**Geekbrains**

**Разработка и реализация программного приложения, которое позволяет обучать языковую модель GPT на пользовательских данных и генерировать текстовые ответы на запросы пользователей.**

**Искусственный интеллект. Специалист**

**Шмаков Игорь Александрович**

**Москва**

**2025**

**Введение**

**1.1 Актуальность темы**

Современные информационные технологии активно развиваются, и обработка естественного языка становится одним из ключевых направлений исследований. Языковые модели, такие как GPT, позволяют создавать интеллектуальные системы, способные понимать и генерировать тексты на уровне человеческого понимания. Это открывает новые возможности для автоматизации процессов, улучшения качества обслуживания клиентов и повышения эффективности бизнеса.

Целью данной работы является разработка и реализация программного приложения, которое позволяет обучать языковую модель GPT на пользовательских данных и генерировать текстовые ответы на запросы пользователей. Приложение предназначено для использования на устройствах с ограниченными ресурсами, такими как ноутбуки и персональные компьютеры без графического ускорителя.

**1.2 Цель и задачи проекта**

**Цель:** Разработать приложение для обучения и использования языковой модели GPT на персональных компьютерах без GPU.

Задачи:

* Изучить архитектуру и принципы работы языковых моделей типа GPT.
* Освоить методы трансферного обучения и обработки естественного языка.
* Спроектировать и реализовать архитектуру приложения, обеспечивающую эффективное обучение и генерацию текста.
* Создать удобный интерфейс для взаимодействия с моделью.
* Провести тестирование и отладку приложения.

**1.3 Используемые технологии и инструменты**

* Python — основной язык программирования.
* Transformers библиотека — для работы с моделями GPT.
* PyTorch — фреймворк глубокого обучения.
* PyQt5 — библиотека для разработки графического интерфейса.
* HDF5 — формат хранения больших наборов данных.
* Hugging Face — платформа для загрузки предобученных моделей и набора данных.

**Теоретическая часть**

**2.1 Обзор языковых моделей и архитектуры GPT**

**GPT-модели** (Generative Pre-trained Transformer) представляют собой класс языковых моделей, основанных на архитектуре трансформеров. В основе работы модели лежит механизм самовнимания (self-attention), позволяющий учитывать контекст при обработке текста.

Архитектура GPT включает следующие ключевые компоненты:

* **Encoder-Decoder** структура
* Механизм **multi-head attention**
* **Feed Forward** слои
* **Residual connections**
* **Layer Normalization**

В приложении используется модель **ai-forever/rugpt3small\_based\_on\_gpt2** для работы с русским языком.

**2.2 Принципы трансферного обучения**

**Трансферное обучение** позволяет использовать предварительно обученные модели для решения конкретных задач. В приложении реализован механизм загрузки базовой модели и её дообучения на специфическом датасете:

self.model = GPT2LMHeadModel.from\_pretrained(model\_name)

self.tokenizer = GPT2Tokenizer.from\_pretrained(model\_name)

Процесс трансферного обучения включает следующие этапы:

1. Загрузка базовой модели
2. Настройка параметров обучения
3. Дообучение на целевом датасете
4. Сохранение дообученной модели

**2.3 Методы обработки естественного языка**

В приложении реализованы следующие методы обработки текста:

* Токенизация текста
* Очистка данных
* Фильтрация датасета
* Генерация текста с учетом параметров (max\_length, temperature)

Основные этапы обработки текста:

1. Предварительная обработка (чистка данных)
2. Токенизация
3. Трансформация в тензоры
4. Генерация текста

***def tokenize\_function(self, examples):***

***texts = []***

***for dialog in examples['conversation']:***

***if isinstance(dialog, list) and len(dialog) > 0:***

***last\_msg = dialog[-1]***

***if isinstance(last\_msg, dict) and 'content' in last\_msg:***

***texts.append(last\_msg['content'])***

***return self.tokenizer(***

***texts,***

***truncation=True,***

***padding='max\_length',***

***max\_length=256,***

***return\_tensors="pt"***

***)***

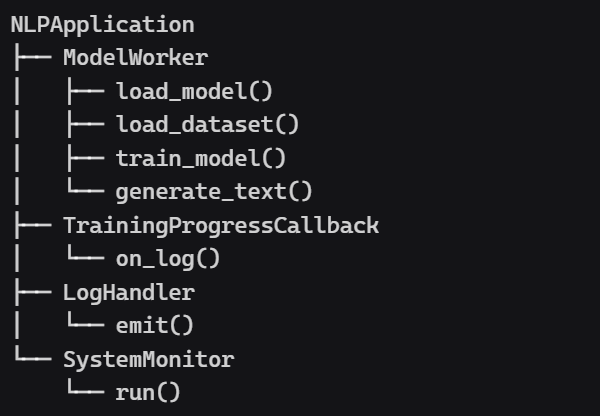
**Практическая часть**

**3.1 Проектирование архитектуры приложения**

Архитектура приложения включает следующие компоненты:

* **ModelWorker** - поток для выполнения операций с моделью
* **TrainingProgressCallback** - обратный вызов для отслеживания прогресса обучения
* **LogHandler** - обработчик логов для вывода в интерфейс
* **SystemMonitor** - мониторинг системы

Структура приложения:



**3.2 Реализация функционала обучения модели**

Процесс обучения включает:

* Подготовку датасета
* Токенизацию данных
* Настройка параметров обучения:

***training\_args = TrainingArguments(***

***output\_dir="./model\_output",***

***overwrite\_output\_dir=True,***

***num\_train\_epochs=3,***

***per\_device\_train\_batch\_size=2,***

***gradient\_accumulation\_steps=4,***

***learning\_rate=5e-5,***

***weight\_decay=0.01,***

***save\_steps=200,***

***logging\_steps=20,***

***disable\_tqdm=False,***

***use\_cpu=True,***

***remove\_unused\_columns=False,***

***report\_to=None)***

Процесс обучения модели:

1. Загрузка и токенизация данных
2. Настройка параметров обучения
3. Инициализация трейнера
4. Запуск процесса обучения

***trainer = Trainer(***

***model=self.model,***

***args=training\_args,***

***train\_dataset=tokenized\_dataset["train"],***

***data\_collator=self.data\_collator,***

***callbacks=[callback]***

***)***

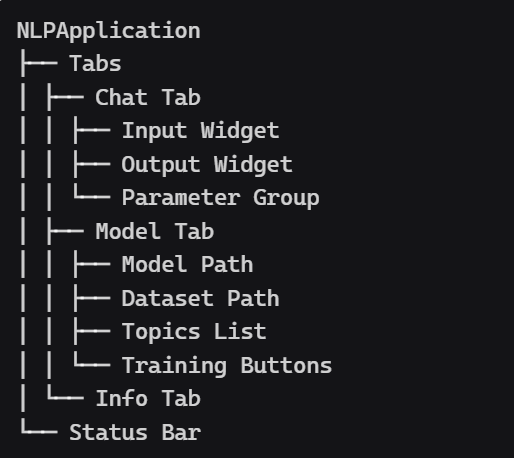
***trainer.train()***

**3.3 Разработка пользовательского интерфейса**

Интерфейс приложения включает следующие компоненты:

* **Main Window** - основное окно приложения
* **Tabs** - вкладки для разных функциональных блоков
* **Widgets** - элементы управления
* **Logging System** - система логирования

Структура интерфейса:



Основные элементы управления:

* **QTextEdit** - для ввода и вывода текста
* **QListWidget** - для отображения списка тем
* **QSpinBox** - для настройки параметров генерации
* **QPushButton** - кнопки управления

**3.4 Тестирование и отладка приложения**

1. Генерация текста
2. Мониторинг производительности

Тестирование проводилось на следующих этапах:

* **Unit Testing** - проверка отдельных компонентов
* **Integration Testing** - проверка взаимодействия компонентов
* **System Testing** - проверка работы всего приложения

Основные сценарии тестирования:

1. Загрузка модели и датасета
2. Валидация данных

***def validate\_dataset(self, dataset):  
 """Проверка целостности данных в датасете"""  
 try:  
 error\_count = 0  
 total\_samples = len(dataset['train'])  
  
 sample\_size = min(100, total\_samples)  
 sample = dataset['train'].select(range(sample\_size))  
  
 for i, example in enumerate(sample):  
 if not isinstance(example, dict):  
 error\_count += 1  
 continue  
  
 if 'conversation' not in example:  
 error\_count += 1  
 continue  
  
 dialog = example['conversation']  
 if not isinstance(dialog, list) or len(dialog) == 0:  
 error\_count += 1  
 continue  
  
 last\_msg = dialog[-1]  
 if not isinstance(last\_msg, dict) or 'content' not in last\_msg:  
 error\_count += 1  
 continue  
  
 error\_percent = (error\_count / sample\_size) \* 100  
  
 if error\_percent > 5:  
 return False, f"Обнаружено {error\_count} ошибок в {sample\_size} примерах ({error\_percent:.1f}%)"  
  
 return True, f"Проверено {sample\_size} примеров, ошибок: {error\_count}"  
  
 except Exception as e:  
 return False, f"Ошибка проверки данных: {str(e)}"***

**Заключение**

**4.1 Результаты работы**

В результате разработки создано приложение, позволяющее:

* Загружать и обучать языковые модели
* Работать с датасетами
* Генерировать текст на русском языке
* Отслеживать процесс обучения
* Мониторить системные ресурсы

Основные технические характеристики:

* Поддержка многопоточности
* Система логирования
* Графический интерфейс
* Мониторинг производительности

