Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н. Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

Факультет *«Фундаментальные науки»*

Кафедра *«Математическое моделирование»*

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему:

ПРИМЕНЕНИЕ СИСТЕМЫ RAG ДЛЯ СОЗДАНИЯ ЧАТ-БОТА ПОМОЩНИКА ДЛЯ СТУДЕНТОВ ВУЗА.

Студент группы ФН12-71Б _____ *Н.Ю.Яшин*
(подпись, дата)

Руководитель курсовой работы _____ *Д.Н. Федянин*
(подпись, дата)

2024г.

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н. Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой ФН-12

_____ А.П. Крищенко

«____» _____ 20___ г.

ЗАДАНИЕ
на выполнение курсовой работы

по дисциплине

_____

Студент группы ФН12-71Б

Яшин Николай Юрьевич

(Фамилия, Имя, Отчество)

Тема курсовой работы: **Применение системы RAG для создания чат-бота помощника для студентов вуза.**

Направленность КР _____

(учебная, исследовательская, практическая, производственная и др.)

Источник тематики _____

График выполнения работы: 25% к __ нед., 50% к __ нед., 75% к __ нед., 100% к __ нед.

Задание _____

Оформление курсовой работы:

Расчетно-пояснительная записка на ____ листах формата А4.

Перечень графического (иллюстративного) материала _____

Дата выдачи задания «____» _____ 20___ г.

Руководитель курсовой работы _____ *Д.Н. Федянин*

(подпись, дата)

Студент _____ *Н.Ю.Яшин*

(подпись, дата)

Примечание. Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Содержание

1 Введение 4
2 Формальное описание задачи 5
3 Retrieval модель 6
4 Сбор данных 8
5 Оценка качества Retrieval 9
6 Augmented-Generation 10
7 Оценка работы генератора 12
8 Демонстрация работы всей системы 13
9 Заключение 15
Список литературы 16

Постановка задачи

1. Собрать и подготовить данные для задачи.
2. Создать модель Retrieval
3. Реализовать Augmentated Generation.
4. Оценить качество генерации.

1 Введение

В последние годы наблюдается бурное развитие методов искусственного интеллекта для обработки текстовых данных. Одним из самых значительных достижений можно считать создание Больших языковых моделей (Large Language Models, LLM), которые основываются на механизме внимания [1]. Эти модели содержат миллиарды обучаемых параметров, что и делает их "большими"по сравнению с традиционными нейросетями.

Такие модели позволяют решать широкий спектр задач:

- Текстовая классификация
- Генерация текста
- Ответы на вопросы
- Обобщение текста
- Перевод текста
- Извлечение информации
- Диалоговые системы
- Автоматизация написания кода
- Обработка юридических документов
- Моделирование диалектов и разговорной речи

В рамках данной курсовой работы основное внимание будет уделено задаче ответов на вопросы. В первоначальной реализации большие языковые модели способны эффективно хранить информацию внутри своих параметров и отвечать правильно на большинство вопросов.

Однако возникают сложности, когда моделям необходимо использовать информацию, которую они не видели в процессе обучения. Это может быть как актуальная информация (например, вопрос "Какой сейчас курс доллара?"), так и данные, связанные с функционированием внутренних сервисов компании.

Для решения этой проблемы был разработан механизм Retrieval Augmented Generation (RAG), который позволяет моделям отвечать на вопросы, "подсматривая"информацию из внешних источников.

2 Формальное описание задачи

Назовём множество $\mathcal{V} = \{t_1, t_2, \dots t_N\}$ - **словарём**. Его элементы будут называться **токенами**.
На практике за токены можно обозначать различные части естественного языка: цельные слова, морфемы, знаки препинания, специальные токены (токен окончания текста, токен переноса строки и т.д.).

Элементы множества $\mathcal{T} = \{(t_1, t_2, \dots, t_k) \mid t_i \in \mathcal{V},\ \ k \geq 1\}$ назовём **текстами**.

