МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)»

Институт № 8 «Компьютерные науки и прикладная математика» **Кафедра 805**
Направление подготовки 01.03.04 «Прикладная математика» **Группа М8О-403Б-18**
Профиль Математическое и программное обеспечение систем обработки информации и управления
Квалификация (степень) бакалавр

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

На тему: Построение системы генерации стилизованных текстов с использованием алгоритмов искусственного интеллекта и нейронных сетей.

Автор ВКРБ Ларькин Владимир Дмитриевич (фамилия, имя, отчество) (_____)
Научный руководитель Пановский Валентин Николаевич (фамилия, имя, отчество) (_____)

К защите допустить

Заведующий кафедрой № 805 Пантелеев Андрей Владимирович (фамилия, имя, отчество) (_____)
« 24 » мая 2022 г.

Москва 2022

РЕФЕРАТ

Отчёт содержит 27 стр., 8 рис., 1 табл., 5 источн., 1 прил.

ГЕНЕРАЦИЯ ТЕКСТА, НЕЙРОННЫЕ СЕТИ, NLP, TRANSFORMER, ДООБУЧЕНИЕ, RUGPT-3

В работе представлено решение задачи дообучения нейросетевой языковой модели ruGPT-3 Small архитектуры Transformer на сравнительно небольшом корпусе текстов, принадлежащих конкретной предметной области, для усвоения моделью стилистики текстов данной области и последующего её использования для генерации новых текстов.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире с ростом уровня образования и увеличением спроса на специалистов высокой квалификации для оптимизации производственных процессов требуется повышать уровень автоматизации предприятий, чтобы разгрузить работников и предоставить им возможность заниматься интеллектуальным трудом. А с увеличением вычислительных мощностей и развитием компьютерных наук всё больше рутинных задач поддаются автоматизации. Задача автоматизации написания разного рода текстов стоит особенно остро, так как она актуальна в самых разных сферах человеческой деятельности.

В настоящей работе предлагается система, которая позволяет автоматизировать большую часть действий, требуемых для создания базовой структуры документа и его частичного заполнения, предоставляя возможность коррекции и дополнения полученного текста пользователем «на лету».

Работа системы демонстрируется применительно к задаче генерации сценариев юмористических телешоу. Актуальность решения этой задачи обусловлена его применимостью в сферах психологии и психиатрии для тестирования уровня эмпатии способом, близким к методике А. Меграбяна и Н. Эпштейна [1]: пациенту предлагается прочитать несколько текстов и ответить на ряд вопросов, касающихся испытываемых им чувств, после чего на основании полученных ответов делается вывод по поводу уровня его эмпатии.

Но возможности данной системы не ограничены только этой предметной областью. Предлагаемое решение предоставляет гибкий механизм дообучения под генерацию текстов из той предметной области, которой принадлежит обучающая выборка текстов.

ОСНОВНАЯ ЧАСТЬ

1. Теоретическая часть

1.1 Определения

Корпус текстов — множество подобранных и определённым образом обработанных текстов.

Токен — элементарная единица разбиения корпуса.

Токенизация — процесс разбиения корпуса на токены с присвоением им уникальных числовых идентификаторов.

Языковая модель — распределение $P(w_t|w_1, w_2, w_3, \dots, w_n)$ вероятностей встретить токен w_t в корпусе сразу после n токенов w_i, $i \in [1, n]$, идущих подряд, где $w_i \in W \; \forall i$, W — множество всех токенов корпуса, n — длина контекста модели.

Длина контекста — количество n токенов w_i, $i \in [0, n]$, предшествующих токену w_t, от которых зависит вероятность появления в тексте токена w_t.

Дообучение — процесс обучения уже обученной на некоторых данных модели машинного обучения на новых данных. В случае языковой модели это означает подстройку модели под новое распределение токенов.

Перплексия — мера схожести двух вероятностных распределений, используемая для оценки качества генерации текста языковой моделью. Перплексия задаётся формулой 1.1.

$$\text{PP}(W) = \sqrt[n]{\frac{1}{P(w_1, w_2, \dots, w_n)}} \tag{1.1}$$

1.2 Постановка задачи

Заданы:

— корпус, состоящий из текстов, принадлежащих конкретной предметной области,

— предобученная нейросетевая языковая модель, хорошо моделирующая вероятностное распределение слов в естественном языке.

Требуется: дообучить данную языковую модель на данных из корпуса, получив новую языковую модель, моделирующую распределение вероятностей слов в данном корпусе, и применить её для генерации новых текстов, принадлежащих предметной области данного корпуса, встроив в графический пользовательский интерфейс.

2. Практическая часть

2.1 Процесс работы с системой

Процесс взаимодействия с программой выглядит следующим образом:

а) обученная на десятках гигабайт текста и хорошо моделирующая распределение слов в русском языке нейронная сеть дообучается на требуемом наборе данных, подстраиваясь под требуемую предметную область,

б) в дообученную языковую модель подаётся затравка — начало текста, которое модели необходимо продолжить,

в) пользователь оценивает результат генерации и может вручную отредактировать или отменить его,

г) процесс повторяется, начиная с пункта б, но в качестве затравки теперь выступает результат коррекции из пункта в.

Более наглядно процесс работы продемонстрирован на рисунке 2.1.

