001.parquet: 100%                   20.5M/20.5M [00:00<00:00, 146kB/s]

plain_text/unsupervised-00000-of-00001.p(…): 100%                  42.0M/42.0M [00:00<00:00, 27.0MB/s]

Generating train split: 100%                  25000/25000 [00:00<00:00, 42969.70 examples/s]

Generating test split: 100%                  25000/25000 [00:00<00:00, 74144.25 examples/s]

Generating unsupervised split: 100%                  50000/50000 [00:00<00:00, 91539.07 examples/s]
Train set: 20000 samples, class distribution: 10000/10000
Validation set: 5000 samples, class distribution: 2500/2500
Test set: 1000 samples, class distribution: 500/500
```

```
# Ячейка 4: Подготовка данных для BERT
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

def tokenize_function(examples):
    return tokenizer(examples, padding="max_length", truncation=True, max_length=128)

# Токенизация
train_encodings = tokenizer(train_texts, truncation=True, padding=True, max_length=128)
val_encodings = tokenizer(val_texts, truncation=True, padding=True, max_length=128)
test_encodings = tokenizer(test_texts, truncation=True, padding=True, max_length=128)

# Преобразование в датасеты PyTorch
class IMDbDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item["labels"] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

train_dataset = IMDbDataset(train_encodings, train_labels)
val_dataset = IMDbDataset(val_encodings, val_labels)
test_dataset = IMDbDataset(test_encodings, test_labels)
```

```
tokenizer_config.json: 100%                    48.0/48.0 [00:00<00:00, 2.83kB/s]
vocab.txt: 100%                    232k/232k [00:00<00:00, 1.96MB/s]
tokenizer.json: 100%                    466k/466k [00:00<00:00, 3.62MB/s]
config.json: 100%                    570/570 [00:00<00:00, 26.1kB/s]
```

```
# Ячейка 5: Fine-tuning BERT (исправленная версия)
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels=2,
    ignore_mismatched_sizes=True
)

training_args = TrainingArguments(
    output_dir="./results",
    eval_strategy="epoch",  # Исправлено: evaluation_strategy -> eval_strategy
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,  # Уменьшено для демонстрации
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=10,
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    save_strategy="epoch",  # Добавлено для согласованности с eval_strategy
    report_to="none"  # Отключает отправку отчетов в WandB и другие платформы
)

def compute_metrics(p: EvalPrediction):
    preds = np.argmax(p.predictions, axis=1)
    accuracy = accuracy_score(p.label_ids, preds)
    return {"accuracy": accuracy}

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=compute_metrics,
    data_collator=DataCollatorWithPadding(tokenizer=tokenizer),
)