**4.2 Перспективы развития**

Направления дальнейшего развития:

* Расширение функционала фильтрации датасета
* Оптимизация производительности
* Добавление новых языковых моделей
* Реализация многопользовательского режима
* Интеграция с облачными сервисами

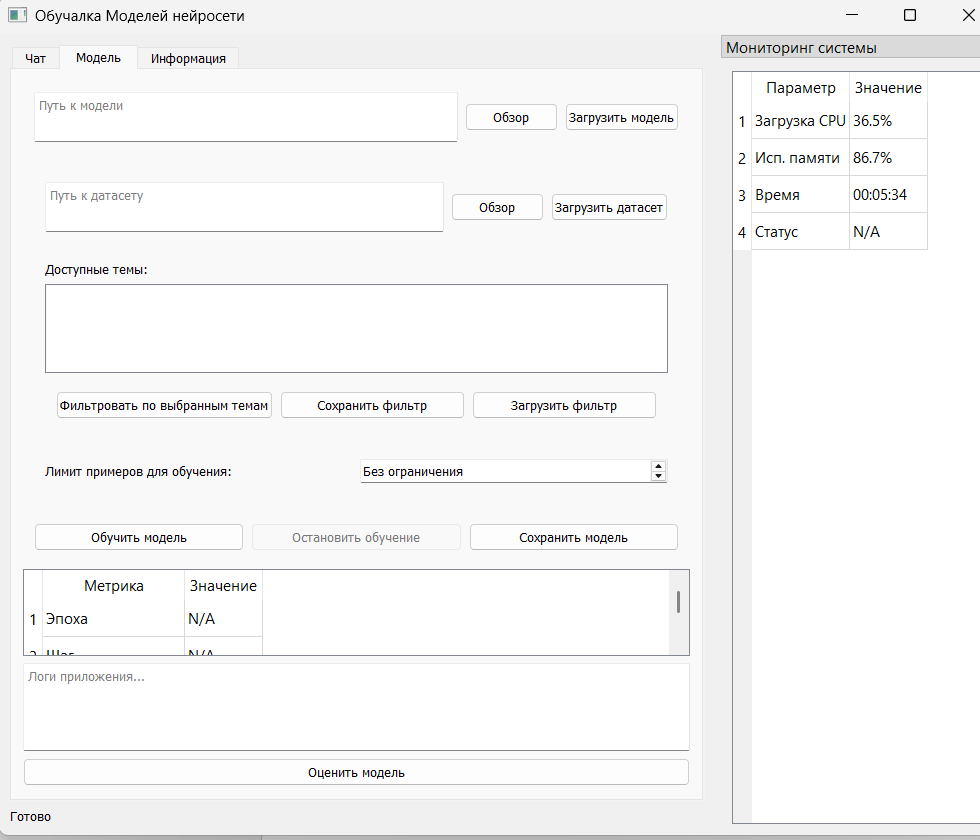
**Список литературы**

1. Официальный сайт Hugging Face - документация по библиотеке transformers
2. Документация PyQt5 - руководство по разработке GUI приложений
3. Материалы по GPT-моделям - научные статьи и исследования
4. Научные статьи по трансферному обучению
5. Документация библиотеки datasets
6. Руководство по PyTorch

**Приложения**

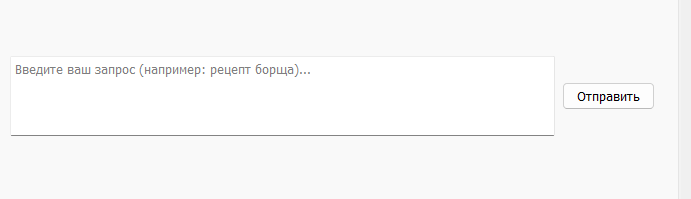
**Приложение 1. Скриншоты интерфейса**

1. **Главное окно приложения**
   * Основные вкладки

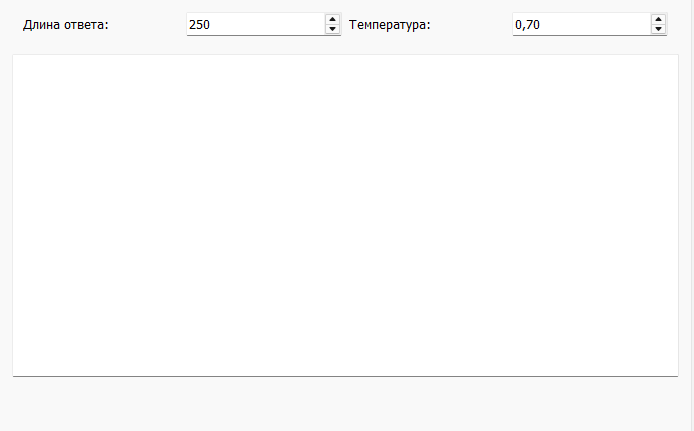


* + Панель мониторинга
  + Элементы управления

1. **Вкладка чата**
   * Поле ввода запроса



* + Область вывода ответа



1. **Вкладка модели**
   * Настройка пути к модели
   * Управление датасетом
   * Параметры обучения
2. **Вкладка информации**
   * Инструкция по использованию
   * Советы по настройке

**Приложение 2.**

**Примеры генерации текста**

1. **Пример 1: Генерация рецепта**
   * Запрос: “рецепт борща”
   * **Бот:** : - 1/2 стакана бульона (можно добавить свежую зелень), соль и перец по вкусу. если используете сметану, добавьте ее к луку для соуса или салату из свежих овощей с добавлением сметаны; - для заправки можно использовать немного воды комнатной температуры без горстки соли на ваш выбор – около 80 мл в зависимости от размера порции готового супа "борщ".
   * Параметры генерации: длина 100 токенов, температура 0.7

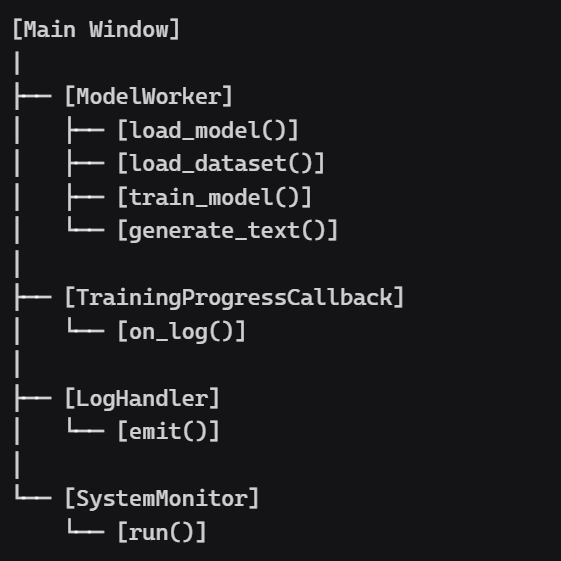
**Приложение 3. Логи работы приложения**

1. **Логи загрузки модели**
2. 2025-06-10 12:00:00 - INFO - Загрузка модели из ./models
3. 2025-06-10 12:00:15 - INFO - Модель успешно загружена
4. **Логи обучения**
5. 2025-06-10 12:10:00 - INFO - Начало обучения
6. 2025-06-10 12:10:05 - INFO - Эпоха 1/3
7. 2025-06-10 12:15:00 - INFO - Потери: 0.05
8. **Логи генерации текста**
9. 2025-06-10 12:20:00 - INFO - Генерация текста
10. 2025-06-10 12:20:05 - INFO - Длина ответа: 250 токенов

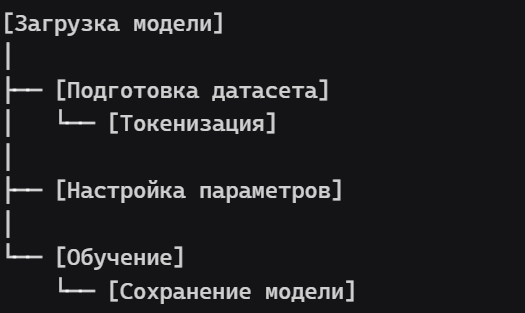
**Приложение 4.**

**Блок-схемы**

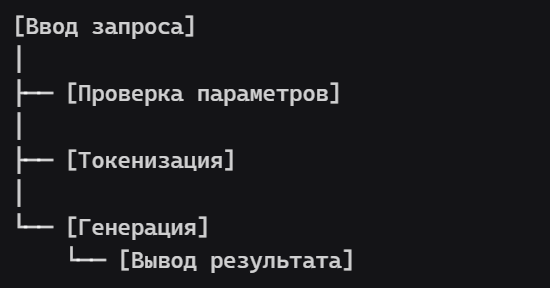
1. **Блок-схема архитектуры приложения**



