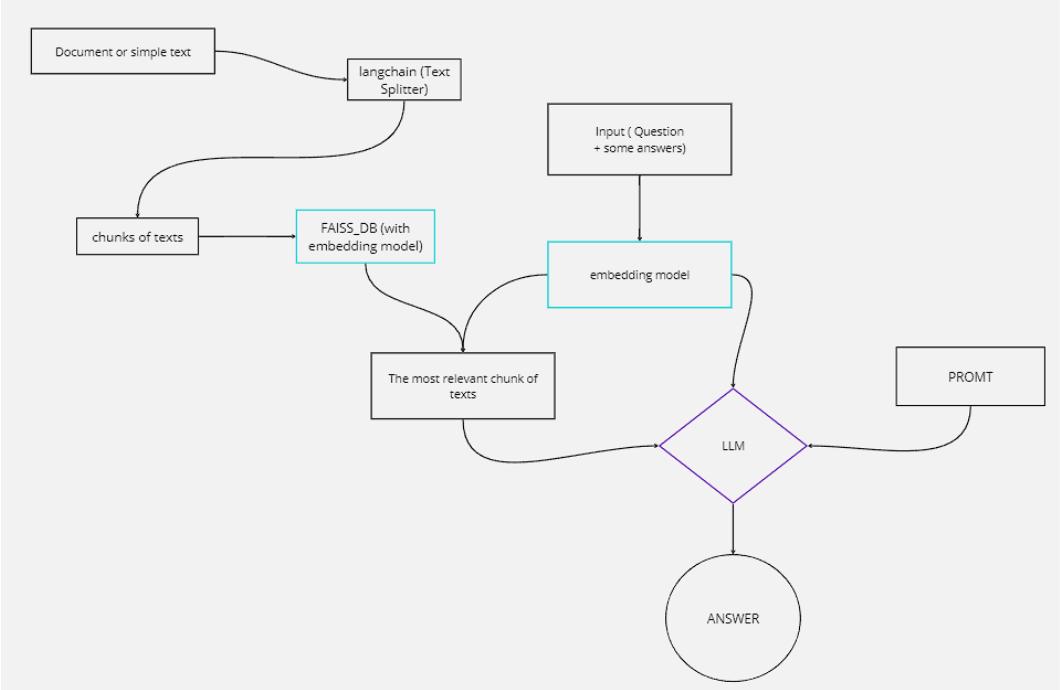
Проект: Вопросно – ответная система. (+Сравнение качества ответов различных архитектур open-source Mistral 13b и openai gpt-3.5, 4 с соответствующими энкодерами)

Вход – документ + вопрос + варианты ответа

Выход – правильный (так предполагается) вариант ответа из списка.



Базовый pipeline:

1. Документ попадает в систему, из него достаётся текст.
2. Текст сплитися на чанки. Которые в дальнейшем обрабатываются с помощью encoders и записываются в векторную базу данных.
3. Пользователь задаёт вопрос и варианты ответа. С помощью энкодера происходит векторизация и поиск «ближайшего» чанка из базы данных.
4. На основе выбранного чанка формируется prompt для модели.
5. Модель генерирует ответ.