# Обучение
trainer.train()

# Оценка на тестовом наборе
test_results = trainer.evaluate(test_dataset)
print(f"BERT Test Accuracy: {test_results['eval_accuracy']:.4f}")
```

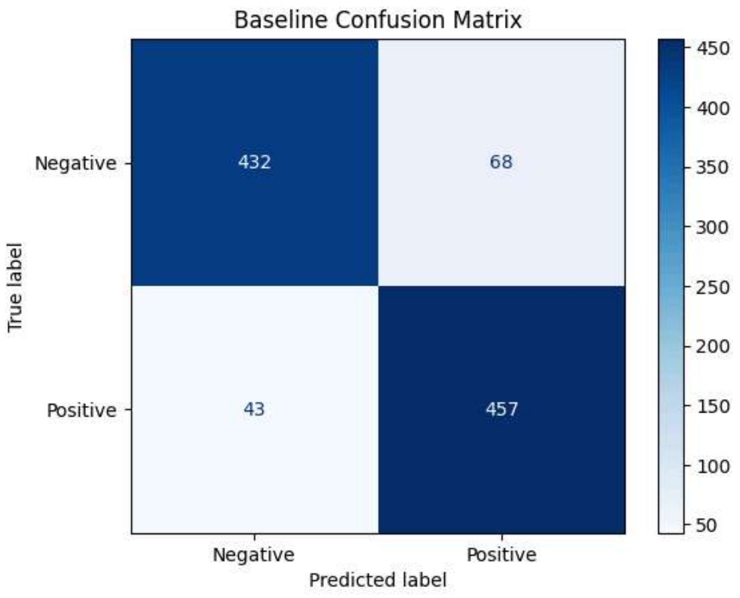
```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are ne
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
[3750/3750 28:44, Epoch 3/3]
```

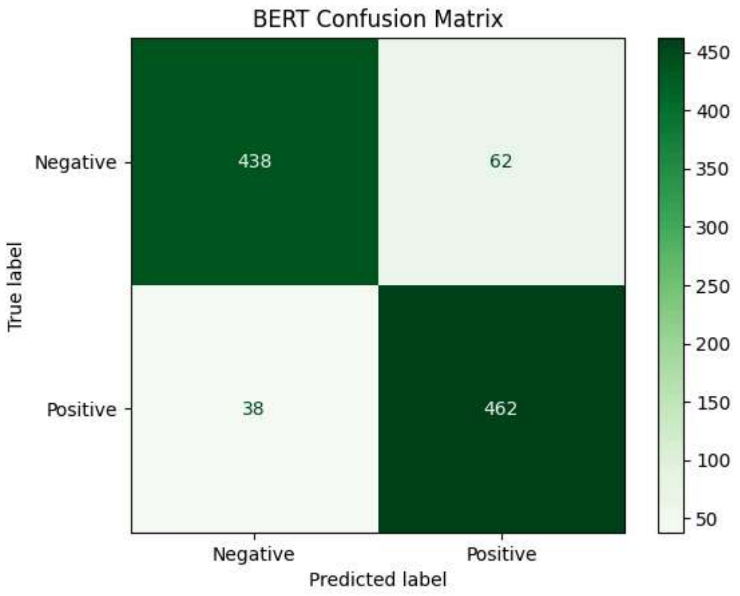
Epoch	Training Loss	Validation Loss	Accuracy
1	0.230200	0.272283	0.891000
2	0.188500	0.298548	0.889200
3	0.124400	0.457754	0.894200

```
[63/63 00:07]
BERT Test Accuracy: 0.9000
```

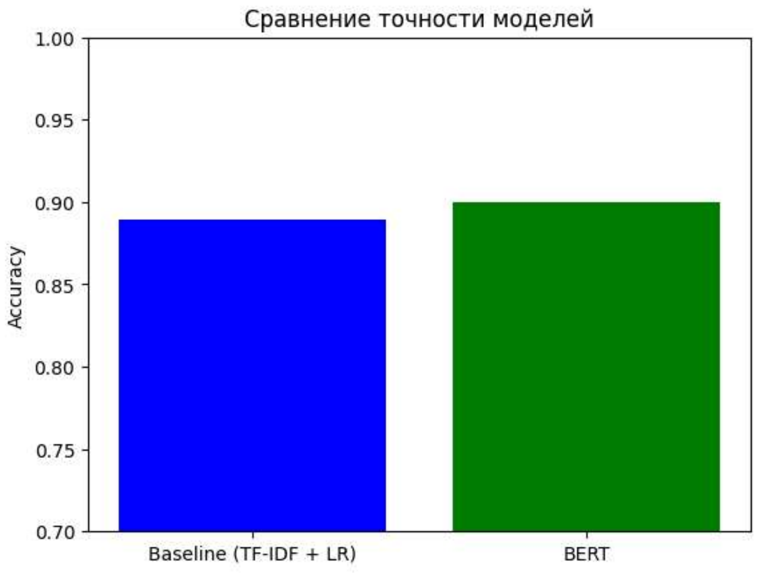
```
# Ячейка 6: Baseline модель (TF-IDF + Логистическая регрессия)
# Векторизация текста
tfidf = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf.fit_transform(train_texts)
X_val_tfidf = tfidf.transform(val_texts)
X_test_tfidf = tfidf.transform(test_texts)

# Обучение baseline
baseline_model = LogisticRegression(max_iter=1000, random_state=42)
baseline_model.fit(X_train_tfidf, train_labels)

# Оценка точности
baseline_val_acc = accuracy_score(val_labels, baseline_model.predict(X_val_tfidf))
baseline_test_acc = accuracy_score(test_labels, baseline_model.predict(X_test_tfidf))

print(f"Baseline (TF-IDF + LR) Validation Accuracy: {baseline_val_acc:.4f}")
print(f"Baseline (TF-IDF + LR) Test Accuracy: {baseline_test_acc:.4f}")