1. **Блок-схема процесса обучения**



1. **Блок-схема генерации текста**



**Приложение 5.**

**Исходный код приложения**

import json  
import os  
import sys  
import re  
import logging  
from typing import Optional, Dict, List  
from datetime import datetime  
  
import torch  
from datasets import DatasetDict, load\_from\_disk  
from transformers import GPT2LMHeadModel, GPT2Tokenizer, Trainer, TrainingArguments  
from transformers import DataCollatorForLanguageModeling, TrainerCallback  
from PyQt5.QtWidgets import (QApplication, QMainWindow, QVBoxLayout, QHBoxLayout,  
 QWidget, QLabel, QTextEdit, QPushButton, QFileDialog,  
 QMessageBox, QTabWidget, QProgressBar, QListWidget,  
 QDockWidget, QTableWidget, QTableWidgetItem, QSpinBox,  
 QDoubleSpinBox)  
from PyQt5.QtCore import QThread, pyqtSignal, QObject, Qt, QTimer  
  
  
class LogHandler(logging.Handler):  
 *"""Этот класс обрабатывает вывод логов и направляет их в графический интерфейс приложения (QTextEdit),  
 позволяя видеть информацию о ходе выполнения процессов."""* def \_\_init\_\_(self, text\_edit):  
 super().\_\_init\_\_()  
 self.text\_edit = text\_edit  
 self.setFormatter(logging.Formatter('%(asctime)s - %(levelname)s - %(message)s'))  
  
 def emit(self, record):  
 msg = self.format(record)  
 if record.levelno >= logging.ERROR:  
 self.text\_edit.append(f"<font color='red'>{msg}</font>")  
 elif record.levelno >= logging.WARNING:  
 self.text\_edit.append(f"<font color='orange'>{msg}</font>")  
 else:  
 self.text\_edit.append(msg)  
 self.text\_edit.verticalScrollBar().setValue(self.text\_edit.verticalScrollBar().maximum())  
  
  
class TextEditStream:  
 *"""Класс для перенаправления stdout/stderr в QTextEdit  
 Стандартные потоки вывода перенаправляются таким образом,  
 чтобы любая консольная информация отображалась прямо в окне приложения."""* def \_\_init\_\_(self, text\_edit, is\_error=False):  
 self.text\_edit = text\_edit  
 self.is\_error = is\_error  
  
 def write(self, message):  
 if message.strip():  
 if self.is\_error:  
 self.text\_edit.append(f"<font color='red'>{message}</font>")  
 else:  
 self.text\_edit.append(message)  
  
 def flush(self):  
 pass  
  
  
# Настройка основного логгера  
logger = logging.getLogger(\_\_name\_\_)  
logger.setLevel(logging.INFO)  
  
  
class SystemMonitor(QObject):  
 update\_stats = pyqtSignal(dict)  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.running = True  
  
 def run(self):  
 import psutil  
 while self.running:  
 try:  
 stats = {  
 'cpu': psutil.cpu\_percent(),  
 'memory': psutil.virtual\_memory().percent,  
 'time': datetime.now().strftime("%H:%M:%S"),  
 'status': "Работает"  
 }  
 self.update\_stats.emit(stats)  
 QThread.msleep(2000)  
 except Exception as e:  
 logger.error(f"Ошибка мониторинга: {str(e)}")  
  
  
class TrainingProgressCallback(QObject, TrainerCallback):  
 update\_progress = pyqtSignal(str, str, dict)  
  
 def on\_log(self, args, state, control, logs=None, \*\*kwargs):  
 if logs:  
 try:  
 metrics = {  
 'epoch': f"{state.epoch:.1f}/{state.num\_train\_epochs}",  
 'step': f"{state.global\_step}/{state.max\_steps}",  
 'loss': f"{logs.get('loss', 'N/A'):.4f}",  
 'learning\_rate': f"{logs.get('learning\_rate', 'N/A'):.6f}",  
 'speed': f"{logs.get('speed', 'N/A')} samples/sec" if 'speed' in logs else 'N/A'  
 }  
 self.update\_progress.emit("Обучение", "прогресс", metrics)  
 except Exception as e:  
 error\_msg = f"Ошибка форматирования логов: {str(e)}"  
 self.update\_progress.emit(error\_msg, "ошибка", {})  
  
  
class ModelWorker(QThread):  
 *"""  
 Это фоновый рабочий поток,  
 ответственный за выполнение всех трудоемких операций (загрузка модели, обучение, генерация текста и оценка).  
 """* update\_signal = pyqtSignal(str, str)  
 progress\_signal = pyqtSignal(int)  
 complete\_signal = pyqtSignal(bool, str)  
 topics\_loaded = pyqtSignal(list, bool)  
 update\_progress = pyqtSignal(str, str, dict)  
  
 def \_\_init\_\_(self, operation: str, \*\*kwargs):  
 super().\_\_init\_\_()  
 self.operation = operation  
 self.kwargs = kwargs  
 self.model = None  
 self.tokenizer = None  
 self.dataset = None  
 self.data\_collator = None  
 self.tokenized\_count = 0  
 self.stop\_requested = False  
  
 def run(self):  
 try:  
 if self.stop\_requested:  
 return  
  
 if self.operation == 'load\_model':  
 self.load\_model()  
 elif self.operation == 'load\_dataset':  
 self.load\_dataset()  
 success, topics = self.analyze\_dataset()  
 self.topics\_loaded.emit(topics if success else [], success)  
 elif self.operation == 'filter\_dataset':  
 success, message = self.filter\_dataset(self.kwargs.get('selected\_topics', []))  
 self.complete\_signal.emit(success, message)  
 elif self.operation == 'train':  
 self.train\_model()  
 elif self.operation == 'generate':  
 success, result = self.generate\_text()  
 self.complete\_signal.emit(success, result)  
 elif self.operation == 'evaluate':  
 self.evaluate\_model()  
 except Exception as e:  
 self.complete\_signal.emit(False, f"Ошибка: {str(e)}")  
 logger.error(f"Ошибка в ModelWorker: {str(e)}")  
  
 def request\_stop(self):  
 self.stop\_requested = True  
  
 def load\_model(self):  
 model\_path = self.kwargs.get('model\_path')  
  
 if model\_path and os.path.exists(model\_path):  
 self.update\_signal.emit(f"Загрузка модели из {model\_path}", "модель")  
 self.model = GPT2LMHeadModel.from\_pretrained(model\_path)  
 self.tokenizer = GPT2Tokenizer.from\_pretrained(model\_path)  
 self.complete\_signal.emit(True, f"Модель загружена из {model\_path}")  
 else:  
 self.update\_signal.emit("Загрузка русскоязычной модели из сети...", "модель")  
 model\_name = "ai-forever/rugpt3small\_based\_on\_gpt2"  
 self.model = GPT2LMHeadModel.from\_pretrained(model\_name)  
 self.tokenizer = GPT2Tokenizer.from\_pretrained(model\_name)  
 self.complete\_signal.emit(True, "Русскоязычная модель загружена")  
  
 def load\_dataset(self):  
 dataset\_path = self.kwargs.get('dataset\_path')  
 self.update\_signal.emit(f"Загрузка датасета из {dataset\_path}", "датасет")  
  
 try:  
 if os.path.exists(os.path.join(dataset\_path, "dataset\_dict.json")):  
 self.dataset = load\_from\_disk(dataset\_path)  
 else:  
 train\_path = os.path.join(dataset\_path, "train")  
 test\_path = os.path.join(dataset\_path, "test")  
  
 if os.path.exists(train\_path):  
 train\_ds = load\_from\_disk(train\_path)  
 test\_ds = load\_from\_disk(test\_path) if os.path.exists(test\_path) else None  
  
 self.dataset = DatasetDict({  
 "train": train\_ds,  
 "test": test\_ds if test\_ds else train\_ds.select(range(1000))  
 })  
 else:  
 raise ValueError("Не найдены папки train/test")  
  
 if not isinstance(self.dataset, DatasetDict):  
 raise ValueError("Загруженные данные должны быть DatasetDict")  
  
 self.update\_signal.emit("Проверка целостности данных...", "датасет")  
 valid, msg = self.validate\_dataset(self.dataset)  
 if not valid:  
 raise ValueError(f"Проблемы с данными: {msg}")  
 self.update\_signal.emit(msg, "датасет")  
  
 self.dataset = self.clean\_dataset(self.dataset)  
 self.update\_signal.emit("Датасет очищен от некорректных записей", "датасет")  
  
 self.complete\_signal.emit(True, "Датасет успешно загружен и проверен")  
  
 except Exception as e:  
 raise Exception(f"Ошибка загрузки датасета: {str(e)}")  
  
 def validate\_dataset(self, dataset):  
 *"""Проверка целостности данных в датасете"""* try:  
 error\_count = 0  
 total\_samples = len(dataset['train'])  
  
 sample\_size = min(100, total\_samples)  
 sample = dataset['train'].select(range(sample\_size))  
  
 for i, example in enumerate(sample):  
 if not isinstance(example, dict):  
 error\_count += 1  
 continue  
  
 if 'conversation' not in example:  
 error\_count += 1  
 continue  
  
 dialog = example['conversation']  
 if not isinstance(dialog, list) or len(dialog) == 0:  
 error\_count += 1  
 continue  
  
 last\_msg = dialog[-1]  
 if not isinstance(last\_msg, dict) or 'content' not in last\_msg:  
 error\_count += 1  
 continue  
  
 error\_percent = (error\_count / sample\_size) \* 100  
  
 if error\_percent > 5:  
 return False, f"Обнаружено {error\_count} ошибок в {sample\_size} примерах ({error\_percent:.1f}%)"  
  
 return True, f"Проверено {sample\_size} примеров, ошибок: {error\_count}"  
  
 except Exception as e:  
 return False, f"Ошибка проверки данных: {str(e)}"  
  
 def clean\_dataset(self, dataset):  
 *"""Очистка датасета от некорректных записей"""* def is\_valid\_example(example):  
 try:  
 if not isinstance(example, dict):  
 return False  
 if 'conversation' not in example:  
 return False  
 dialog = example['conversation']  
 if not isinstance(dialog, list) or len(dialog) == 0:  
 return False  
 last\_msg = dialog[-1]  
 if not isinstance(last\_msg, dict) or 'content' not in last\_msg:  
 return False  
 return True  
 except:  
 return False  
  