Теперь мы можем описать, как работает LLM модель:
Изначально строится распределение вероятностей для токенов из словаря: $P(t|t_1, t_2, \dots t_k, \theta)$. Здесь θ — это параметры модели, которые задают её архитектуру. Это распределение показывает, какова вероятность следующего токена, если k токенов уже есть.

Для выбора этого токена можно придумать множество способов, но самыми популярными являются *Top-p* и *Top-k*.

Для Top-p способа строится подмножество:

$$\mathcal{V}_p = \{t \in \mathcal{V} | P(t|t_1, t_2, \dots, t_k, \theta) > p\}$$

а для Top-k способа:

$$\mathcal{V}_k = \{t \in \mathcal{V} | rank(P(t|t_1, t_2, \dots, t_k, \theta)) \leq k\}$$

где $rank(s_i)$ - это позиция элемента s_i в отсортированном множестве S.

Далее токен t_{k+1} выбирается случайно:

$$t_{k+1} \sim P(t|t_1, \dots t_k,), t \in V_p$$

Если объединить этот алгоритм воедино, то **большая языковая модель** - это просто функция, которая подбирает следующий токен, чтобы наиболее логично продолжить текст: $t_{k+1} = LLM(t_1, t_2, \dots, t_k)$, где $t_i \in \mathcal{V}$.

Пример использования:

$$\text{LLM}(\texttt{'сегодня'}, \texttt{'я'}, \texttt{'ходил'}, \texttt{'на'}) = \texttt{'работу'}$$

Вообще говоря, этот процесс можем повторять рекурсивно, если "засунуть"полученный токен на вход для следующего прохода:

$$t_{k+2} = LLM(t_1, \dots, t_k, LLM(t_1, \dots, t_k))$$

обычно этот процесс обрывают, когда встречается специальный токен t_{END}.

Таким образом, мы плавно продвигаемся к понятию *Instruct LLM*. Если обычные LLM умеют только хорошо дополнять текст, то инструктивные модели уже могут явно выполнять инструкцию пользователя или отвечать на его вопрос. Такая инструкция называется **промптом**: $prompt \in \mathcal{T}$.

А **Instruct LLM** можно определить как функцию: $answer = ILLM(prompt)$, где $answer \in \mathcal{T}$.

Это различие достигается за счёт данных, на которых модели обучаются. Для обычной LLM нужно подготовить датасет из различных текстов, и при обучении модель будет токен за токеном генерировать эти тексты. Датасет для Instruct LLM надо собирать парами: $DataSet = \{(\text{prompt}_i, \text{answer}_i) \in \mathcal{T}^2\}$. И во время обучения модель будет учиться генерировать ответ по заданной инструкции. От качества и правильности этого датасета напрямую зависит качество работы модели.

Как обсуждалось ранее, проблемы с ответом на вопрос возникают, когда ответ на промпт подразумевает «знания», которых в обучающем датасете не было. В таком случае помогает **Augmented Generation**. В этом подходе мы добавляем для ответа на вопрос некий контекст. **Контекст** (context $\in \mathcal{T}$) - это текст, найденный во внешней базе данных, который по смыслу должен быть похожим на промпт. Поиском наиболее подходящего контекста занимается модель **Retrieval**.

Получаем, что $AugLLM : \mathcal{T}^2 \to \mathcal{T}; AugLLM(prompt, context) = answer$.
Рассмотрим подробнее, как работает модель Retrieval и как дообучать модель AugLLM.

3 Retrieval модель

Для того, чтобы модель умела искать наиболее релевантную информацию, сначала надо обсудить, какая у нас есть информация.
Определим множество **документов** - $D \subset \mathcal{T}$. На практике это могут быть различные внутренние файлы компании, текст с сайта, учебники и так далее.

Но обычно информация, нужная для ответа на вопрос, лежит в какой-то небольшой части всего документа. Поэтому появляется идея разделить весь текст на небольшие кусочки, но достаточного размера для хранения информации.