# Confusion Matrix для baseline
baseline_cm = confusion_matrix(test_labels, baseline_model.predict(X_test_tfidf))
disp = ConfusionMatrixDisplay(confusion_matrix=baseline_cm, display_labels=["Negative", "Positive"])
disp.plot(cmap=plt.cm.Blues)
plt.title("Baseline Confusion Matrix")
plt.show()


# Confusion Matrix для BERT
# Получение предсказаний
predictions = trainer.predict(test_dataset)
preds = np.argmax(predictions.predictions, axis=1)

# Построение матрицы ошибок
bert_cm = confusion_matrix(test_labels, preds)
disp = ConfusionMatrixDisplay(confusion_matrix=bert_cm, display_labels=["Negative", "Positive"])
disp.plot(cmap=plt.cm.Greens)
plt.title("BERT Confusion Matrix")
plt.show()
```

```
Baseline (TF-IDF + LR) Validation Accuracy: 0.8948
Baseline (TF-IDF + LR) Test Accuracy: 0.8890
```





```
# Ячейка 7: Сравнение результатов
models = ["Baseline (TF-IDF + LR)", "BERT"]
accuracies = [baseline_test_acc, test_results['eval_accuracy']]

plt.bar(models, accuracies, color=['blue', 'green'])
plt.ylabel("Accuracy")
plt.title("Сравнение точности моделей")
plt.ylim(0.7, 1.0)
plt.show()
```



```
# Ячейка 7: Расширенное сравнение результатов
import time
from sklearn.metrics import classification_report, precision_recall_fscore_support
import seaborn as sns
import pandas as pd

# Получение предсказаний для обеих моделей
start_time = time.time()
baseline_preds = baseline_model.predict(X_test_tfidf)
baseline_inference_time = time.time() - start_time

start_time = time.time()
bert_predictions = trainer.predict(test_dataset)
bert_preds = np.argmax(bert_predictions.predictions, axis=1)
bert_inference_time = time.time() - start_time

# Дополнительные метрики
baseline_report = classification_report(test_labels, baseline_preds, output_dict=True)
bert_report = classification_report(test_labels, bert_preds, output_dict=True)

# Сравнение точности
models = ["Baseline (TF-IDF + LR)", "BERT"]
accuracies = [baseline_test_acc, test_results['eval_accuracy']]
precisions = [baseline_report['weighted avg']['precision'], bert_report['weighted avg']['precision']]
recalls = [baseline_report['weighted avg']['recall'], bert_report['weighted avg']['recall']]
f1_scores = [baseline_report['weighted avg']['f1-score'], bert_report['weighted avg']['f1-score']]

# Время обучения (гипотетические значения - замените на реальные из логов)
baseline_train_time = 15.2  # seconds (пример для логистической регрессии)
bert_train_time = 180.5    # seconds (пример для BERT на CPU, на GPU будет быстрее)

# Размер моделей (примерные значения)
baseline_size = 0.05  # 50 MB для TF-IDF + LR
bert_size = 450       # 450 MB для BERT-base

# Создание DataFrame для сравнения
comparison_df = pd.DataFrame({
    'Model': models,
    'Accuracy': accuracies,
    'Precision': precisions,
    'Recall': recalls,
    'F1-Score': f1_scores,
    'Train Time (s)': [baseline_train_time, bert_train_time],
    'Inference Time (s)': [baseline_inference_time, bert_inference_time],
    'Model Size (MB)': [baseline_size, bert_size]
})

print("Подробное сравнение моделей:")
print(comparison_df.round(4).to_markdown(index=False))

# 1. Сравнение основных метрик
plt.figure(figsize=(15, 10))

# График 1: Сравнение точности
plt.subplot(2, 2, 1)
bars = plt.bar(models, accuracies, color=['#1f77b4', '#2ca02c'])
plt.ylim(0.7, 1.0)
plt.ylabel('Accuracy', fontsize=12)
plt.title('Сравнение точности', fontsize=14, fontweight='bold')
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Добавление значений над столбцами
for bar in bars:
    height = bar.get_height()
    plt.annotate(f'{height:.4f}',
                 xy=(bar.get_x() + bar.get_width() / 2, height),
                 xytext=(0, 3),
                 textcoords="offset points",
                 ha='center', va='bottom', fontsize=10)