 cleaned\_dataset = dataset.filter(is\_valid\_example)  
 removed = len(dataset['train']) - len(cleaned\_dataset['train'])  
  
 if removed > 0:  
 self.update\_signal.emit(  
 f"Удалено {removed} некорректных записей (осталось {len(cleaned\_dataset['train'])})",  
 "датасет"  
 )  
  
 return cleaned\_dataset  
  
 def analyze\_dataset(self):  
 if not self.dataset:  
 return False, "Датасет не загружен"  
  
 try:  
 topic\_fields = ['topic', 'classified\_topic', 'category', 'label']  
 available\_fields = set(self.dataset['train'].column\_names)  
 topic\_field = next((f for f in topic\_fields if f in available\_fields), None)  
  
 if not topic\_field:  
 return False, "В датасете нет информации о темах"  
  
 sample = self.dataset['train'].select(range(min(1000, len(self.dataset['train']))))  
 topics = list(set(sample[topic\_field]))  
  
 return True, topics  
  
 except Exception as e:  
 return False, f"Ошибка анализа тем: {str(e)}"  
  
 def filter\_dataset(self, selected\_topics):  
 if not self.dataset:  
 return False, "Датасет не загружен"  
  
 if not selected\_topics:  
 return True, "Фильтрация не применена (не выбраны темы)"  
  
 try:  
 topic\_fields = ['topic', 'classified\_topic', 'category', 'label']  
 available\_fields = set(self.dataset['train'].column\_names)  
 topic\_field = next((f for f in topic\_fields if f in available\_fields), None)  
  
 if not topic\_field:  
 return False, "Не найдено поле с темами для фильтрации"  
  
 self.update\_signal.emit(f"Фильтрация по темам: {', '.join(selected\_topics)}", "датасет")  
  
 def filter\_function(example):  
 return example[topic\_field] in selected\_topics  
  
 filtered\_dataset = self.dataset.filter(filter\_function)  
 self.dataset = filtered\_dataset  
  
 msg = (f"Фильтрация завершена\n"  
 f"Осталось примеров: {len(self.dataset['train'])}\n"  
 f"Выбранные темы: {', '.join(selected\_topics)}")  
  
 return True, msg  
  
 except Exception as e:  
 return False, f"Ошибка фильтрации: {str(e)}"  
  
 def tokenize\_function(self, examples):  
 if self.stop\_requested:  
 raise RuntimeError("Операция отменена пользователем")  
  
 processed = self.tokenized\_count % 100  
 if processed == 0:  
 self.update\_signal.emit(f"Токенизировано {self.tokenized\_count} примеров...", "подготовка")  
 self.tokenized\_count += len(examples['conversation'])  
  
 texts = []  
 for dialog in examples['conversation']:  
 if isinstance(dialog, list) and len(dialog) > 0:  
 last\_msg = dialog[-1]  
 if isinstance(last\_msg, dict) and 'content' in last\_msg:  
 texts.append(last\_msg['content'])  
 else:  
 texts.append(str(last\_msg))  
  
 if not texts:  
 raise ValueError("Не удалось извлечь текст для токенизации")  
  
 return self.tokenizer(  
 texts,  
 truncation=True,  
 padding='max\_length',  
 max\_length=256,  
 return\_tensors="pt"  
 )  
  
 def train\_model(self):  
 if self.stop\_requested:  
 return  
  
 if not self.model or not self.tokenizer:  
 raise ValueError("Модель не загружена")  
 if not self.dataset:  
 raise ValueError("Датасет не загружен")  
  
 self.tokenized\_count = 0  
 self.update\_signal.emit("Начало подготовки данных...", "подготовка")  
  
 try:  
 sample = self.dataset['train'][0]  
 sample\_preview = str(sample)[:70].replace('\n', ' ') + "..."  
 self.update\_signal.emit(f"Пример данных: {sample\_preview}", "подготовка")  
  
 max\_samples = self.kwargs.get('max\_samples')  
 if max\_samples and max\_samples > 0 and len(self.dataset['train']) > max\_samples:  
 self.dataset['train'] = self.dataset['train'].select(range(max\_samples))  
 self.update\_signal.emit(f"Датасет ограничен до {max\_samples} примеров", "подготовка")  
  
 tokenized\_dataset = self.dataset.map(  
 self.tokenize\_function,  
 batched=True,  
 batch\_size=16,  
 remove\_columns=self.dataset['train'].column\_names  
 )  
  
 if len(tokenized\_dataset['train']) == 0:  
 raise ValueError("Нет данных после токенизации")  
  
 self.update\_signal.emit(f"Токенизация завершена. Примеров: {len(tokenized\_dataset['train'])}", "подготовка")  
  
 self.data\_collator = DataCollatorForLanguageModeling(  
 tokenizer=self.tokenizer,  
 mlm=False  
 )  
  
 training\_args = TrainingArguments(  
 output\_dir="./model\_output",  
 overwrite\_output\_dir=True,  
 num\_train\_epochs=3,  
 per\_device\_train\_batch\_size=2,  
 gradient\_accumulation\_steps=4,  
 learning\_rate=5e-5,  
 weight\_decay=0.01,  
 save\_steps=200,  
 logging\_steps=20,  
 disable\_tqdm=False,  
 use\_cpu=True,  
 remove\_unused\_columns=False,  
 report\_to=None  
 )  
 callback = TrainingProgressCallback()  
  
 trainer = Trainer(  
 model=self.model,  
 args=training\_args,  
 train\_dataset=tokenized\_dataset["train"],  
 data\_collator=self.data\_collator,  
 callbacks=[callback]  
 )  
  
 self.update\_signal.emit("Начало обучения модели...", "обучение")  
 trainer.train()  
  
 if self.stop\_requested:  
 self.update\_signal.emit("Обучение остановлено пользователем", "обучение")  
 return  
  
 self.complete\_signal.emit(True, "Обучение успешно завершено")  
  
 except Exception as e:  
 error\_msg = f"Ошибка обучения: {str(e)}"  
 self.update\_signal.emit(error\_msg, "ошибка")  
 raise  
  
 def generate\_text(self):  
 prompt = self.kwargs.get('prompt', "")  
 if not prompt:  
 return False, "Не указан запрос для генерации"  
  
 try:  
 gen\_params = {  
 'max\_length': self.kwargs.get('max\_length', 250),  
 'temperature': self.kwargs.get('temperature', 0.7),  
 'top\_p': 0.95,  
 'repetition\_penalty': 1.5,  
 'no\_repeat\_ngram\_size': 3,  
 'do\_sample': True,  
 'pad\_token\_id': self.tokenizer.eos\_token\_id,  
 'early\_stopping': True  
 }  
  
 input\_ids = self.tokenizer.encode(prompt, return\_tensors="pt")  
 output = self.model.generate(input\_ids, \*\*gen\_params)  
 generated = self.tokenizer.decode(output[0], skip\_special\_tokens=True)  
  
 cleaned = self.clean\_generated\_text(generated, prompt)  
 return True, cleaned  
  
 except Exception as e:  
 return False, f"Ошибка генерации: {str(e)}"  
  
 def clean\_generated\_text(self, text, prompt):  
 *"""  
 Удаляются лишние пробелы, символы табуляции и нестандартные символы, приводится к удобочитаемому виду.  
 """* if text.startswith(prompt):  
 text = text[len(prompt):].lstrip()  
 text = ' '.join(text.split())  
 text = re.sub(r'[\t•�]', '', text)  
 text = ' '.join(text.split())  
 text = re.sub(r'(\d+)[.)]\s\*', r'\1. ', text)  
 last\_period = max(text.rfind('.'), text.rfind('!'), text.rfind('?'))  
 if last\_period > 0:  
 text = text[:last\_period + 1]  
 return text.strip().capitalize()  
  
 def evaluate\_model(self):  
 if not self.model:  
 raise ValueError("Модель не загружена")  
  
 total\_params = sum(p.numel() for p in self.model.parameters())  
 trainable\_params = sum(p.numel() for p in self.model.parameters() if p.requires\_grad)  
  
 result = (  
 f"Информация о модели:\n"  
 f"Архитектура: GPT-2\n"  
 f"Всего параметров: {total\_params:,}\n"  
 f"Обучаемых параметров: {trainable\_params:,}\n"  
 f"Размер словаря: {len(self.tokenizer):,}\n"  
 )  
  
 if self.dataset:  
 result += f"\nИнформация о датасете:\n"  
 result += f"Обучающих примеров: {len(self.dataset['train']):,}\n"  
 if 'test' in self.dataset:  
 result += f"Тестовых примеров: {len(self.dataset['test']):,}\n"  
  
 sample = self.dataset['train'][0]  
 result += f"\nПример данных:\n"  
 for key, value in list(sample.items())[:5]:  
 val\_str = str(value)[:100] + "..." if len(str(value)) > 100 else str(value)  
 result += f"{key}: {val\_str}\n"  
  
 self.complete\_signal.emit(True, result)  
  
  
class MonitorWindow(QDockWidget):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_("Мониторинг системы")  
 self.setFeatures(QDockWidget.DockWidgetMovable | QDockWidget.DockWidgetFloatable)  
  
 widget = QWidget()  
 layout = QVBoxLayout(widget)  
  
 self.stats\_table = QTableWidget()  
 self.stats\_table.setColumnCount(2)  
 self.stats\_table.setHorizontalHeaderLabels(["Параметр", "Значение"])  
 self.stats\_table.setRowCount(4)  
  
 metrics = ["Загрузка CPU", "Исп. памяти", "Время", "Статус"]  
 for i, metric in enumerate(metrics):  
 self.stats\_table.setItem(i, 0, QTableWidgetItem(metric))  
 self.stats\_table.setItem(i, 1, QTableWidgetItem("N/A"))  
  