Рассмотрим $d \in D, d_i = (t_1, t_2, \ldots, t_N)$. Элементы

$$C_d = \begin{cases} ch_1 = (t_1, t_2, \ldots, t_m) \\ ch_2 = (t_{m+1}, t_{m+2}, \ldots, t_{2m}) \\ \ldots \\ ch_{\frac{N}{m}} = (t_{N-m+1}, t_{N-m+2}, \ldots, t_N) \end{cases}$$

назовём **чанками**. Параметр m - размер чанков, можно фиксировать любым числом. В нашем случае возьмём равным 100.

Но на самом деле, если задавать чанки таким образом, то может возникнуть ситуация, когда нужная информация в документе будет разделена двумя чанками. Эту проблему можно решить, если сделать небольшое наложение чанков:

$$ch_i = (t_{m(i-1)+1}, t_{m(i-1)+2}, \ldots, t_{mi}, t_{mi+1}, \ldots, t_{mi+k})$$

Этот способ деления на чанки хорошо подходит, когда документ представлен единым большим текстом. Также существует другой способ составить чанки, который особенно подходит для извлечения информации с сайтов: использовать параграфы как отдельные чанки, которые на языке разметки HTML выделены с помощью специальных символов `<p> text </p>`.

Чанки можно сохранить в массив сразу после получения документов и работать уже с ними: $C = \bigcup_{d \in D} C_d$

Теперь перейдём к тому, как будем искать чанк с наиболее релевантной информацией. Для этого будем использовать векторизацию текста. Эти вектора называются **эмбеддингами**, и должны быть близкими для близких по смыслу текстов.

Наиболее популярным сейчас способом является использование архитектуры BERT (Bidirectional Encoder Representations from Transformers) [2]. Данные модели используют механизм внимания: для каждого токена из текста находится «важность» всех других токенов, и итоговый вектор агрегирует все векторы с коэффициентами важности. Поэтому на выходе получается эмбеддинг, который «понимает» весь текст.

$$\text{BERT} : \mathcal{T} \to \mathbb{R}^m$$

размерность выхода берта m задаётся на этапе создания архитектуры любым целым числом. В работе я использовал предобученный берт с параметром $m = 312$.

Близкие по смыслу тексты оказываются близкими друг к другу в пространстве $\mathbb{R}^m$, а у близких векторов близкий угол между ними. Поэтому целесообразно использовать косинусное расстояние: $CosDist(a, b) = 1 - \frac{(a,b)}{\|a\|\|b\|}$.

И таким образом, Retrieval модель работает следующим образом:

$$Retrieval(prompt, C) = \arg\min_{ch \in C} CosDist(\text{BERT}(prompt), \text{BERT}(ch))$$

Приведу код реализации модели:

```
import numpy as np; import pandas as pd

def cos_distance(e1, e2):
    return 1 - dot(e1, e2)/np.linalg.norm(e1)/np.linalg.norm(e2)

class Retrieval:
    def __init__(self, chunks, model, tokenizer):
        if isinstance(chunks, pd.DataFrame):
            self.chunks = list(chunks["chunks"])
        elif isinstance(chunks, list):
            self.chunks = chunks
        else:
            raise TypeError('чанки должны быть или списком или датафреймом')
        self.model = model
        self.tokenizer = tokenizer
        self.chunk_vecs = []
        for chunk in self.chunks:
            self.chunk_vecs.append(list(embed_bert(chunk, model, tokenizer)))
        self.chunk_vecs = np.array(self.chunk_vecs)

    def get_chunks(self):
        return self.chunks

    def find_context(self, question):
        q_emb = embed_bert(question, self.model, self.tokenizer)
        min_ind = 0
        distances = []
        for i in range(len(self.chunks)):
```

```
            distances.append(cos_distance(q_emb, self.chunk_vecs[i]))
        min_ind = np.argmin(distances)

        return self.chunks[min_ind], distances[min_ind], min_ind, distances

    def find_top_k_context(self, question, k):
        q_emb = embed_bert(question, self.model, self.tokenizer)
        min_ind = 0
        distances = []
        for i in range(len(self.chunks)):
            distances.append((cos_distance(q_emb, self.chunk_vecs[i]), i))

        distances.sort(reverse = False, key = lambda x: x[0])
        distances = distances[:k]
        indexes = list(map(lambda x: x[1], distances))
        dist = list(map(lambda x: x[0], distances))
        return indexes, dist
```