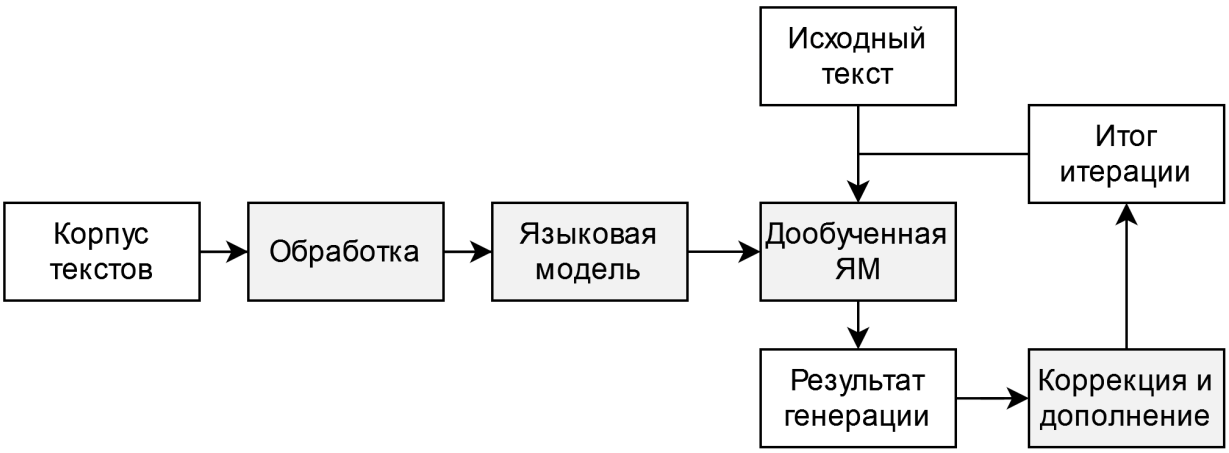

Рисунок 2.1 — Высокоуровневая схема системы

2.2 Выбор языковой модели

Среди доступных под свободными лицензиями языковых моделей рассматривались те, что приведены в таблице 2.1. Они построены на основе архитектуры Transformer, показывающей лучшие на данный момент результаты в задаче генерации текста [2], а длина контекста, который они учитывают,

достаточно большая, что важно для поддержания связности повествования в сгенерированном тексте. Названия моделей и их характеристики приведены в таблице 2.1.

Таблица 2.1 — Некоторые нейросетевые языковые модели под свободными лицензиями

Семейство модели	Название модели	Число параметров	Длина контекста
Russian GPT-3	ruGPT-3 Small	117 млн.	2048
	ruGPT-3 Large	760 млн.	2048
	ruGPT-3 XL	1,3 млрд.	2048
GPT-2	GPT-2 Small	124 млн.	1024
	GPT-2 Medium	355 млн.	1024
	GPT-2 XL	1,5 млрд.	1024

Из рассмотренных вариантов лучшие всего подошла модель ruGPT-3 Small от «Сбера» [3]. Её преимуществом является то, что она обучалась на русскоязычном корпусе и заточена под генерацию текста, в первую очередь, на русском языке, а малое относительно других моделей семейства ruGPT-3 число параметров позволяет дообучать её, располагая сравнительно небольшими мощностями.

2.3 Обработка текста

Перед обучением нейронной сети требуется привести данные к особому виду. В данном случае предварительная обработка корпуса заключается в выделении структурных блоков текста специальными синтаксическими конструкциями на естественном языке, формат которых определён заранее и сохраняется неизменным во всём корпусе. Выбор естественного языка обусловлен тем, что обученной на корпусе текстов на естественном языке языковой модели в этом случае не понадобится много данных для подстройки под новый синтаксис.

Код преобразования данных к нужному виду представлен в листинге А.2.

Пример обработанных данных показан в листинге 1.

```
Место действия -- ПАВ. лобби/лифт.
Время действия -- день. день 1.
Действующие лица -- элеонора, Управляющий, массовка.
Ремарка -- Лифт открывается. Элеонора в лифте с букетом в руках,
дочитывает записку. Смена плана. Перед лифтом стоит Управляющий.
Управляющий говорит: «Доброе утро, Элеонора Андреевна! Красивые
цветы!»
Элеонора говорит: «Спасибо, я и сама заметила. Ты что-то хотел?»
Управляющий говорит: «Да. Лифт»
Ремарка -- Элеонора выходит из лифта. Управляющий, проводив её
взглядом, входит. зк
Макс говорит: «И вот, спустя пару недель, она явно испытывает
симпатию. Но пока к нему – незнакомцу, а не к тебе»
Ремарка -- Лифт закрывается.
```

Листинг 1 — Пример обработанного текста

2.4 Разбиение корпуса

При обучении данные разбиваются на части, способные поместиться в видеопамять, следовательно, важно производить разбиение определённым образом для лучшего результата. Все тексты обучающей выборки разделяются на части такого размера, чтобы:

а) в токенизированном виде их длина была не меньше длины контекста модели, чтобы при генерации учитывалось максимально возможное количество информации,

б) их длина была не слишком большой, чтобы при обучении иметь возможность подавать их в модель в случайном порядке для более эффективной оптимизации,

в) каждая часть была самостоятельным текстом, принадлежащим исходной предметной области.

Для подачи разбитых данных в модель был написан собственный класс **TextsDataset**, представленный в листинге 2..1. Он представляет собой коллекцию, которая при инициализации загружает с диска данные в виде длинных текстовых файлов, разбивает их вышеописанным способом и предоставляет интерфейс для доступа к получившимся коротким фрагментам.