 self.stats\_table.resizeColumnsToContents()  
 layout.addWidget(self.stats\_table)  
  
 self.setWidget(widget)  
  
 def update\_stats(self, stats):  
 self.stats\_table.item(0, 1).setText(f"{stats['cpu']}%")  
 self.stats\_table.item(1, 1).setText(f"{stats['memory']}%")  
 self.stats\_table.item(2, 1).setText(stats['time'])  
 self.stats\_table.resizeColumnsToContents()  
  
  
class NLPApplication(QMainWindow):  
 *"""  
 Представляет собой главное окно приложения, связанное с GUI-компонентами и логикой работы с моделями.  
 Устанавливает начальные значения переменных, создает интерфейс и инициализирует монитор состояния системы.  
 """* def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.model = None  
 self.tokenizer = None  
 self.dataset = None  
 self.worker = None  
 self.monitor = None  
 self.monitor\_window = None  
  
 self.init\_ui()  
 self.setWindowTitle("Обучалка Моделей нейросети")  
 self.resize(1000, 800)  
 self.init\_monitor()  
 self.setup\_logging()  
  
 def save\_model(self):  
 if not self.model:  
 QMessageBox.warning(self, "Ошибка", "Нет модели для сохранения")  
 return  
  
 options = QFileDialog.Options()  
 save\_dir = QFileDialog.getExistingDirectory(self, "Выберите папку для сохранения модели", "", options=options)  
  
 if save\_dir:  
 try:  
 self.model.save\_pretrained(save\_dir)  
 self.tokenizer.save\_pretrained(save\_dir)  
 self.update\_status(f"Модель сохранена в {save\_dir}", "успех")  
 QMessageBox.information(self, "Успех", "Модель успешно сохранена")  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка", f"Не удалось сохранить модель: {str(e)}")  
  
 def setup\_logging(self):  
 *"""Настраивает логгер, подключая его к графическому интерфейсу для вывода сообщений в  
 специальном поле (QTextEdit) и записи логов в файл."""* for handler in logger.handlers[:]:  
 logger.removeHandler(handler)  
  
 log\_handler = LogHandler(self.training\_log)  
 logger.addHandler(log\_handler)  
  
 sys.stdout = TextEditStream(self.training\_log)  
 sys.stderr = TextEditStream(self.training\_log, is\_error=True)  
  
 def update\_training\_metrics(self, message, stage, metrics):  
 *"""  
 Обновляет таблицу показателей в процессе обучения модели,  
 отображая прогресс обучения в режиме реального времени.  
 """* if stage == "прогресс":  
 self.metrics\_table.item(0, 1).setText(metrics['epoch'])  
 self.metrics\_table.item(1, 1).setText(metrics['step'])  
 self.metrics\_table.item(2, 1).setText(metrics['loss'])  
 self.metrics\_table.item(3, 1).setText(metrics['learning\_rate'])  
 if 'speed' in metrics:  
 self.metrics\_table.item(4, 1).setText(metrics['speed'])  
 self.metrics\_table.resizeColumnsToContents()  
  
 def init\_monitor(self):  
 self.monitor\_window = MonitorWindow()  
 self.addDockWidget(Qt.RightDockWidgetArea, self.monitor\_window)  
  
 self.monitor = SystemMonitor()  
 self.monitor\_thread = QThread()  
 self.monitor.moveToThread(self.monitor\_thread)  
 self.monitor.update\_stats.connect(self.monitor\_window.update\_stats)  
  
 self.monitor\_thread.started.connect(self.monitor.run)  
 self.monitor\_thread.start()  
  
 def init\_ui(self):  
 central\_widget = QWidget()  
 self.setCentralWidget(central\_widget)  
 main\_layout = QVBoxLayout(central\_widget)  
  
 tabs = QTabWidget()  
 main\_layout.addWidget(tabs)  
  
 # Вкладка чата  
 chat\_tab = QWidget()  
 chat\_layout = QVBoxLayout(chat\_tab)  
  
 param\_group = QWidget()  
 param\_layout = QHBoxLayout(param\_group)  
  
 param\_layout.addWidget(QLabel("Длина ответа:"))  
 self.max\_len\_spin = QSpinBox()  
 self.max\_len\_spin.setRange(50, 1000)  
 self.max\_len\_spin.setValue(250)  
 param\_layout.addWidget(self.max\_len\_spin)  
  
 param\_layout.addWidget(QLabel("Температура:"))  
 self.temp\_spin = QDoubleSpinBox()  
 self.temp\_spin.setRange(0.1, 2.0)  
 self.temp\_spin.setValue(0.7)  
 self.temp\_spin.setSingleStep(0.1)  
 param\_layout.addWidget(self.temp\_spin)  
  
 chat\_layout.addWidget(param\_group)  
  
 self.chat\_output = QTextEdit()  
 self.chat\_output.setReadOnly(True)  
 chat\_layout.addWidget(self.chat\_output)  
  
 input\_group = QWidget()  
 input\_layout = QHBoxLayout(input\_group)  
  
 self.user\_input = QTextEdit()  
 self.user\_input.setMaximumHeight(80)  
 self.user\_input.setPlaceholderText("Введите ваш запрос (например: рецепт борща)...")  
 input\_layout.addWidget(self.user\_input)  
  
 self.send\_btn = QPushButton("Отправить")  
 self.send\_btn.clicked.connect(self.send\_message)  
 input\_layout.addWidget(self.send\_btn)  
  
 chat\_layout.addWidget(input\_group)  
  
 # Вкладка модели  
 model\_tab = QWidget()  
 model\_layout = QVBoxLayout(model\_tab)  
  
 model\_group = QWidget()  
 model\_group\_layout = QHBoxLayout(model\_group)  
  
 self.model\_path\_edit = QTextEdit()  
 self.model\_path\_edit.setMaximumHeight(50)  
 self.model\_path\_edit.setPlaceholderText("Путь к модели")  
 model\_group\_layout.addWidget(self.model\_path\_edit)  
  
 browse\_model\_btn = QPushButton("Обзор")  
 browse\_model\_btn.clicked.connect(self.browse\_model)  
 model\_group\_layout.addWidget(browse\_model\_btn)  
  
 load\_model\_btn = QPushButton("Загрузить модель")  
 load\_model\_btn.clicked.connect(self.load\_model)  
 model\_group\_layout.addWidget(load\_model\_btn)  
  
 model\_layout.addWidget(model\_group)  
  
 dataset\_group = QWidget()  
 dataset\_group\_layout = QVBoxLayout(dataset\_group)  
  
 path\_group = QWidget()  
 path\_layout = QHBoxLayout(path\_group)  
  
  
 self.dataset\_path\_edit = QTextEdit()  
 self.dataset\_path\_edit.setMaximumHeight(50)  
 self.dataset\_path\_edit.setPlaceholderText("Путь к датасету")  
 path\_layout.addWidget(self.dataset\_path\_edit)  
  
 browse\_dataset\_btn = QPushButton("Обзор")  
 browse\_dataset\_btn.clicked.connect(self.browse\_dataset)  
 path\_layout.addWidget(browse\_dataset\_btn)  
  
 load\_dataset\_btn = QPushButton("Загрузить датасет")  
 load\_dataset\_btn.clicked.connect(self.load\_dataset)  
 path\_layout.addWidget(load\_dataset\_btn)  
  
 dataset\_group\_layout.addWidget(path\_group)  
  
 filter\_group = QWidget()  
 filter\_layout = QVBoxLayout(filter\_group)  
  
 filter\_layout.addWidget(QLabel("Доступные темы:"))  
  
 self.topics\_list = QListWidget()  
 self.topics\_list.setSelectionMode(QListWidget.MultiSelection)  
 filter\_layout.addWidget(self.topics\_list)  
  
 btn\_group = QWidget()  
 btn\_layout = QHBoxLayout(btn\_group)  
  
 self.filter\_btn = QPushButton("Фильтровать по выбранным темам")  
 self.filter\_btn.clicked.connect(self.apply\_filter)  
 btn\_layout.addWidget(self.filter\_btn)  
  
 self.save\_filter\_btn = QPushButton("Сохранить фильтр")  
 self.save\_filter\_btn.clicked.connect(self.save\_filter)  
 btn\_layout.addWidget(self.save\_filter\_btn)  
  
 self.load\_filter\_btn = QPushButton("Загрузить фильтр")  
 self.load\_filter\_btn.clicked.connect(self.load\_filter)  
 btn\_layout.addWidget(self.load\_filter\_btn)  
  
 filter\_layout.addWidget(btn\_group)  
 dataset\_group\_layout.addWidget(filter\_group)  
  
 limit\_group = QWidget()  
 limit\_layout = QHBoxLayout(limit\_group)  
 limit\_layout.addWidget(QLabel("Лимит примеров для обучения:"))  
  
 self.dataset\_limit = QSpinBox()  
 self.dataset\_limit.setRange(0, 1000000)  
 self.dataset\_limit.setValue(0)  
 self.dataset\_limit.setSpecialValueText("Без ограничения")  
 limit\_layout.addWidget(self.dataset\_limit)  
  
 dataset\_group\_layout.addWidget(limit\_group)  
 model\_layout.addWidget(dataset\_group)  
  
 train\_group = QWidget()  
 train\_layout = QHBoxLayout(train\_group)  
  
 self.train\_btn = QPushButton("Обучить модель")  
 self.train\_btn.clicked.connect(self.train\_model)  
 train\_layout.addWidget(self.train\_btn)  
  