4 Сбор данных

Для курсовой работы я взял два документа:
Страница Википедии про МГТУ имени Н.Э. Баумана
FAQ на сайте VK Park
Сайт с графиком работы и часами приёма различных учреждений
Суммарно получилось собрать 74 чанков информации.

Далее, для того чтобы оценить модель Retrieval и для того чтобы дообучить LLM, нужно для каждого чанка составить релевантный вопрос и ответ на этот вопрос. Для этого будем использовать синтетические данные.
Синтетические данные для задач NLP (Natural Language Processing) — это искусственно созданные данные, которые имитируют реальные текстовые данные, но не содержат информации, полученной из реальных источников. Для создания таких данных я воспользуюсь генеративной моделью от Яндекса: YandexGPT.

Для генерации данных я использовал температуру 0.3. Температура в ЛЛМ регулирует случайность в процессе генерации текста. Низкая температура (например, 0.3) делает модель более детерминированной, что приводит к более предсказуемым результатам. Высокая температура (например, 1.0 и выше) увеличивает разнообразие, делая ответы более случайными и креативными, но иногда менее точными.
И я использовал следующий промпт для генерации:

Промпт

Я готовлю синтетические данные для РАГ. Придумай вопрос и ответ для чанка. Вопрос должен быть похож на человеческий, как будто его задаёт студент. Также вопрос должен содержать информацию, которая есть в чанке. Раздели вопрос и ответ с помощью —.

Пример получившихся данных:

Пример

chunk: «Размер стипендии составляет 15 000 рублей в месяцСтипендия от VK начисляется ежемесячно в течении 4 месяцев (сентябрь - декабрь 2023 для осеннего семестра; февраль 2024 - май 2024 для весеннего семестра)»,
question: «Сколько месяцев будет выплачиваться стипендия от VK и какой размер этой стипендии? »,
answer: «Стипендия от VK начисляется ежемесячно в течение 4 месяцев: с сентября по декабрь 2023 года для осеннего семестра и с февраля по май 2024 года для весеннего семестра. Размер стипендии составляет 15 000 рублей в месяц.»

5 Оценка качества Retrieval

Для того, чтобы оценить качество модели, можно посмотреть на задачу поиска нужного чанка как на задачу рекомендации и использовать соответствующие метрики. [3] В качестве запроса будет использоваться вопрос пользователя, а в качестве объектов будут использоваться чанки. Но особенность нашего датасета в том, что для каждого вопроса есть только 1 релевантный чанк.
Перечислим популярные метрики:

Recall(полнота):

$$\text{Recall@k} = \frac{\text{Количество релевантных чанков среди топ-k}}{\text{Общее количество релевантных чанков в датасете}}$$

Mean Reciprocal Rank (MRR):

$$\text{MRR} = \frac{1}{|Q|}\sum_{q\in Q}\frac{1}{\text{rank}_q}$$

Mean Rank (MR):

$$\text{MR} = \frac{1}{|Q|}\sum_{q\in Q}\text{rank}_q$$

здесь rank_q означает номер релевантного чанка q в выдаче

Normalized discounted cumulative gain (nDCG):

$$\text{NDCG@k} = \frac{\text{DCG@k}}{\text{IDCG@k}}$$

где

$$\text{DCG@k} = \sum_{i=1}^{k}\frac{rel_i}{\log_2(i+1)}$$

$$\text{IDCG@k} = \sum_{i=1}^{k}\frac{1}{\log_2(i+1)}.$$

здесь $rel_i = \begin{cases} 1, \text{если i-ый чанк релевантен} \\ 0, \text{иначе} \end{cases}$

