**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

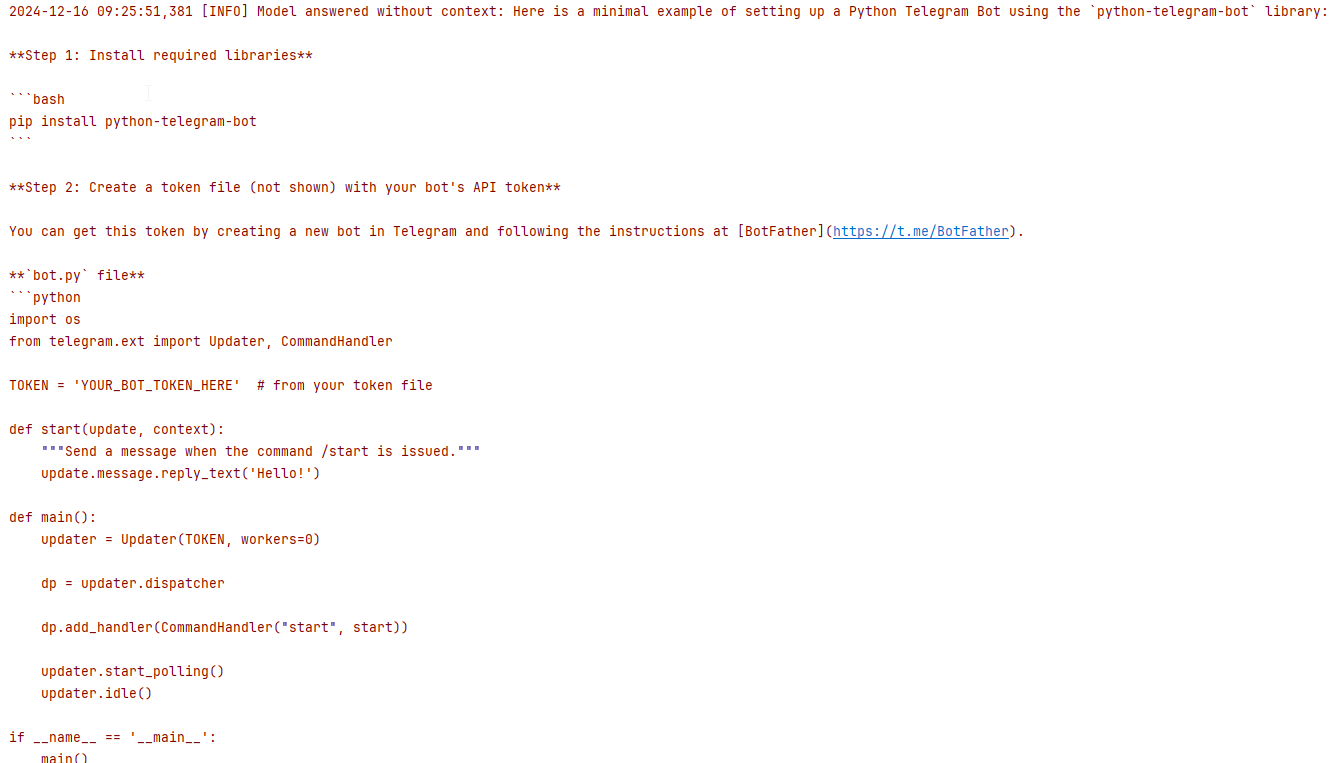
1. Для модели из лабораторной работы 1-2:
   1. Применить методы оптимизации для RAG (очистка данных, разбивка на блоки, и т.д. см. п. 3 в ссылках);
      1. Сравнить результаты:
         1. до внедрения RAG (модель использует общие знания);
         2. после внедрения RAG (модель использует новые данные);
         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 четка инструкция – наиболее простой кейс – выдать пример использования библиотеки.

Запрос без контекста:



Изображение выглядит как текст, снимок экрана, Шрифт, алгебра

Автоматически созданное описание

Запрос с контекстом:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, алгебра