 self.stop\_train\_btn = QPushButton("Остановить обучение")  
 self.stop\_train\_btn.clicked.connect(self.stop\_training)  
 self.stop\_train\_btn.setEnabled(False)  
 train\_layout.addWidget(self.stop\_train\_btn)  
  
 save\_model\_btn = QPushButton("Сохранить модель")  
 save\_model\_btn.clicked.connect(self.save\_model)  
 train\_layout.addWidget(save\_model\_btn) # Добавить в layout с другими кнопками  
  
 model\_layout.addWidget(train\_group)  
  
 self.metrics\_table = QTableWidget()  
 self.metrics\_table.setColumnCount(2)  
 self.metrics\_table.setHorizontalHeaderLabels(["Метрика", "Значение"])  
 self.metrics\_table.setRowCount(5)  
 metrics = ["Эпоха", "Шаг", "Потери", "Скорость обучения", "Скорость"]  
 for i, metric in enumerate(metrics):  
 self.metrics\_table.setItem(i, 0, QTableWidgetItem(metric))  
 self.metrics\_table.setItem(i, 1, QTableWidgetItem("N/A"))  
 self.metrics\_table.resizeColumnsToContents()  
 model\_layout.addWidget(self.metrics\_table)  
  
 self.training\_log = QTextEdit()  
 self.training\_log.setReadOnly(True)  
 self.training\_log.setPlaceholderText("Логи приложения...")  
 model\_layout.addWidget(self.training\_log)  
  
 eval\_btn = QPushButton("Оценить модель")  
 eval\_btn.clicked.connect(self.evaluate\_model)  
 model\_layout.addWidget(eval\_btn)  
  
 # Вкладка информации  
 info\_tab = QWidget()  
 info\_layout = QVBoxLayout(info\_tab)  
  
 self.info\_text = QTextEdit()  
 self.info\_text.setReadOnly(True)  
 self.info\_text.setPlainText(  
 "Инструкция:\n"  
 "1. Загрузите модель (локально или из сети)\n"  
 "2. Загрузите датасет\n"  
 "3. При необходимости отфильтруйте по темам\n"  
 "4. Установите лимит примеров для обучения\n"  
 "5. Обучите модель\n"  
 "6. Используйте чат для генерации рецептов\n\n"  
 "Советы:\n"  
 "- Температура 0.7-1.0 для баланса креативности/точности\n"  
 "- Длина 200-400 токенов для подробных рецептов"  
 )  
 info\_layout.addWidget(self.info\_text)  
  
 tabs.addTab(chat\_tab, "Чат")  
 tabs.addTab(model\_tab, "Модель")  
 tabs.addTab(info\_tab, "Информация")  
  
 self.progress\_bar = QProgressBar()  
 self.progress\_bar.setVisible(False)  
 main\_layout.addWidget(self.progress\_bar)  
  
 self.status\_bar = QLabel("Готово")  
 main\_layout.addWidget(self.status\_bar)  
  
 def update\_status(self, message, stage="инфо"):  
 try:  
 self.status\_bar.setText(message)  
 safe\_message = str(message).replace('\n', ' ').replace('\r', '')  
  
 if stage == "ошибка":  
 self.training\_log.append(f"<font color='red'>{safe\_message}</font>")  
 elif stage == "обучение":  
 self.training\_log.append(f"<font color='blue'>{safe\_message}</font>")  
 elif stage == "подготовка":  
 self.training\_log.append(f"<font color='green'>{safe\_message}</font>")  
 else:  
 self.training\_log.append(safe\_message)  
  
 if self.monitor\_window:  
 self.monitor\_window.stats\_table.item(3, 1).setText(safe\_message[:50])  
  
 logger.info(f"[{stage}] {safe\_message}")  
 except Exception as e:  
 logger.error(f"Ошибка при обновлении статуса: {str(e)}")  
  
 def browse\_model(self):  
 options = QFileDialog.Options()  
 model\_dir = QFileDialog.getExistingDirectory(  
 self, "Выберите папку с моделью", "", options=options)  
  
 if model\_dir:  
 self.model\_path\_edit.setPlainText(model\_dir)  
  
 def browse\_dataset(self):  
 options = QFileDialog.Options()  
 dataset\_dir = QFileDialog.getExistingDirectory(  
 self, "Выберите папку с датасетом", "", options=options)  
  
 if dataset\_dir:  
 self.dataset\_path\_edit.setPlainText(dataset\_dir)  
  
 def load\_model(self):  
 model\_path = self.model\_path\_edit.toPlainText()  
  
 if not model\_path:  
 reply = QMessageBox.question(  
 self,  
 "Подтверждение",  
 "Загрузить модель из Hugging Face Hub? (Требуется интернет)",  
 QMessageBox.Yes | QMessageBox.No  
 )  
 if reply == QMessageBox.No:  
 return  
 model\_path = None  
  
 self.start\_worker('load\_model', model\_path=model\_path)  
  
 def load\_dataset(self):  
 dataset\_path = self.dataset\_path\_edit.toPlainText()  
 if not dataset\_path:  
 QMessageBox.warning(self, "Ошибка", "Укажите путь к датасету")  
 return  
  
 self.topics\_list.clear()  
 self.start\_worker('load\_dataset', dataset\_path=dataset\_path)  
  
 if self.worker:  
 self.worker.topics\_loaded.connect(self.update\_topics\_list)  
  
 def update\_topics\_list(self, topics, success):  
 self.topics\_list.clear()  
  
 if success and topics:  
 self.topics\_list.addItems(topics)  
 self.update\_status(f"Загружено {len(topics)} тем", "датасет")  
 self.filter\_btn.setEnabled(True)  
 else:  
 self.update\_status("Не удалось загрузить темы или датасет не содержит информации о темах", "предупреждение")  
 self.filter\_btn.setEnabled(False)  
  
 def apply\_filter(self):  
 selected\_items = self.topics\_list.selectedItems()  
 if not selected\_items:  
 QMessageBox.warning(self, "Ошибка", "Выберите хотя бы одну тему")  
 return  
  
 selected\_topics = [item.text() for item in selected\_items]  
 self.start\_worker('filter\_dataset', selected\_topics=selected\_topics)  
  
 def save\_filter(self):  
 selected\_items = self.topics\_list.selectedItems()  
 if not selected\_items:  
 QMessageBox.warning(self, "Ошибка", "Нет выбранных тем для сохранения")  
 return  
  
 selected\_topics = [item.text() for item in selected\_items]  
  
 options = QFileDialog.Options()  
 path, \_ = QFileDialog.getSaveFileName(  
 self, "Сохранить фильтр", "", "JSON Files (\*.json)", options=options)  
  
 if path:  
 try:  
 with open(path, 'w', encoding='utf-8') as f:  
 json.dump(selected\_topics, f, ensure\_ascii=False, indent=2)  
 self.update\_status(f"Фильтр сохранен в {path}", "инфо")  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка", f"Не удалось сохранить фильтр: {str(e)}")  
  
 def load\_filter(self):  
 options = QFileDialog.Options()  
 path, \_ = QFileDialog.getOpenFileName(  
 self, "Загрузить фильтр", "", "JSON Files (\*.json)", options=options)  
  
 if path:  
 try:  
 with open(path, 'r', encoding='utf-8') as f:  
 selected\_topics = json.load(f)  
  
 for i in range(self.topics\_list.count()):  
 self.topics\_list.item(i).setSelected(False)  
  
 for topic in selected\_topics:  
 items = self.topics\_list.findItems(topic, Qt.MatchExactly)  
 if items:  
 items[0].setSelected(True)  
  
 self.update\_status(f"Загружен фильтр: {', '.join(selected\_topics)}", "инфо")  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка", f"Не удалось загрузить фильтр: {str(e)}")  
  
 def train\_model(self):  
 if not self.model:  
 QMessageBox.warning(self, "Ошибка", "Сначала загрузите модель")  
 return  
  
 self.training\_log.clear()  
 self.train\_btn.setEnabled(False)  
 self.stop\_train\_btn.setEnabled(True)  
  
 max\_samples = self.dataset\_limit.value()  
 if max\_samples <= 0:  
 max\_samples = None  
  
 self.start\_worker('train', max\_samples=max\_samples)  
  
 def stop\_training(self):  
 if self.worker and self.worker.isRunning():  
 self.worker.request\_stop()  
 self.update\_status("Останавливаем обучение...", "предупреждение")  
 self.train\_btn.setEnabled(True)  
 self.stop\_train\_btn.setEnabled(False)  
  
 def send\_message(self):  
 *"""Реализует механизм отправки пользовательского запроса и последующей генерации ответа от модели."""* prompt = self.user\_input.toPlainText().strip()  
 if not prompt:  
 QMessageBox.warning(self, "Ошибка", "Введите текст запроса")  
 return  
  
 if not self.model:  
 QMessageBox.warning(self, "Ошибка", "Сначала загрузите модель")  
 return  
  
 self.chat\_output.append(f"<b>Вы:</b> {prompt}")  
 self.user\_input.clear()  
  
 gen\_params = {  
 'prompt': prompt,  
 'max\_length': self.max\_len\_spin.value(),  
 'temperature': self.temp\_spin.value()  
 }  
  
 self.start\_worker('generate', \*\*gen\_params)  
  
 def evaluate\_model(self):  
 if not self.model:  
 QMessageBox.warning(self, "Ошибка", "Сначала загрузите модель")  
 return  
  
 self.start\_worker('evaluate')  
  