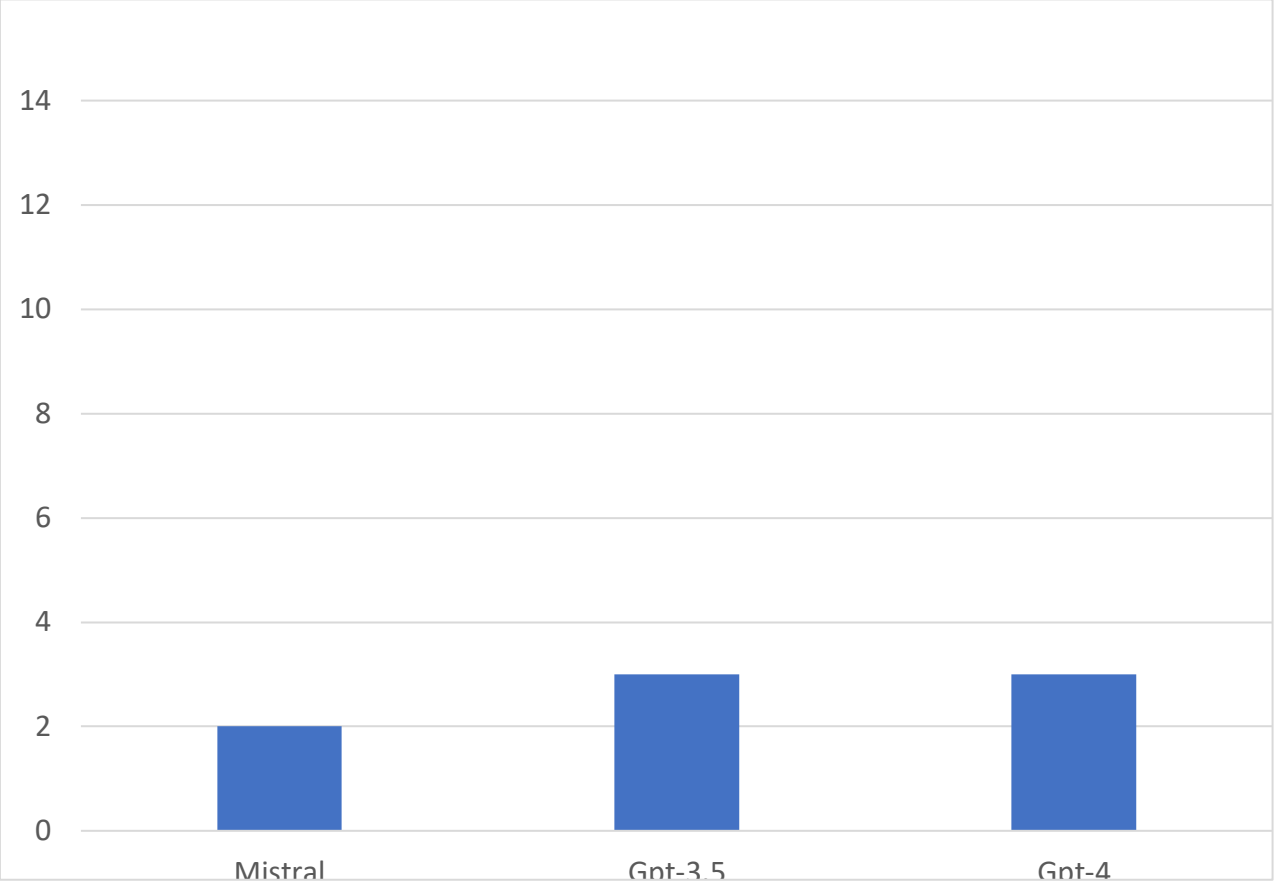
История создания.

1. Работа с данными:
   Вытаскиваем данные из pdf с помощью библиотеки pypdf (fuzz). И производим предобработку. Для разделения на чанки используется RecursiveCharacterTextSplitter из langchain.

2. Первые шаги:

Изначально было решено собрать некоторый baseline и посмотреть на метрики. В качестве encoder использовался **'sentence-transformers/all-MiniLM-L6-v2'** в качестве базы данных **FAISS**, а в роли модели выступала предобученная **mistralai/Mistral-7B-Instruct-v0.1.** С промптом тоже не заморачивался **– «Ответь на вопрос используя «текст» и варианты ответа: «варианты ответа».**

И вот, пайплайн создан - время тестов. В качестве датасета использовался набор набор данных с *openedu* (лекции в формате pdf, а также тесты). Запускаем тесты и… **По нулям! (ну почти) «мистраль – 2/14, openai – 3/14, 3/14)**



3. Промптирование.

После того как эйфория от успеха прошла, и получаса дебага – стало понятно, что модель часто может модернизировать варианта ответа или вообще отвечать своими словами. Самое время написать промпт, который бы

