

Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,
МЕХАНИКИ И ОПТИКИ”

ФАКУЛЬТЕТ СРЕДНЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3
«Методы оптимизации модели RAG»

Выполнили:

студенты группы К33421

Максимов Д. Э.

Ковалев В. Д.

Азаренков Г.Д.

Санкт-Петербург
2024

ЗАДАЧИ

Применить методы оптимизации для расширенной модели LLM.

ЭТАПЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

1. Для модели из лабораторной работы 1-2:
 - а. Применить методы оптимизации для RAG (очистка данных, разбивка на блоки, и т.д. см. п. 3 в ссылках);
 - і. Сравнить результаты:
 1. до внедрения RAG (модель использует общие знания);
 2. после внедрения RAG (модель использует новые данные);
 3. после внедрения RAG + применение оптимизации.
2. По результатам тестирования, определить, при каких обстоятельствах LLM сильно галлюцинирует (например, использование узких контекстных терминов). Предложить возможные решения для найденных проблем, провести повторное тестирование.

ХОД ВЫПОЛНЕНИЯ

Для тестирования возьмем датасет “documentation” с документацией библиотеки “python-telegram-bot”, используемый также в предыдущей лабораторной работе. Из трех он наиболее приближен к реальному сценарию использования — ответы на вопросы по документации какой-то библиотеки. Часто у нейросети в тренировочных данных присутствует старая версия библиотеки, которая уже давно перестала быть актуальной.

В качестве примера возьмем запрос “Give me minimal example of the bot setup for python-telegram-bot”. В запросе включены все данные, чтобы нейросеть без контекста была в честных условиях — известно название библиотеки и дана четкая инструкция — наиболее простой кейс — выдать пример использования библиотеки.

Запрос без контекста:

2024-12-16 09:25:51,381 [INFO] Model answered without context: Here is a minimal example of setting up a Python Telegram Bot using the 'python-telegram-bot' library:

****Step 1: Install required libraries****

```
'''bash
pip install python-telegram-bot
'''
```

****Step 2: Create a token file (not shown) with your bot's API token****

You can get this token by creating a new bot in Telegram and following the instructions at [BotFather](<https://t.me/BotFather>).

****bot.py file****

```
'''python
import os
from telegram.ext import Updater, CommandHandler

TOKEN = 'YOUR_BOT_TOKEN_HERE' # from your token file

def start(update, context):
    """Send a message when the command /start is issued."""
    update.message.reply_text('Hello!')

def main():
    updater = Updater(TOKEN, workers=0)

    dp = updater.dispatcher

    dp.add_handler(CommandHandler("start", start))

    updater.start_polling()
    updater.idle()

if __name__ == '__main__':
    main()

'''
```

****How to run the bot****

```
'''bash
python bot.py
'''
```

This will start your bot, and when someone sends a '/start' command to it, it will reply with 'Hello!'.

Note: This is a very basic example, you should add more functionality and error handling depending on your needs.

Запрос с контекстом:

2024-12-16 09:24:50,528 [INFO] Model answered: Here's a minimal example to get started with Python-Telegram-Bot:

First, you need to install the required libraries. Run this command in your terminal:

```
```bash
pip install python-telegram-bot
```
```

Next, create a new file 'bot.py' and add the following code:

```
```python
import logging

from telegram import Update
from telegram.ext import ApplicationBuilder, CommandHandler, MessageHandler

Enable logging
logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
 level=logging.INFO)

Telegram bot token (obtained from @BotFather)
TOKEN = 'YOUR_BOT_TOKEN_HERE'

def start(update: Update, _):
 """Send a message when the command /start is issued."""
 update.message.reply_text('Hello! This is my Python-Telegram-Bot.')

def echo(update: Update, _):
 """Echo any incoming messages."""
 update.message.reply_text(update.message.text)

def main():
 # Create the Application and pass it your bot's token
 app = ApplicationBuilder().token(TOKEN).build()

 # Add a command /start
 app.add_handler(CommandHandler('start', start))

 # Add a message handler to handle incoming messages
 app.add_handler(MessageHandler(Filters.text, echo))

 # Run the application
 print(f'Starting bot with token {TOKEN}...')
 app.run_polling()

if __name__ == '__main__':
 main()
```
```

Make sure to replace 'YOUR_BOT_TOKEN_HERE' with your actual Telegram Bot Token.