Усреднённые результаты метрик для всех запросов:

- **Средний Recall@1**: 0.638
- **Средний Recall@3**: 0.809
- **Средний Recall@5**: 0.851
- **Средний MRR**: 0.731
- **Средний Mean Rank**: 3.064
- **Средний NDCG@1**: 0.638
- **Средний NDCG@3**: 0.732
- **Средний NDCG@5**: 0.749

Получаем, что примерно в 64% случаях релевантный чанк находится в топ-1 выдачи. В топ-3 находятся 81%, а в топ-5 85%. Также релевантный чанк в среднем находится на 3-м месте в выдаче.
Получаем, что нет сильной разницы между топ-3 и топ-5 выдачей. А вот между топ-1 и топ-3 выдачей есть. Поэтому для формирования контекста можно объединить топ-3 выдачи:

$$\text{context} = Concat(\text{chunk}_{(1)}, \text{chunk}_{(2)}, \text{chunk}_{(3)})$$

6 Augmented-Generation

Для курсовой работы мы возьмём небольшую LLM на 1 миллиард параметров, предобученную на датасете с русским языком. [4]

Насколько мы помним из введения, обычная LLM умеет только продолжать текст. Поэтому надо на вход подавать тексты определённого, зафиксированного вида, где вопрос, контекст и ответ будут разделены. Это разделение достигается с помощью специальных токенов <|eot_id|>, <|start_header_id|> и <|end_header_id|>.

Структура текстовых данных

<|begin_of_text|><|start_header_id|>system<|end_header_id|>
<|eot_id|><|start_header_id|>user<|end_header_id|>
context
<|eot_id|><|start_header_id|>user<|end_header_id|>
question<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
answer
<|eot_id|>

Для всей РАГ системы я реализовал класс на языке программирования *Python*. Объект класса получает вопрос посредством функции RAG.answer(prompt). При получении вопроса пользователя РАГ система сначала находит топ-3 самых релевантных чанка, объединяет их и отвечает на вопрос, исходя из контекста.

```python
class RAG:
    def __init__(self, retr: Retrieval, model, tokenizer):
        self.retr = retr
        self.model_llm = model
        self.tokenizer_llm = tokenizer

    def _get_answer(self, prompt: str, context: str, model, tokenizer,
        tokens_number = 512, temp = 0.3, top_k = 50, top_p = 0.95):

        input_ids = tokenizer.apply_chat_template([{"role": "user",
                                                    "content": context},
                                                       {"role": "user",
                                                       "content": prompt}],
                                                     truncation=True,
                                                  add_generation_prompt=False,
                                                padding=False,
                                                return_tensors="pt").to("cuda")
        with torch.no_grad():

            outputs = model.generate(
                    input_ids=input_ids,
                    max_new_tokens=tokens_number,
                    do_sample=True,
                    temperature=temp,
                    top_k=top_k,
                    top_p=top_p,
                    pad_token_id=tokenizer.pad_token_id
            )

        decoded_output=tokenizer.batch_decode(outputs,skip_special_tokens=False)[0]

        ans = decoded_output.split('<|start_header_id|>assistant<|end_header_id|>')
        answer = ans[-1].replace('<|eot_id|>', '').strip()
        return answer

    def answer(self, prompt: str):
        indexes = self.retr.find_top_k_context(prompt,3)[0]
        context = ''
        for ind in indexes:
            context+=self.retr.chunks[ind]
            context+='\n\n'

        return self._get_answer(prompt, context, self.model_llm, self.tokenizer_llm)
```

7 Оценка работы генератора

В отличие от традиционных методов машинного обучения, где есть четко определенные количественные метрики (такие как Gini, R-квадрат, AIC, BIC, матрица неточностей и т. д.), оценка систем RAG более сложна. Эта сложность возникает из-за того, что ответы, генерируемые системами RAG, представляют собой неструктурированный текст, требующий сочетания качественных и количественных метрик для точной оценки их производительности.

