Софийски университет „Св. Кл. Охридски”

Факултет по математика и информатика

Интелигентни агенти с генеративен изкуствен интелект

Курсов Проект

на тема:

„Research Buddy”

Студент: **Петър Йорданов Петров** Ф.Н. **5MI0800197**

Курс: 4, Учебна година: 2025/26

Съдържание

1 Цел на проекта

Целта на проекта е да помогне на хора, които имат интерес към науката и биха желали да навлязат в научната среда чрез четенето на научни статии. Това е много трудна задача за човек, който няма опит в самата сфера (или като цяло с четенето на научни статии), защото е много трудно да се проследи историята на развитие на дадена подсфера, но дори това да се направи, откритията могат да бъдат в „разбъркан ред“.

Проектът представлява уеб приложение използващо изкуствен интелект, който филтрира всички релевантни научни статии по заявка на потребителя. Това е т.нар. функционалност за търсене.

След това потребителят може да инспектира по-подробно дадена статия, да получи автоматична сегментация на съдържанието ѝ, и при наличие на въпроси, за всеки сегмент да попита/чати с агент.

2 Данни

Проектът представлява версия на Retrieval Augmented Generation (RAG) архитектурата. За справка, моделът използва база от данни с метаданни за повечето научни статии в STEM сферите от arXiv.

https://www.kaggle.com/datasets/Cornell-University/arxiv/data

Всяка научна статия съдържа следните метаданни, които са релевантни за Research Buddy:

- id на статията, с което може лесно да се достъпи съдържанието на pdf файла
- authors – низ с авторите на статията, изброени
- title – заглавие на статията
- categories – низ с изброените категории/сфери, които заляга статията (напр. phys, math.CA, …)
- abstract – резюмето на статията в началото ѝ
- update_date – последна дата на подновяване

3 Функционални изисквания

Проектът се разделя основно на 2 функционалности:

1. Екран/опция за търсене на статии по вход на потребителя

Потребителят има възможността да въведе вход в търсачката и при всяко натискане на клавиш, се изпраща заявка към сървъра, която да изведе топ K резултатите от заявката.

С цел оптимизиране на броя заявки и натовареността на сървъра, самото изпращане на заявки е *debounce*-нато.

Изведените резултати са снабдени със заглавие на статията, съдържание на резюмето ѝ, авторите ѝ и всички останали метаданни. Също така е визуализиран и числов резултат, който е метрика за това колко подобен/релевантен е документът спрямо заявката.

При натискането върху някоя статия, потребителят бива препратен към друг екран с по-подробна информация за нея.

2. Екран за визуализиране на статия и анализ

Когато потребителят бива препратен върху екрана за специфична статия, автоматично ще му бъде визуализирано нейното съдържание във формата на *.pdf* файл.

Също така, допълнително ще му бъде извършена т.нар. сегментация на текста. Съдържанието на статията ще бъде разделено по смисъл от изкуствен интелект на отделни парчета, които са близки по смисъл и това ще бъде визуализирано по подходящ начин на потребителя като подчертано.

Също така, диаграмите/таблиците/формулите ще бъдат подчертани.

Потребителят ще има възможността да избере подчертан елемент и ще му бъде предоставена опцията да проведе чат с изкуствен интелект относно този елемент.

За всеки подчертан елемент, ще може да се проведе различен чат. Но отговорите на изкуствения интелект ще имат следното предвид:

- Цялата история от чатове върху всички елементи
- Цялата информация в статията
- Допълнителна информация от други статии в базата данни

4 Архитектура и модели

Приложението има 2 основни компонента:

- SPA клиент на React, който извършва самата визуализация на pdf-а, на резултатите от заявката за търсене и на съобщенията в чатовете
- Flask сървър, който в отделни endpoint-и извършва функционалността за търсене, сегментиране и чат.

За съхранение на документите се използва *ChromaDB* с embedding-и от all-MiniLM-L6-v2.

Също така, при сегментиране на дадена статия, в отделна колекция на същата база данни се пазят и embedding-ите за парчетата на самата статия.

Използваният LLM е llama-3.3-70b-versatile, предоставен от API-то на Groq

Той е използван като ReAct агент с няколко tool-ове, с които може да прави следното:

- Да търси текст в целия документ в текущата научна статия
- Да търси в останалите документи в базата, по текст или по метаданни

Сегментирането на данните се извършва чрез просто chunk-ване по брой token-и и след това чрез сравняване на embedding-ите между съседни вектори. Ако близостта спадне под даден threshold, който се изчислява динамично, двата съседни парчета текст са от различни сегменти.

Търсенето на документи преминава през 3 етапа:

1. Чрез структуриран изход, LLM-ът извлича ограничения спрямо метаданните, по които да търси в базата данни.
2. Преминаване на входа на потребителя през Hypothetical Document Embeddings (HyDE), което позволява на доста абстрактна заявка напр. от вида „Предложи ми статии за компютърни мрежи“, да се превърне в заявка от вида „Статиите за компютърни мрежи биват следните няколко вида ...“. Моделът изкуствено генерира изход, който съдържа повече ключови думи, с които търсенето може да се обогати значително.

5 Библиотеки

Frontend:

- React.js
- Material UI
- React Markdown – за по-красиво показване на изхода на модела при чат

- React PDF- за автоматично извличане и визуализиране на PDF

Backend:

- chromadb
- flask
- huggingface_hub – за Embedding модела
- langchain + langgraph – за създаване на ReAct агент с множество tool-ове и за интерфейс при работа с документи от данни
- PyMuPDF – за извличане на bounding boxes на различните параграфи в текста, което позволява лесното визуализиране/подчертаване

6 Източници

- https://docs.langchain.com/
- https://dev.to/simplr_sh/the-best-way-to-chunk-text-data-for-generating-embeddings-with-openai-models-56c9