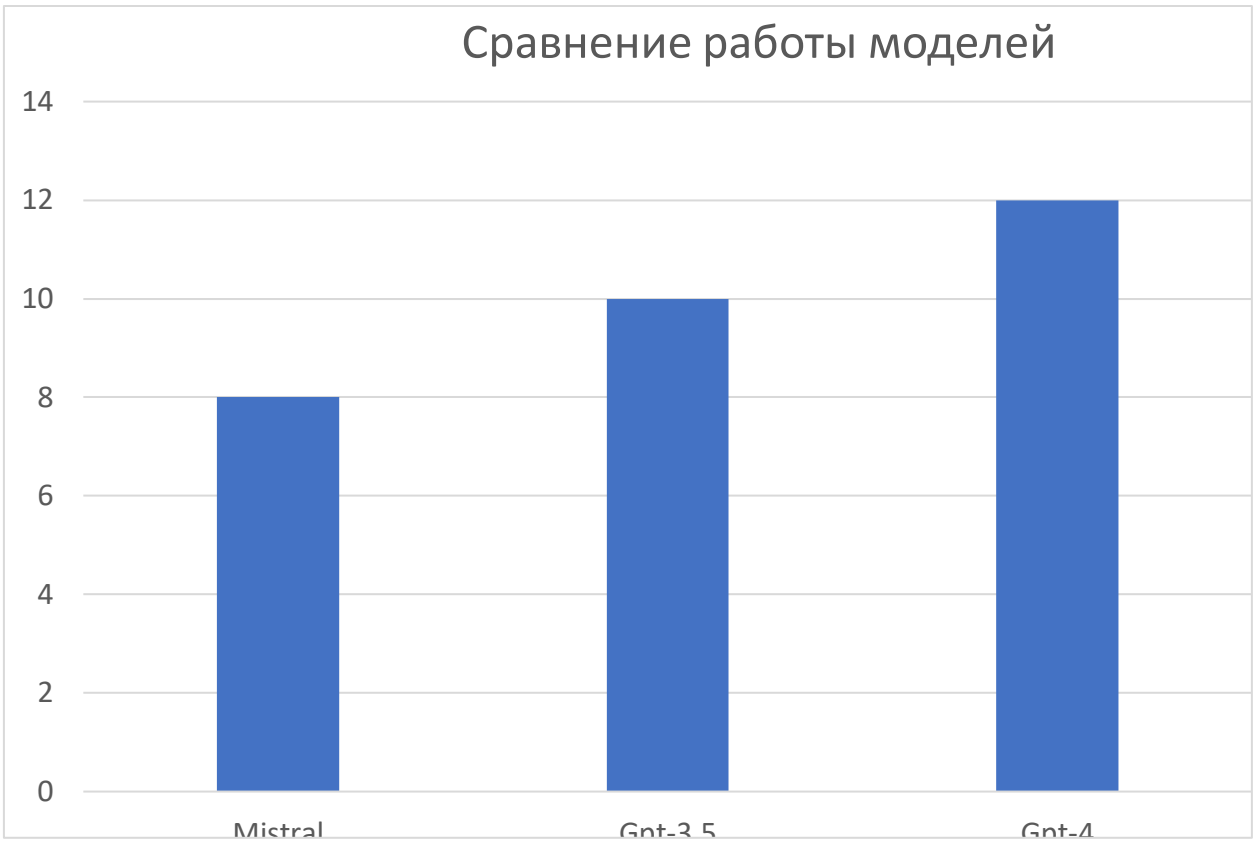
помог избежать таких проблем.

```
Prompt = f"""
Выбери правильный вариант ответа на вопрос, используя информацию из текста.
Вопрос: {question},
Текст: {text},
Варианты ответа: {answers}
В качетсве output выдай один из вариантов ответа.
"""
```

Запускаем еще раз, результаты сильно лучше. Модели от openai сразу поняли, что я от них хочу и все ответы были даны в нужном формате. Mistral же до сих пор не хотела сотрудничать, периодически выдавая галлюцинации. После нескольких неудачных попыток обратить внимание модели на формат вывода (Обрати внимание, что твой ответ должен полностью состоять из значения поля «Варианты ответа», !IMPORTANT for answer use only words from «Варианты ответа» и т.д) пришлось прибегнуть к магии. Ситуацию спасли 2 фразы: (было решено оставить первую)

1) Будь внимателен , моя карьера зависит от этого.
2) Будь внимателен, жизни людей зависят от этого.

получаем неплохой результат для небольшой модели 8/14 – **больше 50%!**). Окрыленный «успехом» прикручиваю openai модели и токенайзеры получаю еще более головокружительный результат gpt-3.5 – 10/14, gpt-4 – 12/14.