 def start\_worker(self, operation, \*\*kwargs):  
 *"""Запускает фоновый поток для выполнения различных операций (загрузка модели,  
 обучение, генерация текста и оценка)."""* if self.worker and self.worker.isRunning():  
 QMessageBox.warning(self, "Ошибка", "Дождитесь завершения текущей операции")  
 return  
  
 self.progress\_bar.setVisible(True)  
 self.progress\_bar.setRange(0, 0)  
  
 self.worker = ModelWorker(operation, \*\*kwargs)  
 self.worker.update\_signal.connect(self.update\_status)  
 self.worker.complete\_signal.connect(self.operation\_complete)  
 self.callback = TrainingProgressCallback()  
 self.worker.update\_progress.connect(self.update\_training\_metrics)  
 kwargs['callback'] = self.callback  
 self.worker.kwargs = kwargs  
  
 self.worker.model = self.model  
 self.worker.tokenizer = self.tokenizer  
 self.worker.dataset = self.dataset  
  
 self.worker.start()  
  
 def operation\_complete(self, success, message):  
 *"""Обрабатывает сигнал завершения какой-либо операции (успешное завершение или ошибку)."""* self.progress\_bar.setVisible(False)  
 self.train\_btn.setEnabled(True)  
 self.stop\_train\_btn.setEnabled(False)  
  
 if success:  
 if self.worker.model:  
 self.model = self.worker.model  
 if self.worker.operation == 'train':  
 save\_path = "./saved\_model"  
 self.model.save\_pretrained(save\_path)  
 self.tokenizer.save\_pretrained(save\_path)  
 self.update\_status(f"Модель сохранена в {save\_path}", "успех")  
 if self.worker.tokenizer:  
 self.tokenizer = self.worker.tokenizer  
 if self.worker.dataset:  
 self.dataset = self.worker.dataset  
  
 if self.worker.operation == 'generate':  
 self.chat\_output.append(f"<b>Бот:</b> {message}")  
 elif self.worker.operation == 'evaluate':  
 QMessageBox.information(self, "Оценка модели", message)  
  
 self.update\_status("Операция успешно завершена", "успех")  
 else:  
 QMessageBox.critical(self, "Ошибка", message)  
  
 def closeEvent(self, event):  
 if self.worker and self.worker.isRunning():  
 self.worker.request\_stop()  
 self.worker.wait(2000)  
  
 if self.monitor:  
 self.monitor.running = False  
 self.monitor\_thread.quit()  
 self.monitor\_thread.wait()  
  
 event.accept()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 try:  
 import transformers  
 import datasets  
 import psutil  
 except ImportError as e:  
 print(f"Ошибка: Не установлены требуемые библиотеки ({str(e)})")  
 print("Установите: pip install transformers datasets psutil")  
 sys.exit(1)  
  
 app = QApplication(sys.argv)  
  
 torch.set\_num\_threads(psutil.cpu\_count(logical=False))  
 os.environ['CUDA\_VISIBLE\_DEVICES'] = ''  
  
 window = NLPApplication()  
 window.show()  
 sys.exit(app.exec\_())

**Приложение 6.**

**Руководство пользователя**

* **Инструкция по использованию приложения**
* **Шаг 1: Установка и запуск приложения**
* Скачайте архив с приложением.
* Распакуйте архив в любую удобную директорию.
* Откройте терминал или командную строку и перейдите в распакованную директорию.
* Запустите приложение командой:
* python chat\_bot.py
* или двойным кликом по исполняемому файлу, если приложение упаковано в исполняемый файл.
* **Шаг 2: Интерфейс приложения**
* После запуска откроется основное окно приложения, которое имеет следующую структуру:
* **Вкладка Чат**: Здесь осуществляется общение с ботом и генерация ответов.
* **Вкладка Модель**: Управление моделью и набором данных.
* **Вкладка Информация**: Справочная информация о программе и рекомендации по использованию.
* **Шаг 3: Использование вкладки "Чат"**
* Напишите запрос в поле ввода ("Введите ваш запрос").
* Нажмите кнопку "Отправить".
* Получите ответ от бота в области вывода ("Бот:").
* Примеры запросов:
* "Расскажите анекдот."
* "Какой рецепт пиццы?"
* "Что посмотреть вечером?"
* **Шаг 4: Управление моделью (вкладка "Модель")**
* **4.1 Загрузка модели**
* Выберите директорию с моделью, нажав кнопку "Обзор".
* Нажмите кнопку "Загрузить модель".
* Если модель не указана, программа предложит скачать стандартную модель из интернета.
* **4.2 Загрузка набора данных**
* Выберите директорию с вашим набором данных.
* Нажмите кнопку "Загрузить датасет".
* Программа проанализирует набор данных и покажет доступные категории (темы).
* **4.3 Фильтрация данных**
* Если в вашем наборе данных имеются темы, выберите интересующие вас темы, отметив их в списке слева. Нажмите кнопку "Фильтровать по выбранным темам".
* **4.4 Обучение модели**
* Нажмите кнопку "Обучить модель". Начнется процесс обучения, который займет некоторое время в зависимости от размера вашего набора данных и возможностей компьютера.
* Можно ограничить число примеров для обучения, установив значение в поле "Лимит примеров для обучения".
* **4.5 Оценка модели**
* По завершении обучения нажмите кнопку "Оценить модель", чтобы увидеть отчет о количестве параметров и характеристиках модели.
* **Шаг 5: Рекомендуемые настройки**
* При генерации текста рекомендуется установить следующие параметры:
* **Максимальная длина ответа**: около 200–400 токенов для длинных и информативных ответов.
* **Температура**: 0.7–1.0 для балансировки креативности и согласованности ответа.
* **Шаг 6: Экспорт модели**
* После успешного обучения модель можно экспортировать, нажав кнопку "Сохранить модель". Будут созданы каталоги с весами и токенайзером модели, которые можно перенести на другой компьютер или использовать отдельно.
* **Советы по улучшению опыта использования**
* Регулярно проверяйте логи, чтобы отслеживать возможные ошибки и предупреждающие сообщения.
* Экспериментируйте с разными температурами и длинами ответов для достижения наилучших результатов.
* При недостаточной производительности уменьшите размер пакета данных или попробуйте упростить модель.

**Приложение 7.**

**Тестовые данные**

Для приложение использовались набор датасетов ZeroAgency/ru-big-russian-dataset скачанные локально.

Структура каталогов датасета выглядит следующим образом:

datasets/

├── train/

│ ├── dataset\_info.json

│ ├── state.json

│ ├── data-00000-of-00020.arrow

│ ├── data-00001-of-00020.arrow

│ └── ...

├── test/

│ ├── dataset\_info.json

│ ├── state.json

│ ├── data-00000-of-00001.arrow

│ └── ...

└── dataset\_dict.json

**Приложение 7**

Лог программы:

Начало подготовки данных...

2025-06-12 00:15:49,390 - INFO - [подготовка] Начало подготовки данных...

Пример данных: {'conversation': [{'role': 'user', 'content': 'Используя указанные кул...

2025-06-12 00:15:49,391 - INFO - [подготовка] Пример данных: {'conversation': [{'role': 'user', 'content': 'Используя указанные кул...

Датасет ограничен до 500 примеров

2025-06-12 00:15:49,405 - INFO - [подготовка] Датасет ограничен до 500 примеров

Map: 0%| | 0/500 [00:00

Токенизировано 0 примеров...

2025-06-12 00:15:49,427 - INFO - [подготовка] Токенизировано 0 примеров...

Map: 3%|3 | 16/500 [00:00

Map: 10%|9 | 48/500 [00:00

Map: 16%|#6 | 80/500 [00:00

Map: 22%|##2 | 112/500 [00:00

Map: 29%|##8 | 144/500 [00:00

Map: 35%|###5 | 176/500 [00:00

Map: 42%|####1 | 208/500 [00:01

Map: 48%|####8 | 240/500 [00:01

Map: 54%|#####4 | 272/500 [00:01

Map: 61%|###### | 304/500 [00:01

Map: 67%|######7 | 336/500 [00:01

Map: 74%|#######3 | 368/500 [00:01

Map: 80%|######## | 400/500 [00:01

Токенизировано 400 примеров...

2025-06-12 00:15:51,254 - INFO - [подготовка] Токенизировано 400 примеров...

Map: 86%|########6 | 432/500 [00:01

Map: 93%|#########2| 464/500 [00:02

Map: 99%|#########9| 496/500 [00:02

Map: 100%|##########| 500/500 [00:02

Токенизация завершена. Примеров: 500

2025-06-12 00:15:51,628 - INFO - [подготовка] Токенизация завершена. Примеров: 500

Начало обучения модели...

2025-06-12 00:15:51,721 - INFO - [обучение] Начало обучения модели...