Листинг 2..1 — Класс датасета, хранящий данные в разбитом виде

```
class TextsDataset(Dataset):
    """Texts one by one"""

    def __init__(self, tokenizer: PreTrainedTokenizer, path: str,
        block_size=2048):
        assert os.path.isdir(path)

        block_size = block_size - (tokenizer.max_len -
            tokenizer.max_len_single_sentence)

        logger.info("Creating features from dataset file at %s", path)

        self.examples = []
        try:
            for file in os.listdir(path):
                file_path = os.path.join(path, file)
                with open(file_path, encoding="utf-8") as f:
                    text = f.read()

                tokenized_text =
                    tokenizer.convert_tokens_to_ids(tokenizer.tokenize(text))

                logger.info(f"Tokenized {file_path}: tokens len:
                    {len(tokenized_text)}")

                for i in range(0, len(tokenized_text) - block_size + 1,
                    block_size):  # Truncate in block of block_size
                    self.examples.append(
                        tokenizer.build_inputs_with_special_tokens(
                            tokenized_text[i: i + block_size]
                        )
                    )
        except Exception as e:
            logger.exception(e)

        logger.info(f"Created dataset of size {len(self.examples)}")

    def __len__(self):
```

```
34          return len(self.examples)
35
36      def __getitem__(self, item):
37          return torch.tensor(self.examples[item], dtype=torch.long)
```

2.5 Процесс обучения

Исходная модель ruGPT-3 Small была загружена из библиотеки Transformers для языка Python. Обучение производилось с помощью оригинального программного кода от «Сбера» [4], в котором был изменён механизм подачи данных в модель так, чтобы это происходило с использованием собственного класса **TextsDataset** из листинга 2..1.

Обучение происходило с помошью метода оптимизации Adam с шагом градиентного спуска $5 \cdot 10^{-5}$ в течение 100 эпох. В качестве набора данных были взяты сценарии юмористических телешоу. Итоговое значение перплексии составило 10,7.

2.6 Пользовательский интерфейс

Для создания графического интерфейса пользователя был использован фреймворк Streamlit. Он позволяет средствами языка Python создавать веб-приложения, которые открываются прямо в браузере на любой операционной системе [5].

Пользователь взаимодействует с программой через следующие элементы управления:

— поле ввода текста,

— кнопка «Дополнить»,

— кнопка «Отменить»,

— кнопка «Скачать результат».

Поле ввода текста позволяет вводить затравку для генерации, в нём же появляется сгенерированное дополнение, которое сразу можно отредак-

тировать. Кнопка «Дополнить» запускает генерацию продолжения текста, находящегося в поле ввода; кнопка «Отменить» отменяет результат одной генерации, пользователь может отменять их сколько угодно вплоть до самого начала; кнопка «Скачать результат» нужна, чтобы загрузить на компьютер текстовый файл, содержащий весь текст, находящийся в поле ввода.

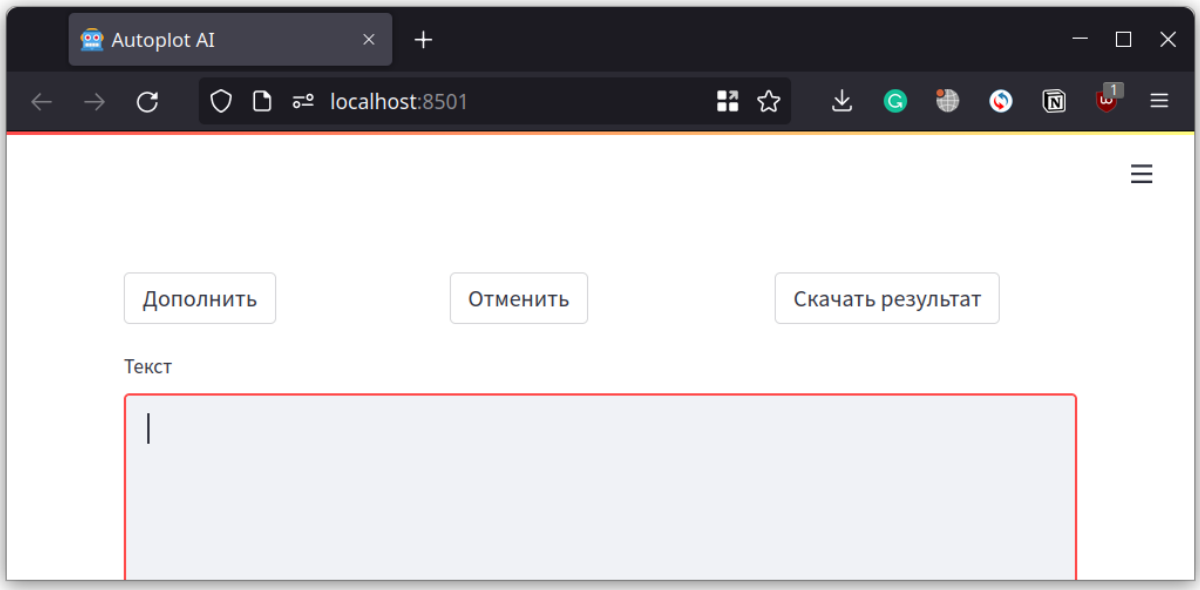

Рисунок 2.2 — Графический пользовательский интерфейс

2.7 Процесс работы с приложением

Рассмотрим сценарий работы с приложением:

а) пользователь вводит затравку (рисунок 2.3),

б) модель её продолжает (рисунок 2.4),

в) пользователь вводит дополнительную информацию, чтобы напрвить ход повествования (рисунок 2.5),

г) модель продолжает текст (рисунок 2.6),

д) пользователь остаётся неудовлетворён результатом, исправляет его и добавляет новые сведения (рисунок 2.7).