BLEU (Bilingual Evaluation Understudy). BLEU измеряет перекрытие между сгенерированным ответом и набором эталонных ответов, уделяя особое внимание точности n-грамм. Он рассчитывается путем измерения перекрытия n-грамм (непрерывных последовательностей из n-слов) между сгенерированными и эталонными ответами. Формула для оценки BLEU следующая:

$$\text{BLEU} = \text{BP} \cdot \exp(\sum_{n=1}^{N} w_n \log P_n)$$

где (BP) — штраф за краткость для штрафования коротких ответов, (P_n) — точность n-грамм, а (w_n) — веса для каждого уровня n-грамм. BLEU количественно измеряет, насколько близко сгенерированный ответ соответствует эталонному ответу.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation). ROUGE измеряет перекрытие n-грамм, последовательностей слов и пар слов между сгенерированными и эталонными ответами, учитывая как полноту, так и точность. Наиболее распространённый вариант ROUGE-N измеряет перекрытие n-грамм между сгенерированными и эталонными ответами. Формула для ROUGE-N следующая:

$$\text{ROUGE-N} = \frac{\sum_{i=1}^{N} \text{Count}_{\text{match}}(n-i)}{\sum_{i=1}^{N} \text{Count}_{\text{ref}}(n-i)}$$

ROUGE оценивает как точность, так и полноту, предоставляя сбалансированную меру того, сколько релевантного контента из ссылки присутствует в сгенерированном ответе.

METEOR (Metric for Evaluation of Translation with Explicit ORdering). METEOR учитывает синонимику, происхождение и порядок слов для оценки сходства между сгенерированным и эталонными ответами. Формула для оценки METEOR следующая:

$$\text{METEOR} = F_{mean} \cdot (1 - \beta P_{penalty})$$

где F_{mean} — это гармоническое среднее значение точности и полноты, а $P_{penalty}$ — это штраф за неправильный порядок слов и другие ошибки. METEOR обеспечивает более тонкую оценку, чем BLEU или ROUGE, поскольку учитывает синонимы и происхождение.

Результаты измерения метрик на моём датасете из 47 вопросов и ответов:

- **Средний BLEU**: 0.16

- **Средний ROUGE**: 0.48
- **Средний METEOR**: 0.33

Результаты метрик получились неоднозначными. Значение BLEU оказалось низким, а METEOR уже в два раза выше. Разница между ними в том, что METEOR учитывает не только точные совпадения n-грамм, но и семантические совпадения (например, синонимы) и порядок слов.
ROUGE измеряет пересечения униграмм (отдельных слов) между сгенерированным текстом и эталонным ответом. Значение 0.48 указывает на то, что почти половина слов в вашей генерации совпадает с эталонным ответом.

Вывод из значений метрик можно сделать такой, что РАГ система довольно точно отвечает на короткие вопросы (метрика ROUGE). А в длинных вопросах по смыслу отвечает неплохо, но объясняет другими словами (например, синонимами).

8 Демонстрация работы всей системы

Для демонстрации работы РАГ системы, сравним её с самой популярной на текущий момент большой языковой моделью - ChatGPT.

Эта модель обучалась на огромном датасете, в том числе на статьях из Википедии. Также у ЧатГПТ встроена своя система поиска и извлечения информации из интернета. Однако она не всегда умеет точно находить и использовать специфическую информацию.

Ниже приведу несколько примеров потенциально возможных вопросов студента или абитуриента вуза, на которые ChatGPT не смог дать точный ответ. И в противовес продемонстрирую результат работы РАГ системы, которая смогла найти нужную информацию.