0%| | 0/189 [00:00

1%| | 1/189 [00:07

1%|1 | 2/189 [00:14

2%|1 | 3/189 [00:21

2%|2 | 4/189 [00:28

3%|2 | 5/189 [00:35

3%|3 | 6/189 [00:41

4%|3 | 7/189 [00:48

4%|4 | 8/189 [00:55

5%|4 | 9/189 [01:02

5%|5 | 10/189 [01:09

6%|5 | 11/189 [01:16

6%|6 | 12/189 [01:23

7%|6 | 13/189 [01:30

7%|7 | 14/189 [01:37

8%|7 | 15/189 [01:44

8%|8 | 16/189 [01:52

9%|8 | 17/189 [01:59

10%|9 | 18/189 [02:06

10%|# | 19/189 [02:13

11%|# | 20/189 [02:21

{'loss': 2.7554, 'grad\_norm': 3.0555243492126465, 'learning\_rate': 4.4973544973544974e-05, 'epoch': 0.32}

11%|# | 20/189 [02:21

11%|#1 | 21/189 [02:28

12%|#1 | 22/189 [02:35

12%|#2 | 23/189 [02:42

13%|#2 | 24/189 [02:50

13%|#3 | 25/189 [02:57

14%|#3 | 26/189 [03:04

14%|#4 | 27/189 [03:12

15%|#4 | 28/189 [03:19

15%|#5 | 29/189 [03:27

16%|#5 | 30/189 [03:34

16%|#6 | 31/189 [03:41

17%|#6 | 32/189 [03:49

17%|#7 | 33/189 [03:56

18%|#7 | 34/189 [04:03

19%|#8 | 35/189 [04:11

19%|#9 | 36/189 [04:18

20%|#9 | 37/189 [04:26

20%|## | 38/189 [04:33

21%|## | 39/189 [04:40

21%|##1 | 40/189 [04:48

{'loss': 2.4301, 'grad\_norm': 3.1625823974609375, 'learning\_rate': 3.968253968253968e-05, 'epoch': 0.64}

21%|##1 | 40/189 [04:48

22%|##1 | 41/189 [04:55

22%|##2 | 42/189 [05:02

23%|##2 | 43/189 [05:10

23%|##3 | 44/189 [05:17

24%|##3 | 45/189 [05:24

24%|##4 | 46/189 [05:32

25%|##4 | 47/189 [05:39

25%|##5 | 48/189 [05:46

26%|##5 | 49/189 [05:54

26%|##6 | 50/189 [06:01

27%|##6 | 51/189 [06:09

28%|##7 | 52/189 [06:16

28%|##8 | 53/189 [06:23

29%|##8 | 54/189 [06:31

29%|##9 | 55/189 [06:38

30%|##9 | 56/189 [06:46

30%|### | 57/189 [06:53

31%|### | 58/189 [07:00

31%|###1 | 59/189 [07:08

32%|###1 | 60/189 [07:16

{'loss': 2.5182, 'grad\_norm': 2.5027153491973877, 'learning\_rate': 3.439153439153439e-05, 'epoch': 0.96}

32%|###1 | 60/189 [07:16

32%|###2 | 61/189 [07:24

33%|###2 | 62/189 [07:38

33%|###3 | 63/189 [07:47

34%|###3 | 64/189 [08:03

34%|###4 | 65/189 [08:20

35%|###4 | 66/189 [08:36

35%|###5 | 67/189 [08:53

36%|###5 | 68/189 [09:09

37%|###6 | 69/189 [09:27

37%|###7 | 70/189 [09:43

38%|###7 | 71/189 [10:00

38%|###8 | 72/189 [10:16

39%|###8 | 73/189 [10:33

39%|###9 | 74/189 [10:49

40%|###9 | 75/189 [11:05

40%|#### | 76/189 [11:20

41%|#### | 77/189 [11:38

41%|####1 | 78/189 [11:57

42%|####1 | 79/189 [12:15

42%|####2 | 80/189 [12:34

{'loss': 2.2522, 'grad\_norm': 2.5431673526763916, 'learning\_rate': 2.91005291005291e-05, 'epoch': 1.27}

42%|####2 | 80/189 [12:34

43%|####2 | 81/189 [12:52

43%|####3 | 82/189 [13:11

44%|####3 | 83/189 [13:30

44%|####4 | 84/189 [13:50

45%|####4 | 85/189 [14:09

46%|####5 | 86/189 [14:28

46%|####6 | 87/189 [14:47

47%|####6 | 88/189 [15:05

47%|####7 | 89/189 [15:23

48%|####7 | 90/189 [15:42

48%|####8 | 91/189 [16:01

49%|####8 | 92/189 [16:20

49%|####9 | 93/189 [16:39

50%|####9 | 94/189 [16:59

50%|##### | 95/189 [17:17

51%|##### | 96/189 [17:36

51%|#####1 | 97/189 [17:55

52%|#####1 | 98/189 [18:15

52%|#####2 | 99/189 [18:33

53%|#####2 | 100/189 [18:53

{'loss': 2.1791, 'grad\_norm': 2.2003307342529297, 'learning\_rate': 2.380952380952381e-05, 'epoch': 1.59}

53%|#####2 | 100/189 [18:53

53%|#####3 | 101/189 [19:12

54%|#####3 | 102/189 [19:31

54%|#####4 | 103/189 [19:51

55%|#####5 | 104/189 [20:09

56%|#####5 | 105/189 [20:28

56%|#####6 | 106/189 [20:46

57%|#####6 | 107/189 [21:05

57%|#####7 | 108/189 [21:24

58%|#####7 | 109/189 [21:42

58%|#####8 | 110/189 [21:49

59%|#####8 | 111/189 [21:57

59%|#####9 | 112/189 [22:06

60%|#####9 | 113/189 [22:14

60%|###### | 114/189 [22:22

61%|###### | 115/189 [22:31

61%|######1 | 116/189 [22:48

62%|######1 | 117/189 [23:08

62%|######2 | 118/189 [23:27

63%|######2 | 119/189 [23:47

63%|######3 | 120/189 [24:06

{'loss': 2.1399, 'grad\_norm': 2.537520170211792, 'learning\_rate': 1.8518518518518518e-05, 'epoch': 1.91}

63%|######3 | 120/189 [24:06

64%|######4 | 121/189 [24:26

65%|######4 | 122/189 [24:45

65%|######5 | 123/189 [25:04

66%|######5 | 124/189 [25:23

66%|######6 | 125/189 [25:42

67%|######6 | 126/189 [25:52

67%|######7 | 127/189 [26:11

68%|######7 | 128/189 [26:31

68%|######8 | 129/189 [26:50

69%|######8 | 130/189 [27:09

69%|######9 | 131/189 [27:28

70%|######9 | 132/189 [27:47

70%|####### | 133/189 [28:07

71%|####### | 134/189 [28:25

71%|#######1 | 135/189 [28:44

72%|#######1 | 136/189 [29:02

72%|#######2 | 137/189 [29:21

73%|#######3 | 138/189 [29:41

74%|#######3 | 139/189 [29:58

74%|#######4 | 140/189 [30:06

{'loss': 2.0634, 'grad\_norm': 2.6667797565460205, 'learning\_rate': 1.3227513227513228e-05, 'epoch': 2.22}

74%|#######4 | 140/189 [30:06

75%|#######4 | 141/189 [30:15

75%|#######5 | 142/189 [30:23

76%|#######5 | 143/189 [30:31

76%|#######6 | 144/189 [30:40

77%|#######6 | 145/189 [30:48

77%|#######7 | 146/189 [30:56

78%|#######7 | 147/189 [31:04

78%|#######8 | 148/189 [31:13

79%|#######8 | 149/189 [31:21

79%|#######9 | 150/189 [31:29

80%|#######9 | 151/189 [31:37

80%|######## | 152/189 [31:47

81%|######## | 153/189 [31:56

81%|########1 | 154/189 [32:05

82%|########2 | 155/189 [32:13

83%|########2 | 156/189 [32:22

83%|########3 | 157/189 [32:30

84%|########3 | 158/189 [32:38

84%|########4 | 159/189 [32:47

85%|########4 | 160/189 [32:58

{'loss': 2.0119, 'grad\_norm': 2.899214506149292, 'learning\_rate': 7.936507936507936e-06, 'epoch': 2.54}

85%|########4 | 160/189 [32:58

85%|########5 | 161/189 [33:17

86%|########5 | 162/189 [33:36

86%|########6 | 163/189 [33:56

87%|########6 | 164/189 [34:15

87%|########7 | 165/189 [34:34

88%|########7 | 166/189 [34:53

88%|########8 | 167/189 [35:12

89%|########8 | 168/189 [35:32

89%|########9 | 169/189 [35:51

90%|########9 | 170/189 [36:08

90%|######### | 171/189 [36:16

91%|#########1| 172/189 [36:24

92%|#########1| 173/189 [36:33

92%|#########2| 174/189 [36:41

93%|#########2| 175/189 [36:49

93%|#########3| 176/189 [36:58

94%|#########3| 177/189 [37:06

94%|#########4| 178/189 [37:14

95%|#########4| 179/189 [37:22

95%|#########5| 180/189 [37:31

{'loss': 1.9247, 'grad\_norm': 3.253157138824463, 'learning\_rate': 2.6455026455026455e-06, 'epoch': 2.86}

95%|#########5| 180/189 [37:31

96%|#########5| 181/189 [37:39

96%|#########6| 182/189 [37:48

97%|#########6| 183/189 [37:57

97%|#########7| 184/189 [38:05

98%|#########7| 185/189 [38:14

98%|#########8| 186/189 [38:23

99%|#########8| 187/189 [38:31

99%|#########9| 188/189 [38:40

100%|##########| 189/189 [38:44

{'train\_runtime': 2327.9094, 'train\_samples\_per\_second': 0.644, 'train\_steps\_per\_second': 0.081, 'train\_loss': 2.2448087722536116, 'epoch': 3.0}

100%|##########| 189/189 [38:47

100%|##########| 189/189 [38:47

Модель сохранена в ./saved\_model

2025-06-12 00:54:41,523 - INFO - [успех] Модель сохранена в ./saved\_model

Операция успешно завершена

2025-06-12 00:54:41,525 - INFO - [успех] Операция успешно завершена

Операция успешно завершена

2025-06-12 00:56:34,644 - INFO - [успех] Операция успешно завершена