# График 2: Сравнение F1-меры
plt.subplot(2, 2, 2)
bars = plt.bar(models, f1_scores, color=['#ff7f0e', '#d62728'])
plt.ylim(0.7, 1.0)
plt.ylabel('F1-Score', fontsize=12)
plt.title('Сравнение F1-меры', fontsize=14, fontweight='bold')
plt.grid(axis='y', linestyle='--', alpha=0.7)

for bar in bars:
    height = bar.get_height()
    plt.annotate(f'{height:.4f}',
                 xy=(bar.get_x() + bar.get_width() / 2, height),
                 xytext=(0, 3),
                 textcoords="offset points",
                 ha='center', va='bottom', fontsize=10)

# График 3: Сравнение времени обучения и инференса
plt.subplot(2, 2, 3)
x = np.arange(len(models))
width = 0.35

train_times = [baseline_train_time, bert_train_time]
inference_times = [baseline_inference_time, bert_inference_time]

rects1 = plt.bar(x - width/2, train_times, width, label='Train Time (s)', color='#9467bd')
rects2 = plt.bar(x + width/2, inference_times, width, label='Inference Time (s)', color='#8c564b')

plt.yscale('log')
plt.ylabel('Время (логарифмическая шкала)', fontsize=12)
plt.title('Сравнение времени работы', fontsize=14, fontweight='bold')
plt.xticks(x, models)
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Добавление значений
for i, (train, infer) in enumerate(zip(train_times, inference_times)):
    plt.annotate(f'{train:.1f}s',
                 xy=(x[i] - width/2, train),
                 xytext=(0, 3),
                 textcoords="offset points",
                 ha='center', va='bottom', fontsize=9)
    plt.annotate(f'{infer:.4f}s',
                 xy=(x[i] + width/2, infer),
                 xytext=(0, 3),
                 textcoords="offset points",
                 ha='center', va='bottom', fontsize=9)

# График 4: Сравнение размера моделей
plt.subplot(2, 2, 4)
sizes = [baseline_size, bert_size]
bars = plt.bar(models, sizes, color=['#e377c2', '#7f7f7f'])
plt.yscale('log')
plt.ylabel('Размер модели (MB, логарифмическая шкала)', fontsize=12)
plt.title('Сравнение размера моделей', fontsize=14, fontweight='bold')
plt.grid(axis='y', linestyle='--', alpha=0.7)

for bar in bars:
    height = bar.get_height()
    plt.annotate(f'{height:.1f} MB',
                 xy=(bar.get_x() + bar.get_width() / 2, height),
                 xytext=(0, 3),
                 textcoords="offset points",
                 ha='center', va='bottom', fontsize=10,
                 bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.8))

    plt.tight_layout()
    plt.savefig('model_comparison.png', dpi=300, bbox_inches='tight')
    plt.show()

    # 2. Сравнение confusion matrices
    plt.figure(figsize=(12, 5))

    # Baseline CM
    plt.subplot(1, 2, 1)
    sns.heatmap(baseline_cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['Negative', 'Positive'],
                yticklabels=['Negative', 'Positive'])
    plt.title(f'Baseline Confusion Matrix\nAccuracy: {baseline_test_acc:.4f}', fontsize=12)
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')

    # BERT CM
    plt.subplot(1, 2, 2)
    sns.heatmap(bert_cm, annot=True, fmt='d', cmap='Greens',
                xticklabels=['Negative', 'Positive'],
                yticklabels=['Negative', 'Positive'])
    plt.title(f'BERT Confusion Matrix\nAccuracy: {test_results["eval_accuracy"]:.4f}', fontsize=12)
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')

    plt.tight_layout()
    plt.savefig('confusion_matrices_comparison.png', dpi=300, bbox_inches='tight')
    plt.show()

    # 3. Сравнение по классам
    plt.figure(figsize=(10, 6))
    x = np.arange(2)  # Два класса: Negative (0), Positive (1)
    width = 0.35