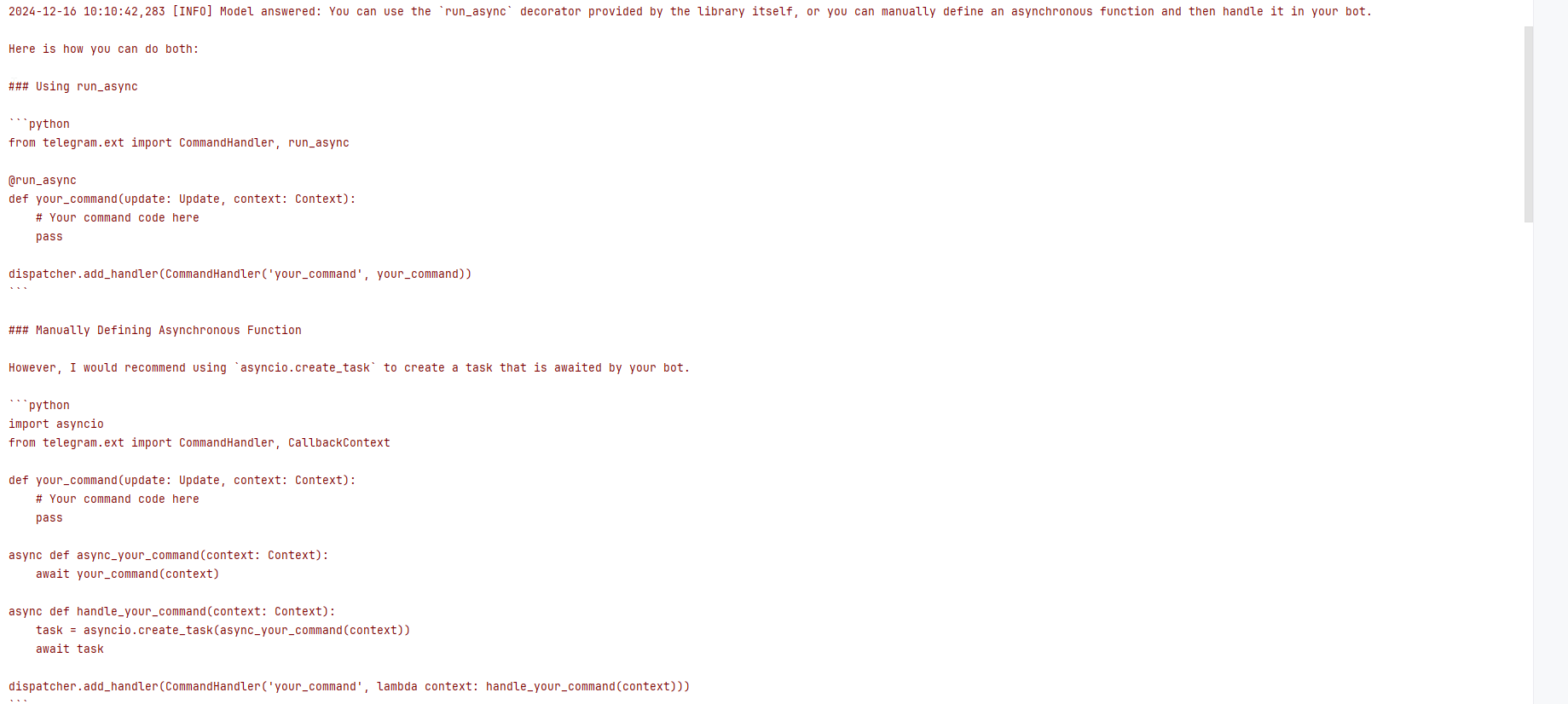
Автоматически созданное описание

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.

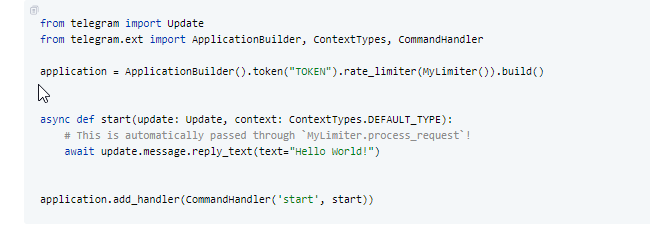
В данном случае мы не применяем дальнейшие оптимизации для третьего шага, так как ответ и так оптимален. Возьмем другой пример.

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



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:



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.



Код решения:



Изображение выглядит как текст, снимок экрана, документ, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, документ, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, документ

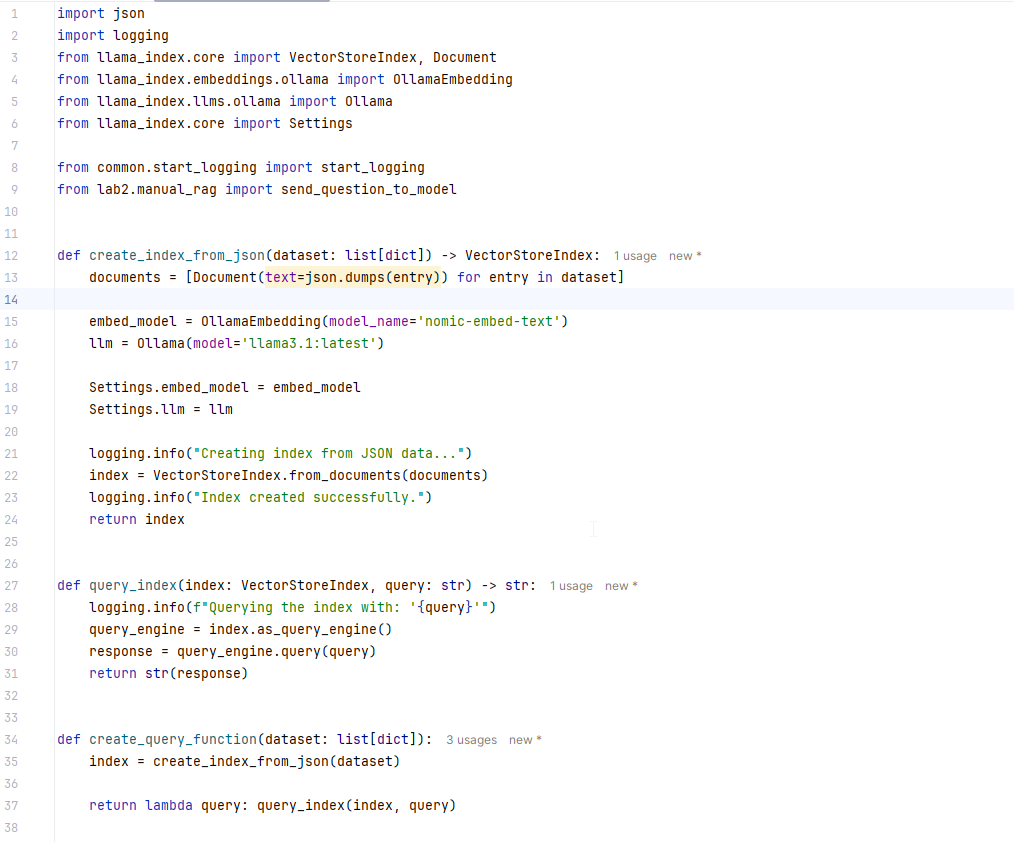
Автоматически созданное описание

**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.



Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Нам не получилось применить этот фреймворк к датасету “documentation”, так как в ответах выдавало “timeout error”, и не получилось 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.