1. ОПРЕДЕЛЕНИЕ ПРОБЛЕМЫ И ЦЕЛЕВЫЕ ТРЕБОВАНИЯ

1.1 Техническое задание

Необходимо разработать систему классификации русскоязычных текстов по трем классам тональности:

- Класс 0: Нейтральная тональность
- Класс 1: Положительная тональность
- Класс 2: Негативная тональность

Требования ТЗ:

- Модель должна работать на обычном железе (стандартные CPU/GPU)
- Выполнение должно завершаться за разумное время при использовании небольших вычислительных ресурсов
- Развертывание на бесплатном облачном сервисе
- Использование компактных моделей (RuBERT-base, не 7B-13B LLM)
- Минимальное потребление памяти

1.2 Выбор архитектурного подхода

Для решения задачи выбрана компактная трансформерная модель RuBERT-base:

- Модель: blanchefort/rubert-base-cased-sentiment
- Параметры: ~110M
- Потребление памяти: 400-500 MB
- Используется для классификации тональности текстов

Архитектура обеспечивает:

- Основной инференс с хорошим качеством
- Работу на обычном железе без требований к GPU
- Независимость от характеристик хостинга

2. АРХИТЕКТУРА РЕШЕНИЯ

2.1 Компоненты системы

Backend Service (routes.py)

- Обработка HTTP запросов от фронтенда
- Адаптивная стратегия батчинга
- Управление обработкой текстов

ML Service (app.py + inference.py)

- FastAPI эндпоинты для предсказаний
- Загрузчик моделей
- Динамическое управление ресурсами

Модель: RuBERT-base-cased

- Трансформерная архитектура BERT
- Основной путь обработки текстов

2.2 Поток данных

Frontend (CSV файл)
↓
Backend Service (routes.py)
- Парсинг CSV
- Разбиение на батчи
↓
ML Service (app.py)
- GET /health: проверка статуса
- POST /predict: один текст
- POST /predict-batch: батч текстов
↓
Inference (inference.py)
- Токенизация
- Батч-обработка текстов
↓
RuBERT Model (PyTorch)
- Forward pass через модель
- Получение logits и вероятностей
↓
Backend (результаты)
- Объединение результатов
- Форматирование ответа
↓
Frontend (output CSV)

3. АРХИТЕКТУРА МОДЕЛИ: RUBERT-BASE

3.1 Базовая архитектура

Модель: blanchefort/rubert-base-cased-sentiment
Архитектура: Transformer BERT

- Энкодер: 12 слоев

- Hidden size: 768
- Attention heads: 12
- Параметры: ~110 млн
- Максимальная длина входа: 512 токенов
- Число классов: 3 (neutral, positive, negative)

3.2 Загрузка модели (Singleton паттерн)

Процесс инициализации:

1. При старте контейнера проверяется наличие fine-tuned модели в MODEL_PATH
2. Если существует локальная версия:
   - Загружаются веса из локального хранилища
3. Если локальная модель отсутствует:
   - Загружается базовая модель из HuggingFace
   - Кэшируется для последующих запусков
4. Модель переводится в режим eval() для инференса
5. Модель переносится на доступное устройство (GPU если есть, иначе CPU)

Преимущества singleton паттерна:

- Модель загружается один раз при старте
- Повторные запросы используют загруженную модель из памяти
- Экономия ресурсов при обработке множественных запросов

3.3 Оптимизации модели

torch.compile() (PyTorch 2.0+)

- Компилирует граф вычислений в машинный код
- Режим reduce-overhead: минимизирует накладные расходы
- Применяется для ускорения инференса

torch.inference_mode()

- Контекстный менеджер для инференса
- Отключает autograd и буферизацию градиентов
- Применяется во всех местах инференса

Динамический batch_size

- Размер батча адаптируется на основе доступной памяти
- При достаточной памяти: максимальный batch_size
- При ограничениях памяти: автоматическое снижение batch_size

Управление памятью

- Явная очистка GPU кэша после каждого чанка
- Удаление временных тензоров
- Перенос на CPU перед форматированием результатов
- Мониторинг использования памяти

4. FINE-TUNING МОДЕЛИ

4.1 Датасет и предобработка

Входной датасет: train.csv

- Колонки: text, label
- Формат label: 0 (нейтральная), 1 (положительная), 2 (негативная)

Предобработка:

- Удаление пустых строк
- Удаление дубликатов
- Проверка валидности labels

Разбиение данных:

- Тренировочное множество: 90% (stratified split)
- Валидационное множество: 10% (stratified split)
- Stratification сохраняет распределение классов

4.2 Конфигурация обучения

Параметры обучения:

- Модель: blanchefort/rubert-base-cased-sentiment
- Число эпох: 4
- Batch size: 16 (обучение), 16 (валидация)
- Оптимизатор: AdamW
- Learning rate: 2e-5
- Warmup steps: 1000
- Weight decay: 0.01
- Gradient accumulation: 2
- FP16: при наличии CUDA

Стратегия валидации:

- Evaluation strategy: каждая эпоха
- Save strategy: каждая эпоха
- Load best model: лучшая по macro-F1
- Early stopping: 2 эпохи без улучшения