И в результате ещё нескольких итераций подобного процесса получается текст, показанный на рисунке 2.8.

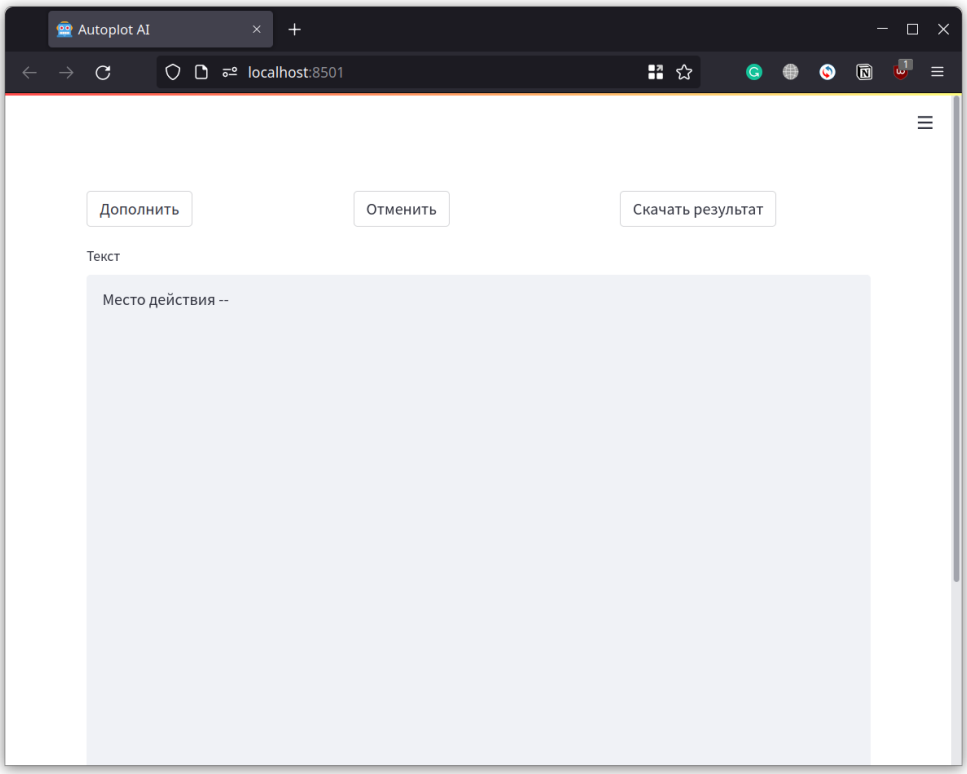

Рисунок 2.3 — Ввод затравки

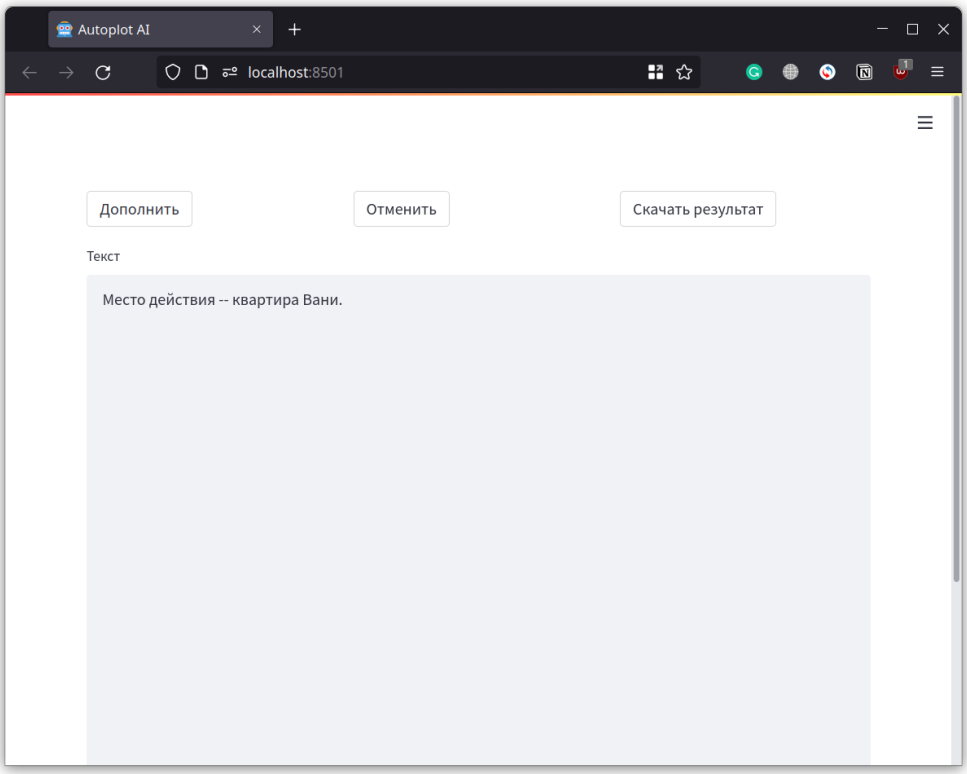

Рисунок 2.4 — Дополнение затравки

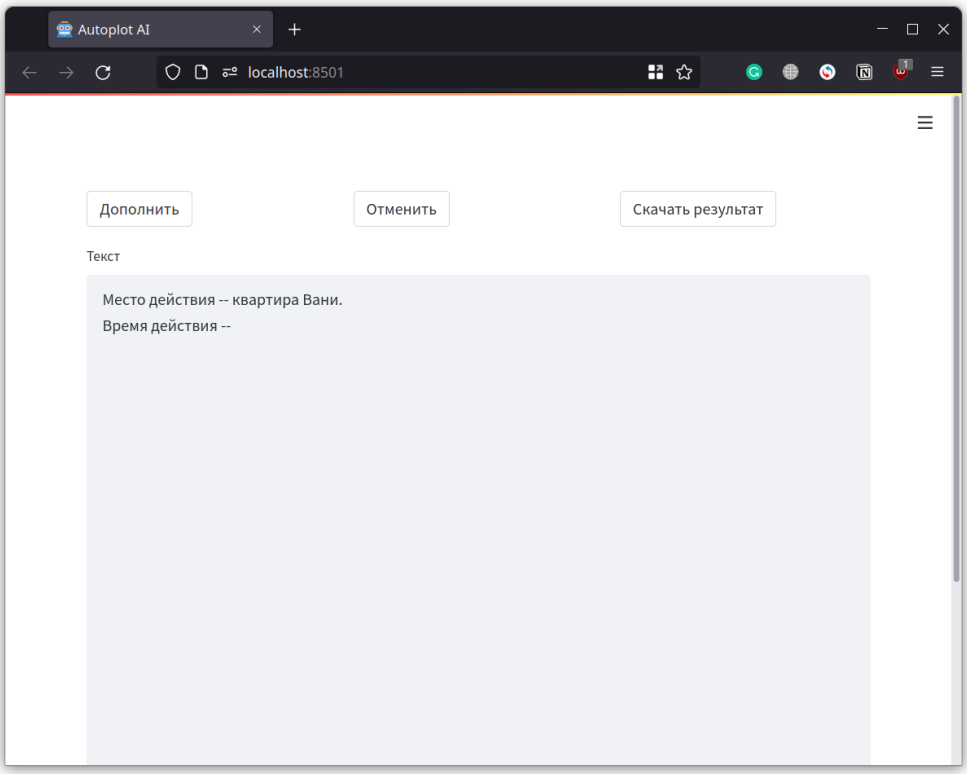

Рисунок 2.5 — Ввод дополнительной информации

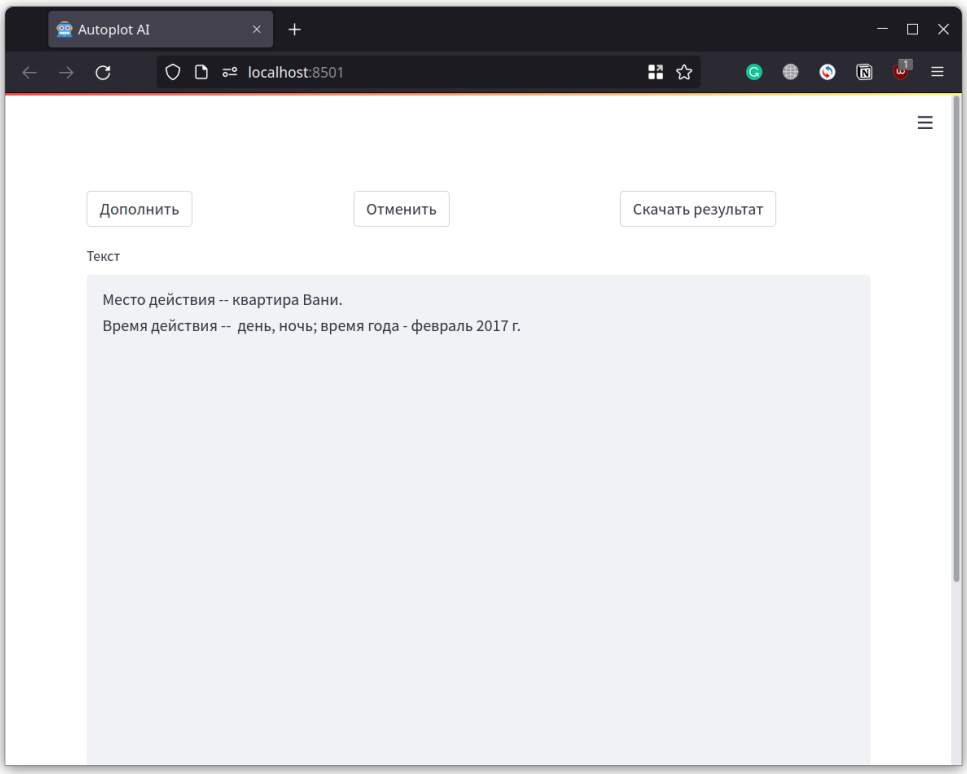

Рисунок 2.6 — Генерация продолжения

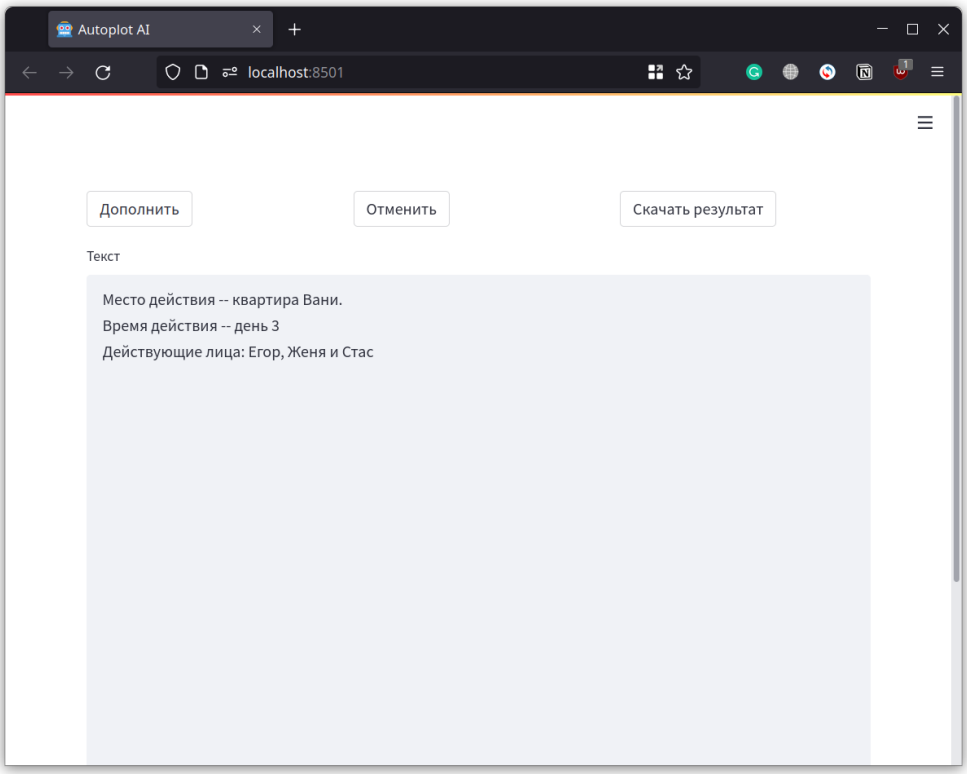

Рисунок 2.7 — Исправление и ввод новой информации

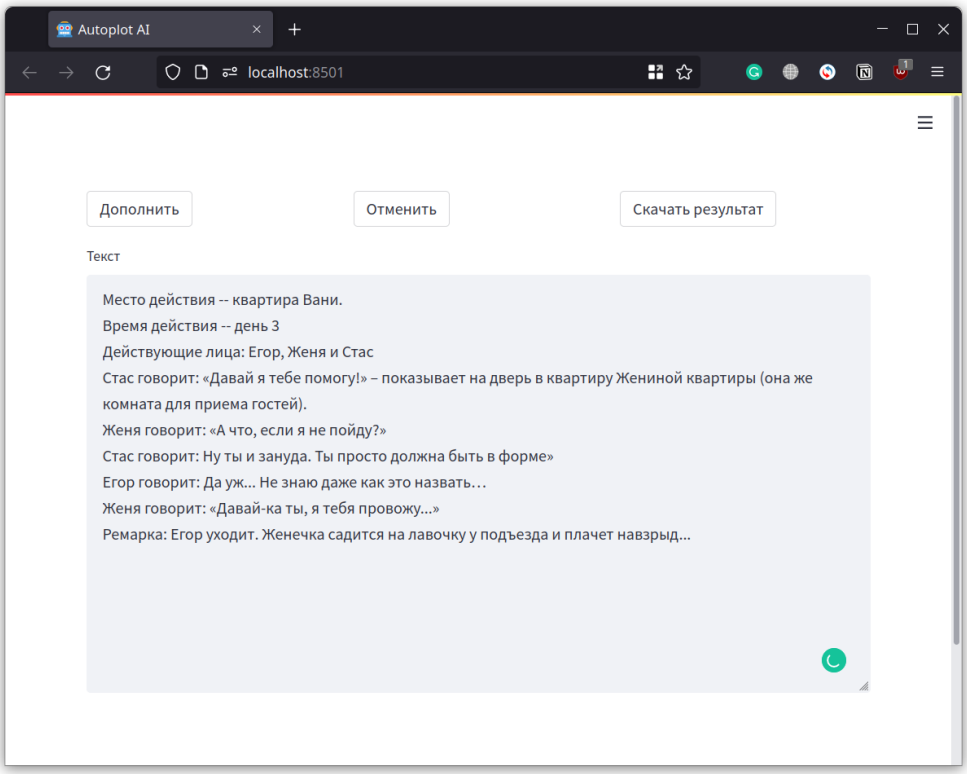

Рисунок 2.8 — Результат генерации

2.8 Дистрибутив

Для обеспечения переносимости и удобства распространения приложения все его файлы требуется поместить в один пакет и прописать инструкции по настройке и запуску программы.

Для выполнения этих требований была выбрана технология Docker. С её помощью всё, что необходимо программе для работы, а именно: исходный код, файлы модели, интерпретатор Python и библиотеки, помещаются в изолированное окружение, называемое контейнером и представляющее собой виртуальную машину с облегчённой операционной системой Ubuntu.

В таком виде для передачи программы пользователю достаточно предоставить один файл — Docker-образ, который он сможет запустить с помощью всего одной команды.

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Диагностика эмпатии по А. Меграбяну и Н. Эпштейну. — Режим доступа: https://hrliga.com/index.php?module=profession&op=view&id=847 (дата обращения: 23.05.2022).

2. Yao Mariya. 10 Leading Language Models For NLP In 2021. — Режим доступа: https://www.topbots.com/leading-nlp-language-models-2020/ (дата обращения: 17.04.2022).

3. RuGPT-3 – AI-модель для написания текстов для разработчиков, обработка естественного языка. — Режим доступа: https://developers.sber.ru/portal/products/rugpt-3ysclid=l249jkw241&attempt=1 (дата обращения: 18.04.2022).