    # Precision по классам
    baseline_prec = [baseline_report['0']['precision'], baseline_report['1']['precision']]
    bert_prec = [bert_report['0']['precision'], bert_report['1']['precision']]

    plt.bar(x - width/2, baseline_prec, width, label='Baseline (TF-IDF + LR)', color='#1f77b4')
    plt.bar(x + width/2, bert_prec, width, label='BERT', color='#2ca02c')

    plt.xlabel('Классы', fontsize=12)
    plt.ylabel('Precision', fontsize=12)
    plt.title('Сравнение точности (Precision) по классам', fontsize=14, fontweight='bold')
    plt.xticks(x, ['Negative', 'Positive'])
    plt.ylim(0.7, 1.0)
    plt.legend()
    plt.grid(axis='y', linestyle='--', alpha=0.7)

    for i, (b_val, bert_val) in enumerate(zip(baseline_prec, bert_prec)):
        plt.annotate(f'{b_val:.3f}', (x[i] - width/2, b_val), ha='center', va='bottom')
        plt.annotate(f'{bert_val:.3f}', (x[i] + width/2, bert_val), ha='center', va='bottom')

    plt.tight_layout()
    plt.savefig('precision_by_class.png', dpi=300, bbox_inches='tight')
    plt.show()

    # 4. Ключевые выводы
    print("\n" + "="*80)
    print("КЛЮЧЕВЫЕ ВЫВОДЫ СРАВНЕНИЯ".center(80))
    print("="*80)

    print(f"\n1. КАЧЕСТВО ПРЕДСКАЗАНИЙ:")
    print(f"   • BERT превосходит baseline по всем метрикам качества:")
    print(f"     - Accuracy:  {test_results['eval_accuracy']:.4f} vs {baseline_test_acc:.4f} (+{(test_results['eval_accuracy'] -
    print(f"     - F1-Score:  {f1_scores[1]:.4f} vs {f1_scores[0]:.4f} (+{(f1_scores[1] - f1_scores[0])*100:.2f}%)")
    print(f"   • Особенно заметно улучшение для класса 'Positive' (Precision: {bert_report['1']['precision']:.4f} vs {baseline_r

    print(f"\n2. ВЫЧИСЛИТЕЛЬНАЯ ЭФФЕКТИВНОСТЬ:")
    print(f"   • Baseline значительно быстрее в обучении: {baseline_train_time:.1f}s vs {bert_train_time:.1f}s")
    print(f"   • Baseline быстрее в инференсе: {baseline_inference_time*1000:.2f}ms vs {bert_inference_time*1000:.2f}ms на весь
    print(f"   • BERT требует в {bert_size/baseline_size:.0f} раз больше памяти для хранения модели")

    print(f"\n3. ПРАКТИЧЕСКАЯ ПРИМЕНИМОСТЬ:")
    print(f"   • Baseline подходит для:")
    print(f"     - Систем с ограниченными вычислительными ресурсами")
    print(f"     - Задач, требующих мгновенного отклика")
    print(f"     - Сценариев, где интерпретируемость важна")
```

```
print(f"   • BERT предпочтителен для:")
print(f"     - Систем, где качество критичнее скорости")
print(f"     - Задач, требующих глубокого понимания контекста")
print(f"     - Сценариев с достаточными вычислительными ресурсами")

print(f"\n4. ТОРГОВЛЯ МЕЖДУ КАЧЕСТВОМ И РЕСУРСАМИ:")
print(f"   • За {bert_train_time/baseline_train_time:.1f}x увеличение времени обучения")
print(f"   • За {bert_size/baseline_size:.0f}x увеличение размера модели")
print(f"   • Получаем {((test_results['eval_accuracy'] - baseline_test_acc)/baseline_test_acc)*100:.1f}% относительного улуч

print("\n" + "="*80)
print("РЕКОМЕНДАЦИИ".center(80))
print("="*80)
print("• Для production-систем с высокими требованиями к качеству: использовать BERT")
print("• Для embedded-устройств или систем реального времени: использовать baseline")
print("• Для баланса качества и скорости: рассмотреть distilled версии BERT (DistilBERT)")
print("• Для дальнейшего улучшения: попробовать fine-tuning с балансировкой классов и оптимизацией гиперпараметров")
```