4.3 Метрики и результаты обучения

Вычисляемые метрики:

- Accuracy: доля правильных предсказаний
- Macro-F1: усредненная F1 по классам без веса
- Weighted-F1: усредненная F1 с учетом распределения классов
- Per-class Precision, Recall, F1

4.4 Сохранение модели

После обучения сохраняются:

- pytorch_model.bin: веса модели
- config.json: конфигурация архитектуры
- tokenizer.json: конфигурация токенайзера
- training_report.txt: отчет с результатами

Директория сохранения: ./models/rubert-finetuned/

5. ИНФЕРЕНС И ОБРАБОТКА ТЕКСТОВ

5.1 Одиночный инференс (predict)

Процесс для одного текста:

1. Токенизация текста:
   - Преобразование в токены
   - Обрезание до max_length=512
   - Паддинг до нужной длины
   - Возвращаемый формат: PyTorch тензоры
2. Перенос на устройство:
   - Все тензоры переносятся на доступное устройство (GPU или CPU)
3. Forward pass:
   - Используется torch.inference_mode()
   - Модель обрабатывает входные тензоры
   - Получаются logits размера [1, 3]
4. Post-processing:
   - Softmax для получения вероятностей
   - Argmax для получения класса с максимальной вероятностью
   - Извлечение уверенности

Возвращаемый результат:
{
'label': 0, # или 1, или 2

'label_name': 'нейтральная',
'confidence': 0.9534,
'probabilities': {
'нейтральная': 0.9534,
'положительная': 0.0324,
'негативная': 0.0142
}
}

5.2 Батч-инференс (predict_batch)

Двухуровневая система батчинга:

Уровень 1: HTTP-батчи (Backend → ML Service)

- Размер определяется адаптивно в зависимости от объема входных данных
- Малые объемы обрабатываются целиком
- Большие объемы разбиваются на управляемые батчи

Уровень 2: Inference-батчи (ML Service)

- Размер определяется на основе доступных ресурсов
- Начальный размер выбирается оптимально
- При необходимости: автоматическое снижение размера

Механизм адаптации:

- Система отслеживает использование памяти
- При превышении пороговых значений: снижение batch_size
- Градуальное снижение при каждом срабатывании
- Минимальный размер: обработка по одному тексту

Процесс обработки:

1. Получение батча на ML Service
2. Проверка доступной памяти
3. Обработка RuBERT-base с адаптированным batch_size
4. При ошибке нехватки памяти:
    - Снижение batch_size
    - Повторная попытка обработки
5. Объединение результатов

5.3 Батч-токенизация

Оптимизированная обработка батча текстов:

1. Батч-токенизация (все тексты сразу)
2. Перенос на устройство одной операцией
3. Векторизованный инференс для всего батча
4. Перенос результатов на CPU батчем
5. Явная очистка памяти GPU
6. API ЭНДПОИНТЫ

6.1 GET /health

Проверка состояния ML Service:
{
"status": "ready",
"model": "rubert-base-cased-sentiment",
"device": "cuda" или "cpu",
"available_memory_mb": 1024,
"version": "1.0"
}

6.2 POST /predict

Анализ одного текста:
Input: {"text": "Отличный фильм, очень понравился!"}
Output: {
"label": 1,
"label_name": "положительная",
"confidence": 0.9876,
"probabilities": {
"нейтральная": 0.0089,
"положительная": 0.9876,
"негативная": 0.0035
}
}

6.3 POST /predict-batch

Анализ батча текстов:
Input: {"texts": ["Отличный фильм!", "Ужасная книга", "Обычный фильм"]}
Output: {
"results": [
{
"label": 1,
"label_name": "положительная",
"confidence": 0.9876,
"probabilities": {...}
},

{
"label": 2,
"label_name": "негативная",
"confidence": 0.8945,
"probabilities": {...}
},
{
"label": 0,
"label_name": "нейтральная",
"confidence": 0.7654,
"probabilities": {...}
}
]
}

7. УПРАВЛЕНИЕ РЕСУРСАМИ

7.1 Потребление памяти

RuBERT-base:

- Веса модели: 400-500 MB
- Токенайзер: 10-20 MB
- Промежуточные активации при инференсе: зависит от batch_size
- При batch_size=128: дополнительно 300-500 MB
- При batch_size=32: дополнительно 100-200 MB
- При batch_size=1: дополнительно 20-50 MB

Адаптивное управление:

- Система отслеживает использование памяти
- Автоматически снижает batch_size при приближении к лимитам
- Обеспечивает работу на различных конфигурациях хостингов

7.2 Вычислительные требования

Поддерживаемые устройства:

- CPU: любой современный процессор (Intel, AMD, ARM)
- GPU: NVIDIA GPU с поддержкой CUDA
- Автоматический выбор доступного устройства

Совместимость:

- PyTorch 1.9+
- Python 3.8+

- Работает на Linux, macOS, Windows

7.3 Хранение

Требования к диску:

- Модель RuBERT: 500 MB
- Исходный код: ~50 MB
- Логи: переменное количество

8. ОБРАБОТКА ДАННЫХ

8.1 Форматы входных данных

CSV файл с колонками:

- ID: уникальный идентификатор текста
- text: текст на русском языке

Валидация:

- Проверка наличия обязательных колонок
- Проверка типов данных
- Пропуск пустых значений с логированием

8.2 Форматы выходных данных

CSV файл с колонками:

- ID: скопировано из входного файла
- label: одно из значений {0, 1, 2}

Гарантии:

- Лейбл всегда в диапазоне {0, 1, 2}
- ID скопирован без изменений
- Порядок сохранен

8.3 Нормализация текста

Обработка текста:

- Нормализация кодировки (UTF-8)
- Обработка специальных символов
- Сохранение исходного содержимого
- Преобразование в нижний регистр при необходимости (внутри модели)

9. ЛОГИРОВАНИЕ И МОНИТОРИНГ

9.1 Логируемые события

При инициализации:

- Загрузка модели: успешно или ошибка
- Доступные ресурсы: память, устройство
- Статус готовности

При обработке:

- Текущие batch_size
- Использование памяти
- Статус обработки батчей

9.2 Уровни логирования

INFO: основные события (инициализация, обработка, завершение)
WARNING: потенциальные проблемы
ERROR: ошибки обработки
DEBUG: детальная информация

9.3 Метрики системы

Отслеживаемые метрики:

- Количество обработанных текстов
- Использование памяти
- Адаптации управления ресурсами

10. АРХИТЕКТУРНЫЕ КОМПОНЕНТЫ

10.1 Модульность

Система спроектирована модульно:

- Инференс логика отделена от API
- Модели загружаются через единый интерфейс
- Каждый компонент может быть заменен независимо

10.2 Расширяемость

Возможности расширения:

- Добавление новых моделей через единый интерфейс
- Настройка параметров обучения и инференса
- Модификация стратегий батчинга

10.3 Надежность

Обеспечение надежности:

- Проверка входных данных
- Обработка ошибок на каждом этапе
- Логирование для отладки
- Корректная очистка ресурсов

11. КЛАССИФИКАЦИЯ И РЕЗУЛЬТАТЫ

11.1 Схема классификации

Три класса тональности:

- Класс 0: Нейтральная тональность
- Класс 1: Положительная тональность
- Класс 2: Негативная тональность

11.2 Выходные данные модели

Для каждого текста модель возвращает:

- Предсказанный класс (label)
- Текстовое описание класса (label_name)
- Уверенность в предсказании (confidence)
- Вероятности для каждого класса (probabilities)

11.3 Обработка граничных случаев

Специальные случаи:

- Пустой текст: обрабатывается корректно
- Очень длинный текст: обрезается до 512 токенов
- Специальные символы: нормально обрабатываются
- Эмодзи: токенизируются как отдельные элементы

12. ТЕХНИЧЕСКАЯ РЕАЛИЗАЦИЯ

12.1 Используемые библиотеки

Основные зависимости:

- PyTorch: фреймворк для глубокого обучения
- Transformers (HuggingFace): моделиархитектура и загрузка
- FastAPI: создание API эндпоинтов
- Pandas: обработка CSV файлов
- NumPy: численные операции

12.2 Версионирование

Совместимость:

- PyTorch: 1.9+
- Transformers: 4.0+
- Python: 3.8+
- Поддержка как CPU, так и GPU

12.3 Развертывание

Способы запуска:

- Локально: python app.py
- Docker: контейнеризованный запуск
- Облако: поддержка основных облачных платформ

13. ИНФОРМАЦИЯ О МОДЕЛИ

13.1 Происхождение модели

Модель:

- Разработана: blanchefort (HuggingFace Hub)
- База: rubert-base-cased
- Адаптация: для задачи анализа тональности
- Лицензия: открытая (Apache 2.0)

13.2 Обучение модели

Процесс обучения:

- Используется Hugging Face Trainer
- Применяются стандартные техники fine-tuning
- Валидация на отдельном множестве
- Сохранение лучшей модели

13.3 Документирование модели

Документы модели:

- config.json: конфигурация архитектуры
- tokenizer_config.json: параметры токенайзера
- training_report.txt: результаты обучения

14. ИТОГОВОЕ ОПИСАНИЕ

Решение состоит из следующих компонентов:

Основная модель:

- RuBERT-base (110M параметров)
- Требование памяти: 400-500 MB
- Используется для классификации тональности текстов

Система инференса:

- Двухуровневое батчинг (HTTP-батчи и inference-батчи)
- Адаптивное управление batch_size
- Динамическое управление ресурсами

API сервис:

- FastAPI для обработки запросов
- Три эндпоинта: /health, /predict, /predict-batch
- Возвращает предсказания в структурированном формате

Обработка данных:

- CSV входной формат с колонками ID и text
- CSV выходной формат с колонками ID и label
- Гарантия корректности лейблов (всегда {0, 1, 2})

Архитектура:

- Модульная структура
- Отделение инференс логики от API
- Возможность расширения и модификации
- Компактная реализация