4. ru-gpts. — Режим доступа: https://github.com/ai-forever/ru-gpts (дата обращения: 17.04.2022).

5. Streamlit documentation. — Режим доступа: https://docs.streamlit.io/ (дата обращения: 18.04.2022).

ПРИЛОЖЕНИЕ А

ЛИСТИНГИ ИСХОДНОГО КОДА

Листинг А.1 — Графический интерфейс пользователя

```
import streamlit as st

from generate import load_tokenizer_and_model, generate, CACHE_DIR


def initialize() -> None:
    """Initialize session state and set page config"""

    st.set_page_config(
        page_title="Autoplot AI",
        page_icon="□",
        layout="wide",
        initial_sidebar_state="collapsed"
    )

    if "model" not in st.session_state or "tokenizer" not in st.session_state:
        with st.spinner("Loading model"):
            tokenizer, model = load_tokenizer_and_model(CACHE_DIR)
            st.session_state["tokenizer"] = tokenizer
            st.session_state["model"] = model

    if "text_versions" not in st.session_state:
        st.session_state["text_versions"] = [""]


def main() -> None:
    """User interface logic"""

    text_versions = st.session_state["text_versions"]
    tokenizer = st.session_state["tokenizer"]
    model = st.session_state["model"]

    button_cols = st.columns(3)
    with button_cols[0]:
        continue_btn = st.button("Дополнить")

    with button_cols[1]:
        undo_btn = st.button("Отменить")

    with button_cols[2]:
        st.download_button("Скачать результат", text_versions[-1], "result.txt")
```

```
42
43      text_container = st.empty()
44      text_area_attrs = {"label": "Текст", "height": 500}
45
46      with text_container:
47          working_text = st.text_area(value=text_versions[-1],
                **text_area_attrs)
48
49      if continue_btn:
50          if len(working_text) == 0:
51              working_text = "Место действия — "
52
53          working_text = working_text[:-100] + generate(model, tokenizer,
                working_text[-100:])[0]
54
55          with text_container:
56              st.text_area(value=working_text, **text_area_attrs)
57
58      if text_versions[-1] != working_text:
59          text_versions.append(working_text)
60          st.experimental_rerun()
61
62      if undo_btn and len(text_versions) > 1:
63          text_versions.pop()
64          working_text = text_versions[-1]
65          with text_container:
66              st.text_area(value=working_text, **text_area_attrs)
67
68
69  if __name__ == "__main__":
70      initialize()
71      main()
```

Листинг А.2 — Модуль генерации текста

```
1   import time
2   import os
3   import sys
4   import random
5
6   from zipfile import ZipFile
7
8   import numpy as np
9   import torch
10
11  from transformers import GPT2LMHeadModel, GPT2Tokenizer
12
```

```
13
14 USE_CUDA = True
15 CACHE_DIR = os.path.join(os.curdir, "model_cache")
16 SEED = random.randint(0, 1000)
17
18 if not os.path.isdir(CACHE_DIR):
19     print("Extracting model...")
20     with ZipFile("model.zip") as f:
21         f.extractall(CACHE_DIR)
22
23 device = "cuda" if torch.cuda.is_available() and USE_CUDA else "cpu"
24
25 print(f"Running on {device}")
26
27
28 def load_tokenizer_and_model(model_name_or_path):
29     print("Loading tokenizer and model from " + CACHE_DIR)
30     tokenizer = GPT2Tokenizer.from_pretrained(model_name_or_path)
31     model = GPT2LMHeadModel.from_pretrained(model_name_or_path).to(device)
32     return tokenizer, model
33
34
35 def generate(
36     model, tok, text,
37     do_sample=True, max_length=50, repetition_penalty=5.0,
38     top_k=5, top_p=0.95, temperature=1,
39     num_beams=None,
40     no_repeat_ngram_size=3
41     ):
42     input_ids = tok.encode(text, return_tensors="pt").to(device)
43     out = model.generate(
44         input_ids.to(device),
45         max_length=max_length,
46         repetition_penalty=repetition_penalty,
47         do_sample=do_sample,
48         top_k=top_k, top_p=top_p, temperature=temperature,
49         num_beams=num_beams, no_repeat_ngram_size=no_repeat_ngram_size
50     )
51     return list(map(tok.decode, out))
52
53
54 def main(beginning):
55     np.random.seed(SEED)
56     torch.manual_seed(SEED)
57
58     tok, model = load_tokenizer_and_model(CACHE_DIR)
```

```
59
60      print("Generating")
61      prev_timestamp = time.time()
62      generated = generate(model, tok, beginning, max_length=200, top_p=0.95,
            temperature=0.7)
63      time_spent = time.time() - prev_timestamp
64
65      print(generated[0])
66
67      print(f"Elapsed time: {time_spent} s.")
68
69
70  if __name__ == "__main__":
71      main(sys.argv[1])
```