4. Пробуем улучшить:
    a. Тюним параметры:
        i. Temperature – выкручиваем в минимум (нам не нужны литературные шедевры, нужны самые вероятные варианты)
        ii. Top_p, top_k оставляем дефолтные (температура 0 – нет смысла менять)
        iii. repetition_penalty установливаем немного больше стандартного значения (1), потому что при температуре 0 mistral имела тенденцию зацикливаться на одном токене и повторять его пока возможно.
        iv. Max_token – ограничиваем максимальное количество токенов для вывода 50. (параметр появился из-за предыдущего пункта)
    b. Меняем токенайзер.
        i. Пробуем использовать самый «дорогой» openai токенайзер из коробки.

    Тестируем, и…. ничего. Метрики практически не изменились.

5. Доверяй, но проверяй.

    Вроде бы результат неплохой. Списываю неудачи mistral на её малые размеры и начинаю создавать backend на fastapi и sqlite и фронтенд на streamlit.
    Думаю, что неплохо было бы выводить не только правильный вариант ответа но и кусок текста, который использовался для генерации ответа. Пару строчек кода и готово.
    Загружаю документ, задаю вопрос, вбиваю варианты. Получаю правильный ответ. Решаю взглянуть на предложенный релевантный chunk. И оказывается что информации для ответа на вопрос в этом чанке **не** содержится. Пробую другие примеры. ПОЧТИ ВЕЗДЕ МИМО! И тут всё встает на свои места. Разница в работе между GPT и Мистраль обусловлена тем, что GPT просто обучена на большем корпусе текстов, что позволяет ей генерировать правильный ответы, даже не имея нужной для этого информации в промпте.

6. Пробуем варианты.
    a. Первое, что пришло в голову – давайте искать ближайший чанк не по вопросу, а по ответу. Т.е будем генерировать с помощью llm ответ на вопрос и его передавать для поиска релевантного куска текста. Идея быстро потухла, потому что llm может генерировать

ответы из слов, которые впринципе никак не содержатся в документе.

b. Дальше решаю попробовать разные метрики. (L2, косинусное расстояние, скалярное произведение). Положительные результаты не наблюдаются.
c. После, нескольких десятков запусков замечаю, что чанки, которые выдает FAISS очень странные. Слова могут содержать внутри себя пробелы, предложения обрываться и нередко можно встретить переносы строк прямо внутри текста. Пробую подать на вход не pdf а текст и ВУАЛЯ – релевантный чанк для всех вопросов находится в тройке лидеров. И тут приходит осознание. В своей работе я использовал библиотеку Langchain в которой содержатся стандартные средства для извлечения текста из pdf, а именно PDFLoader. Но на этапе разработки сервиса было решено заменить его на самописную реализацию из библиотеки fuzz, по причине того, что PDFLoader принимает только путь к файлу (т.е он должен быть обязательно сохранен перед передачей в метод). Переписываю реализацию через встроенные в langchain методы и результат радует. Релевантные чанки, также как и для случая с текстом – в первой тройке.

7. Новая глава.

   В процессе подробного сёрча по принципам работы векторных баз данных в голову приходит идея: «Размеры документов, которые мы храним в базе ничтожно малы. (до 15 страниц), можем позволить себе хранить не только голые вектора, но и некоторую мета информацию, которая может помочь в поиске релевантных кусков.» После чтения документации оказывается, что в FAISS без «костылей» это сделать не удасться. База рассчитана хранение пентабайт данных, поэтому мало того, что не разрешает хранить метаинформацию (кроме индексов). Так и еще сжимает входные вектора для ускорения поиска.
   Выбор пал на Chroma. Основная причина в том, что из-за специфики задачи, неплохо было бы производить поиск не только векторный поиск, но и ,например, семантический. В Chroma это доступно из коробки. Она занимает в несколько раз больше места, чем FAISS но для решаемой задачи – это не критично. После реализации и некоторых тестов получаю хороший результат – релевантный чанк определяется базой как «самый вероятный» в 12 из 14 случаев, в оставшихся двух он в первой тройке.

8. Время новых тестов.

После проделанной работы пришло осознание, что выбранный изначально датасет не покрывает всю область работы системы. Тестовые данные состоят из общих вопросов, на которые обученная на достаточно большом корпусе модель может дать ответ и без дополнительной информации. Необходимо добавить в датасет вопросы, на которые невозможно дать правильный ответ не видя документ. Сформировав новый корпус данных приступаем к тестам.