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 16 стр., 1 источн., 1 прил.

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

В работе представлено решение задачи дообучения языковой модели архитектуры Transformer для генерации стилизованного текста из заданной предметной области.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. [1]

ОСНОВНАЯ ЧАСТЬ

1 Теоретическая часть

1.1 Начало

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

ЗАКЛЮЧЕНИЕ

Приходим к выводу…

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

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 = []
```

```
    for path in paths:
        with open(path) as file:
            try:
                content = file.read()
            except Exception as e:
                print(e, path)
                raise

        try:
            t, e = parse(content)
            if len(e) > 0:
                raise SyntaxError("\n\n\n".join(e))
            parsed.append((path, t))
        except SyntaxError as e:
            errors.append((path, e))
            continue

    print(f"Errors occured in {len(errors)} files")
    print(f"Successfully parsed {len(parsed)} files")

    if os.path.isdir(os.path.join("errors", "data")):
        for f in os.listdir(os.path.join("errors", "data")):
            os.remove(os.path.join(os.path.join("errors", "data"), f))
    for p, e in errors:
        os.makedirs(os.path.join("errors", "data"), exist_ok=True)
        path = os.path.join("errors", "data", os.path.split(p)[-1])
        with open(path + ".log", "w") as f:
            f.write(str(e))

        os.system(f"cp '{p}' '{path}'")
        # break

    if os.path.isdir(OUT_PATH):
        for f in os.listdir(OUT_PATH):
            os.remove(os.path.join(OUT_PATH, f))
    os.makedirs(OUT_PATH, exist_ok=True)

    for i, (path, script) in enumerate(parsed):
        text = humanize(script)

        new_path = os.path.join(OUT_PATH, re.sub(DATA_PATH, "", path))
        os.makedirs(os.path.split(new_path)[0], exist_ok=True)
        with open(new_path, "w") as f:
            f.write(text)

    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")
```