Вопрос	**RAG Система**	**ChatGPT**
Какие языки программирования изучают на первом семестре в VK технопарке?	На первом семестре в VK технопарке, изучаются C++ и Python.	В VK Технопарке студенты изучают различные языки программирования в зависимости от выбранной образовательной программы. Например, в подготовительном курсе по C++ рассматриваются основы работы с памятью, обработка исключений и базовые навыки обобщённого программирования. В рамках курса «Языки вебпрограммирования» от МГТУ им. Н.Э. Баумана используется язык Ruby. Таким образом, выбор языка программирования зависит от конкретной программы обучения в VK Технопарке.
Сколько месяцев будет выплачиваться стипендия от VK и какой размер стипендии?	Стипендия от VK выплачивается ежемесячно в течение 4 месяцев. Стоимость стипендии составляет 15 000 рублей в месяц.	В VK Технопарке предусмотрены стипендии для студентов, однако точная информация о размере и продолжительности выплат не указана в доступных источниках. Рекомендуется обратиться напрямую в VK Технопарк для получения актуальной информации о стипендиях.

Таблица 1: Сравнение ответов моей RAG системы и ChatGPT.

На втором примере видно, что иногда ChatGPT может сомневаться и просить самому проверить информацию. Это происходит, когда модель не смогла найти точную информацию в интернете.

А в первом примере моя система ответила более кратко, но при этом взяла более точную информацию с сайта FAQ VK технопарк. А в ответе ChatGPT не был упомянут язык *Python*.

9 Заключение

В данной курсовой работе была рассмотрена структура и принципы работы системы Retrieval-Augmented Generation (РАГ), которая сочетает в себе извлечение релевантных данных и генерацию ответов на основе этих данных с использованием языковых моделей. Целью работы было создание прототипа системы, способной эффективно извлекать информацию и генерировать осмысленные ответы на вопросы пользователей.

В процессе работы были успешно решены следующие задачи:

- **Сбор и подготовка данных**: Для обучения системы были взяты несколько сайтов нашего ВУЗа, содержащие общую информацию про ВУЗ, информацию про VK технопарк, а также информацию про график работы и расположение учреждений. На основе чанков из этих сайтов были сгенерированны потцениальные вопросы и ответы на эти вопросы.

- **Создание модели Retrieval**: Для извлечения релевантных фрагментов текста был разработан компонент извлечения, который использует модель на основе BERT для генерации текстовых эмбеддингов. Модель успешно находит самые подходящие фрагменты текста в базе данных, что является основой для последующей генерации ответа.

- **Реализация Augmented Generation**: На основе извлеченных фрагментов текста была реализована генерация ответов с использованием ЛЛМ, что позволяет системе эффективно комбинировать извлеченную информацию с генеративной моделью для создания содержательных и релевантных ответов на вопросы.

- **Оценка качества модели**: Для оценки качества извлечения информации главной метрикой было Precision@3, которая показала результаты на уровне 85%. Этот показатель свидетельствует о высокой степени релевантности поиска информации, а в последствии и генерации правильного ответа.

Итоговая версия системы достаточно успешно решает задачу понимания вопросов студента, извлечения нужной информации и генерации на основе этой информации ответов. Модель демонстрирует высокое качество работы на тестовых данных, что подтверждается положительными результатами оценки. Однако для дальнейшего улучшения системы можно продолжить работу над увеличением разнообразия данных, улучшением модели Retrieval и доработкой алгоритмов генерации, что позволит значительно повысить точность и релевантность ответов.

В целом, работа показала, что использование системы Retrieval-Augmented Generation имеет потенциал для решения задач поиска и генерации текста в ответ на вопросы, и продемонстрировала реальный практический результат, который можно использовать в дальнейших исследованиях и проектах.

Список литературы

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., Polosukhin, I. Attention is All You Need // Advances in Neural Information Processing Systems (NIPS). — 2017. - С. 2-8. URL: https://arxiv.org/abs/1706.03762 (дата обращения 29.11.2024)

2. Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova // BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. - 2018. - С. 1-7. URL: https://arxiv.org/abs/1810.04805 (дата обращения 29.11.2024)

3. Cañamares R., Castells P., Moffat A. Offline evaluation options for recommender systems // Information Retrieval Journal. – 2020. – Т. 23. – №. – С. 387-410.

4. URL: https://huggingface.co/Vikhrmodels/Vikhr-Llama-3.2-1B-Instruct (дата обращения 20.12.2024)