To run the bot, save this code in a file named 'bot.py', navigate to that directory in your terminal, and execute the following command:

```
```bash
python bot.py
```
```

Можно заметить, на сколько качество ответа возрастает при использовании контекста – модель берет пример из датасета, используя актуальную версию библиотеки и корректный способ ее запуска, а также добавляет полезные фичи, такие как логгирование и более большое количество стартовых команд для примера.

Модель без контекста же хоть и приводит формально корректный пример, но использует сильно устаревшую версию библиотеки, а также отвечает достаточно лаконично.

В данном случае мы не применяем дальнейшие оптимизации для третьего шага, так как ответ и так оптимален. Возьмем другой пример.

Запрос “How to create async handler of some command, using python-telegram-bot”.

2024-12-16 10:10:42,283 [INFO] Model answered: You can use the `run_async` decorator provided by the library itself, or you can manually define an asynchronous function and

Here is how you can do both:

Using run_async

```
'''python
from telegram.ext import CommandHandler, run_async

@run_async
def your_command(update: Update, context: Context):
    # Your command code here
    pass

dispatcher.add_handler(CommandHandler('your_command', your_command))
'''
```

Manually Defining Asynchronous Function

However, I would recommend using `asyncio.create_task` to create a task that is awaited by your bot.

```
'''python
import asyncio
from telegram.ext import CommandHandler, CallbackContext

def your_command(update: Update, context: Context):
    # Your command code here
    pass

async def async_your_command(context: Context):
    await your_command(context)

async def handle_your_command(context: Context):
    task = asyncio.create_task(async_your_command(context))
    await task

dispatcher.add_handler(CommandHandler('your_command', lambda context: handle_your_command(context)))
'''
```

Как модель с, так и без контекста, очень сильно галлюцинирует несмотря на наличие нужного примера в документации. Она выдает формально корректные, но де факто рудиментарные примеры.

Только один из примеров из документации:

```
from telegram import Update
from telegram.ext import ApplicationBuilder, ContextTypes, CommandHandler

application = ApplicationBuilder().token("TOKEN").rate_limiter(MyLimiter()).build()

async def start(update: Update, context: ContextTypes.DEFAULT_TYPE):
    # This is automatically passed through `MyLimiter.process_request`!
    await update.message.reply_text(text="Hello World!")

application.add_handler(CommandHandler('start', start))
```

Мы применили несколько разных оптимизаций, описанных в статьях в приложении, наиболее эффективным оказалось переформулирование запроса. После обработки Llm выдает полностью корректный код, который пользователь и хотел бы видеть изначально. В выборке, полученной в результате поиска по векторной базе, появились нужные статьи.

```

"async handlers python-telegram-bot"
2024-12-16 10:26:32,140 [INFO] Search completed. Top 5 matches: [{ 'data': { 'name': 'Transition-guide-to-Version-5.0.md', 'description': '## JobQueue\nWe did some se
2024-12-16 10:26:32,146 [INFO] Sending question to model with context...
2024-12-16 10:28:57,517 [INFO] Model answered: To create an async handler for a command in the `python-telegram-bot` library, you can use the `async def` syntax and
Below are examples of how to accomplish both scenarios:

### 1. Using a CommandHandler

When using a `CommandHandler`, you don't need to specify the command manually in your handler function. The `CommandHandler` will automatically pass the command name

```python
from telegram import Update
from telegram.ext import ApplicationBuilder, CommandHandler

Your command handler function
async def help_command(update: Update, context):
 await update.message.reply_text("This is a help text.")

def main():
 # Create the application and process updates
 application = ApplicationBuilder().token('YOUR_TOKEN').build()

 # Add the command handler
 application.add_handler(CommandHandler('help', help_command))

 # Run the application
 application.run_polling()

if __name__ == '__main__':
 main()
...

```

## Код решения:

```

1 import json
2
3 import ollama
4 import faiss
5 import numpy as np
6 import logging
7 from typing import List, Dict, Any, Optional
8
9 from common.start_logging import start_logging
10
11 logging.basicConfig(level=logging.INFO, format='%(asctime)s [%(levelname)s] %(message)s')
12
13
14 def generate_embedding(text: str) -> Optional[np.ndarray]: 2 usages ± Azarenkov Georgy
15 try:
16 response = ollama.embeddings(model='nomic-embed-text', prompt=text)
17 return np.array(response['embedding']).astype('float32')
18 except Exception as e:
19 logging.error(f"Error generating embedding for text '{text}': {e}")
20 return None
21
22
23 def create_faiss_index(data: List[Dict[str, str]]) -> faiss.IndexFlatL2: 1 usage ± Azarenkov Georgy
24 logging.info("Creating FAISS index...")
25 embeddings: List[np.ndarray] = []
26
27 for entry in data:
28 embedding = generate_embedding(str(entry))
29 if embedding is not None:
30 embeddings.append(embedding)
31 else:
32 raise Exception("Failed to generate embedding for an entry.")
33
34 if not embeddings:
35 raise Exception("No embeddings generated.")
36
37 dimension: int = embeddings[0].shape[0]
38 index = faiss.IndexFlatL2(dimension)
39 index.add(np.array(embeddings))
40 logging.info("FAISS index created successfully.")
41 return index
42
43

```

```

44 def search_in_dataset(1 usage ± Azarenkov Georgy
45 query: str,
46 index: faiss.IndexFlatL2,
47 data: List[Dict[str, Any]]
48) -> List[Dict[str, Any]]:
49 logging.info(f"Searching for query: '{query}'")
50 query_embedding = generate_embedding(query)
51
52 if query_embedding is None:
53 raise Exception("Failed to generate query embedding.")
54
55 k = 5
56 distances, indices = index.search(np.array([query_embedding]), k)
57 results = []
58 for i, idx in enumerate(indices[0]):
59 result = data[idx]
60 result_distance = distances[0][i]
61 results.append({"data": result, "distance": result_distance})
62
63 logging.info(f"Search completed. Top {k} matches: {results}")
64 return results
65
66
67 def send_question_to_model(query: str) -> str: 1 usage ± Azarenkov Georgy
68 logging.info("Sending question to model...")
69
70 response = ollama.chat(model='llama3.1:latest', messages=[
71 {
72 "role": "system", "content": '''
73 You are assistant that helps answering questions.
74 '''
75 },
76 {
77 "role": "user", "content": query
78 }
79])
80 answer = response["message"]["content"]
81 return answer

```

```

83 def reformulate_for_rag_search(query: str) -> str: 1 usage new *
84 logging.info("Reformulating...")
85
86 response = ollama.chat(model='llama3.1:latest', messages=[
87 {
88 "role": "system", "content": '''
89 You are reformulating the user query for search over some vector database. Remove redundant words
90 that bring no meaning etc.
91 RETURN SHORT MODIFIED USER QUERY THAT WILL BE FED TO ANOTHER MODEL. DO NOT ANSWER THE QUESTION.
92 '''
93 },
94 {
95 "role": "user", "content": query
96 }
97])
98 answer = response["message"]["content"]
99 return answer
100
101 def send_question_to_model_with_context(query: str, top_entries: List[Dict[str, Any]]) -> str: 1 usage ± Azarenkov Georgy
102 context_lines = "\n".join([f"piece{i + 1}: {entry['data']}" for i, entry in enumerate(top_entries)])
103
104 logging.info("Sending question to model with context...")
105 response = ollama.chat(model='llama3.1:latest', messages=[
106 {
107 "role": "system", "content": '''
108 You are assistant that helps answering questions about dataset. You are given some entries in the dataset,
109 that can help you answer.
110 Include ALL possible results based on tool input. But it is not necessary, if some of them are not
111 relevant.
112 '''
113 },
114 {
115 "role": "tool", "content": context_lines
116 },
117 {
118 "role": "user", "content": query
119 }
120])
121 answer = response["message"]["content"]
122 return answer

```

