



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.

РЕФЕРАТ

Отчёт содержит 16 стр., 1 источн., 1 прил.

**ГЕНЕРАЦИЯ ТЕКСТА, НЕЙРОННЫЕ СЕТИ, NLP, TRANSFORMER,
ДООБУЧЕНИЕ, GPT-3**

В работе представлено решение задачи дообучения нейросетевой языковой модели GPT-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	4
Основная часть	5
1. Теоретическая часть	6
1.1 Начало	6
Заключение	7
Список использованных источников	8
Приложение А Листинги исходного кода	9

ВВЕДЕНИЕ

[1]

ОСНОВНАЯ ЧАСТЬ

1. Теоретическая часть

1.1 Начало

Lorem ipsum dolor sit amet, consectetur 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, consectetur 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.

$$E = mc^2 \tag{1.1}$$

ЗАКЛЮЧЕНИЕ

Приходим к выводу...

Lorem ipsum dolor sit amet, consectetur 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, consectetur 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

```
1 import streamlit as st
2
3 from generate import load_tokenizer_and_model, generate, CACHE_DIR
4
5
6 def initialize() -> None:
7     """Initialize session state and set page config"""
8
9     st.set_page_config(
10         page_title="Autoplot AI",
11         page_icon="📊",
12         layout="wide",
13         initial_sidebar_state="collapsed"
14     )
15
16     if "model" not in st.session_state or "tokenizer" not in st.session_state:
17         with st.spinner("Loading model"):
18             tokenizer, model = load_tokenizer_and_model(CACHE_DIR)
19             st.session_state["tokenizer"] = tokenizer
20             st.session_state["model"] = model
21
22     if "text_versions" not in st.session_state:
23         st.session_state["text_versions"] = [""]
24
25
26 def main() -> None:
27     """User interface logic"""
28
29     text_versions = st.session_state["text_versions"]
30     tokenizer = st.session_state["tokenizer"]
31     model = st.session_state["model"]
32
33     button_cols = st.columns(3)
34     with button_cols[0]:
35         continue_btn = st.button("Дополнить")
36
37     with button_cols[1]:
38         undo_btn = st.button("Отменить")
39
40     with button_cols[2]:
41         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],
48                                     **text_area_attrs)
49
50     if continue_btn:
51         if len(working_text) == 0:
52             working_text = "Место действия — "
53
54         working_text = working_text[:-100] + generate(model, tokenizer,
55                                                         working_text[-100:])[0]
56
57         with text_container:
58             st.text_area(value=working_text, **text_area_attrs)
59
60     if text_versions[-1] != working_text:
61         text_versions.append(working_text)
62         st.experimental_rerun()
63
64     if undo_btn and len(text_versions) > 1:
65         text_versions.pop()
66         working_text = text_versions[-1]
67         with text_container:
68             st.text_area(value=working_text, **text_area_attrs)
69
70 if __name__ == "__main__":
71     initialize()
72     main()

```

Листинг A.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.getcwd(), "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,
63                           temperature=0.7)
64     time_spent = time.time() - prev_timestamp
65
66     print(generated[0])
67
68     print(f"Elapsed time: {time_spent} s.")
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:
48                     # {out[-1]}; {' '.join(next_it)}")
49                     cur_errors.append(f"<{tag}> was not closed: {out[-1]
50                     if len(out) > 0 else ''}; {' '.join(next_it)}")
51                     break
52                 else:
53                     next_it.append(el)
54             if len(next_it) > 0:
55                 out.append(Block(tag, parse_list(next_it[: -1])))
56             else:
57                 out.append(el)
58
59         for e in out:
60             if isinstance(e, str) and re.match(r"</?.+>", e):
61                 i = out.index(e)
62                 cur_errors.append(f"Found tag in processed data: {e};
63                 {out[max(i - 2, 0):i + 1]}")
64
65         return out
66
67     return parse_list(cut), cur_errors
68
69 cur_name = ""
70 def humanize(s: List[Union[Block, str]]) -> str:
71     sentences = []
72
73     global cur_name
74
75     for el in s:
76         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),
86                           "\n"]
87         elif el.tag == "place":
88             sentences += ["\n" + "Место действия —",
89                           humanize(el.content).strip(".") + "."]
90         elif el.tag == "time":
91             sentences += ["\n" + "Время действия —",
92                           humanize(el.content).strip(".").lower() + "."]
93         elif el.tag == "chars":
94             sentences += ["\n" + "Действующие лица —",
95                           humanize(el.content).strip(".") + "."]
96         elif el.tag == "name":
97             cur_name = humanize(el.content).strip().capitalize()
98         elif el.tag == "line":
99             sentences += ["\n" + cur_name, "говорит:", "«" +
100                           humanize(el.content).strip(".") + "»"]
101         elif el.tag == "how":
102             sentences.append(humanize(el.content).lower())
103
104     sentences = filter(lambda t: not re.match(r"^\W*$", t) or t == "\n",
105                       sentences)
106     sentences = " ".join(sentences)
107     if sentences[0] == "\n":
108         sentences = sentences[1:]
109     return sentences
110
111 if __name__ == "__main__":
112     paths = []
113     for root, _, files in os.walk(DATA_PATH):
114         for file in files:
115             paths.append(os.path.join(root, file))
116
117     parsed = []
118     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 occurred 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")
```