Листинг А.3 — Скрипт для валидации и обработки данных

```
1   import os
2   import re
3   import shutil
4
5   from collections import namedtuple
6   from sys import argv
7   from typing import List, Union
8
9
10  DATA_PATH = argv[1]
11  if DATA_PATH[-1] != "/":
12      DATA_PATH += "/"
13
14  OUT_PATH = "humanized"
15
16  Block = namedtuple("Block", ["tag", "content"])
17
18
19  def parse(text: str) -> List[Union[Block, str]]:
20
21      text = re.sub("<<", "«", text)
22      text = re.sub(">>", "»", text)
23      text = re.sub(r"\s+|\n", " ", text)
24      s = re.sub(r"(</?\w+>)", r"[CUT]\1[CUT]", text)
25      cut = list(filter(lambda t: len(t) > 0, map(str.strip, s.split("[CUT]"))))
26
27      open_tag_pat = re.compile(r"<\w+>")
28
29      cur_errors = []
30
```

```
31      def parse_list(l: List[str]) -> List[Union[Block, str]]:
32          it = iter(l)
33          out = []
34          while True:
35              try:
36                  el = next(it)
37              except StopIteration:
38                  break
39
40              if re.match(open_tag_pat, el):
41                  tag = el[1:-1]
42                  next_it = []
43                  while not re.match(f"<\W{tag}>", el):
44                      try:
45                          el = next(it)
46                      except StopIteration:
47                          # raise SyntaxError(f"<{tag}> was not closed:
                               {out[-1]}; {' '.join(next_it)}")
48                          cur_errors.append(f"<{tag}> was not closed: {out[-1]
                               if len(out) > 0 else ''}; {' '.join(next_it)}")
49                          break
50                      else:
51                          next_it.append(el)
52                  if len(next_it) > 0:
53                      out.append(Block(tag, parse_list(next_it[:-1])))
54              else:
55                  out.append(el)
56
57          for e in out:
58              if isinstance(e, str) and re.match(r"</?.+>", e):
59                  i = out.index(e)
60                  cur_errors.append(f"Found tag in processed data: {e};
                        {out[max(i - 2, 0):i + 1]}")
61          return out
62
63      return parse_list(cut), cur_errors
64
65
66  cur_name = ""
67  def humanize(s: List[Union[Block, str]]) -> str:
68      sentences = []
69
70      global cur_name
71
72      for el in s:
73          if isinstance(el, str):
```

```
74              sentences.append(el.strip())
75          else:
76              if el.tag == "header":
77                  continue
78              elif el.tag == "footer":
79                  continue
80              elif el.tag == "remark":
81                  sentences += ["\n" + "Ремарка —", humanize(el.content)]
82              elif el.tag == "author":
83                  sentences += ["\n" + "Слова автора —", humanize(el.content)]
84              elif el.tag == "title":
85                  sentences += ["\n\n" + "Заголовок —", humanize(el.content),
                        "\n"]
86              elif el.tag == "place":
87                  sentences += ["\n" + "Место действия —",
                        humanize(el.content).strip(".") + "."]
88              elif el.tag == "time":
89                  sentences += ["\n" + "Время действия —",
                        humanize(el.content).strip(".").lower() + "."]
90              elif el.tag == "chars":
91                  sentences += ["\n" + "Действующие лица —",
                        humanize(el.content).strip(".") + "."]
92              elif el.tag == "name":
93                  cur_name = humanize(el.content).strip().capitalize()
94              elif el.tag == "line":
95                  sentences += ["\n" + cur_name, "говорит:", "«" +
                        humanize(el.content).strip(".") + "»"]
96              elif el.tag == "how":
97                  sentences.append(humanize(el.content).lower())
98
99      sentences = filter(lambda t: not re.match(r"^\W*$", t) or t == "\n",
            sentences)
100     sentences = " ".join(sentences)
101     if sentences[0] == "\n":
102         sentences = sentences[1:]
103     return sentences
104
105
106 if __name__ == "__main__":
107     paths = []
108     for root, _, files in os.walk(DATA_PATH):
109         for file in files:
110             paths.append(os.path.join(root, file))
111
112     parsed = []
113     errors = []
```

```
114     for path in paths:
115         with open(path) as file:
116             try:
117                 content = file.read()
118             except Exception as e:
119                 print(e, path)
120                 raise
121 
122         try:
123             t, e = parse(content)
124             if len(e) > 0:
125                 raise SyntaxError("\n\n\n".join(e))
126             parsed.append((path, t))
127         except SyntaxError as e:
128             errors.append((path, e))
129             continue
130 
131     print(f"Errors occured in {len(errors)} files")
132     print(f"Successfully parsed {len(parsed)} files")
133 
134     if os.path.isdir(os.path.join("errors", "data")):
135         for f in os.listdir(os.path.join("errors", "data")):
136             os.remove(os.path.join(os.path.join("errors", "data"), f))
137     for p, e in errors:
138         os.makedirs(os.path.join("errors", "data"), exist_ok=True)
139         path = os.path.join("errors", "data", os.path.split(p)[-1])
140         with open(path + ".log", "w") as f:
141             f.write(str(e))
142 
143         os.system(f"cp '{p}' '{path}'")
144         # break
145 
146     if os.path.isdir(OUT_PATH):
147         for f in os.listdir(OUT_PATH):
148             os.remove(os.path.join(OUT_PATH, f))
149     os.makedirs(OUT_PATH, exist_ok=True)
150 
151     for i, (path, script) in enumerate(parsed):
152         text = humanize(script)
153 
154         new_path = os.path.join(OUT_PATH, re.sub(DATA_PATH, "", path))
155         os.makedirs(os.path.split(new_path)[0], exist_ok=True)
156         with open(new_path, "w") as f:
157             f.write(text)
158 
159     if os.path.isdir("invalid_files"):
```

```
160             for f in os.listdir("invalid_files"):
161                 os.remove(os.path.join("invalid_files", f))
162         for path, _ in errors:
163             os.makedirs("invalid_files", exist_ok=True)
164             shutil.copy(path, "invalid_files")
```