```

124 def print_response(user_query: str, dataset: list[dict]) -> None: 1 usage ± Azarenkov Georgy *
125 faiss_index = create_faiss_index(dataset)
126
127 reformulated_user_query = reformulate_for_rag_search(user_query)
128
129 top_matches = search_in_dataset(reformulated_user_query, faiss_index, dataset)
130
131 response = send_question_to_model_with_context(user_query, top_matches)
132
133 logging.info(f"Model answered: {response}")
134
135 response_without_context = send_question_to_model(user_query)
136
137 logging.info(f"Model answered without context: {response_without_context}")
138
139
140 def main(): 1 usage ± Azarenkov Georgy
141 start_logging()
142 # with open('data/people.json', 'r') as file:
143 # dataset = json.load(file)
144 #
145 # main("Who has a profession associated with money?", dataset)
146
147 with open('../lab2/data/documentation.json', 'r', encoding="utf-8") as file:
148 dataset = json.load(file)
149
150 print_response(user_query: "How to create async handler of some command, using python-telegram-bot", dataset)
151
152
153 > if __name__ == '__main__':
154 main()
155

```

## Галлюцинации и повторное тестирование

В рамках поиска запросов, которые вызывают у модели галлюцинации, основная категория, которую мы нашли — запросы, ответа на которых напрямую нет в выборке, которую генерирует RAG. В таких случаях модель банально придумывает ответ, несмотря на то, что лучше было бы сказать, что не знает его, и не вводить в заблуждение. Мы попробовали изменить основной промпт, но незначительные изменения не влияли на ответ нейросети, а более серьезные хоть и частично решали проблему, но ухудшали общее качество ответов.



## Дополнительные оптимизации

Для того, чтобы сравнить низкоуровневый подход работы с RAG, а также более автоматизированную имплементацию, мы взяли фреймворк “Llama-index” и протестировали его на наших данных. Он сильно упрощает имплементацию и автоматически применяет большое количество оптимизаций.

```
1 import json
2 import logging
3 from llama_index.core import VectorStoreIndex, Document
4 from llama_index.embeddings.ollama import OllamaEmbedding
5 from llama_index.llms.ollama import Ollama
6 from llama_index.core import Settings
7
8 from common.start_logging import start_logging
9 from lab2.manual_rag import send_question_to_model
10
11
12 def create_index_from_json(dataset: list[dict]) -> VectorStoreIndex: 1 usage new *
13 documents = [Document(text=json.dumps(entry)) for entry in dataset]
14
15 embed_model = OllamaEmbedding(model_name='nomic-embed-text')
16 llm = Ollama(model='llama3.1:latest')
17
18 Settings.embed_model = embed_model
19 Settings.llm = llm
20
21 logging.info("Creating index from JSON data...")
22 index = VectorStoreIndex.from_documents(documents)
23 logging.info("Index created successfully.")
24 return index
25
26
27 def query_index(index: VectorStoreIndex, query: str) -> str: 1 usage new *
28 logging.info(f"Querying the index with: '{query}'")
29 query_engine = index.as_query_engine()
30 response = query_engine.query(query)
31 return str(response)
32
33
34 def create_query_function(dataset: list[dict]): 3 usages new *
35 index = create_index_from_json(dataset)
36
37 return lambda query: query_index(index, query)
38
```

```

40 def main(): 1 usage new *
41 start_logging()
42
43 with open('../lab2/data/documentation.json', 'r', encoding="utf-8") as file:
44 dataset = json.load(file)
45
46 query_function = create_query_function(dataset)
47
48 user_query = "List some performance optimizations for python-telegram-bot"
49
50 response = query_function(user_query)
51
52 logging.info(f"Response: {response}")
53
54 response_without_context = send_question_to_model(user_query)
55
56 logging.info(f"Response without context: {response_without_context}")
57
58 if __name__ == '__main__':
59 main()
60

```

Нам не удалось применить этот фреймворк к датасету “documentation”, так как в ответах выдавало “timeout error”, и не удалось решить эту проблему. Поэтому мы применили его к датасету с карточками. Ответы получились более емкими/лаконичными, скорее всего это было достигнуто за счет внутреннего промта и локальных оптимизаций.

## ВЫВОД

Явно видна разница между результатами с и без RAG – модель хоть и имела в датасете информацию о библиотеке, но сильно устаревшую, что и продемонстрировала. При этом, оптимизация RAG также дала результат в виде прямого улучшения его работы – llm получила сильно более релевантную выборку.