1 Почему именно генерация с дополненной выборкой (RAG)

Даже самые продвинутые модели генеративного ИИ могут генерировать ответы на основе лишь тех данных, на которых они обучались. Они не способны давать точные ответы на вопросы по теме, выходящей за рамки обучающего датасета. Модели генеративного ИИ просто не знают, что они чего-то не знают. Такой вот каламбур. Это ведет к неточным или некорректным результатам, иногда называемым галлюцинациями или смещением, проще говоря, чепухе.

Поисково-дополненная генерация, или *генерация с дополненной выборкой* (Retrieval Augmented Generation, RAG), — это механизм, который компенсирует названное ограничение путем расширения генеративных моделей техниками извлечения информации. Его работа заключается в извлечении релевантных данных из внешних источников в реальном времени и использовании этих данных для генерации более точных и соответствующих контексту ответов. Модели генеративного ИИ, совмещенные с механизмами поиска[1] RAG, революционизируют эту сферу своей беспрецедентной эффективностью и возможностями. Одно из ключевых преимуществ RAG — ее адаптируемость. Этот механизм можно с легкостью применить к любым типам данных, будь то текст, изображения или аудио. Такая гибкость делает экосистему RAG надежным и эффективным инструментом для расширения возможностей генеративных ИИ.

Однако руководителям проектов уже известно множество платформ, фреймворков и моделей генеративного ИИ, например Hugging Face, Google Vertex AI, OpenAI, LangChain и др. Поэтому дополнительный уровень в виде появляющихся фреймворков и платформ RAG наподобие Pinecone, Chroma, Activeloop, LlamaIndex и пр. только добавляет сложности. Все эти фреймворки генеративного ИИ и RAG зачастую пересекаются, создавая невероятное множество возможных конфигураций. В итоге руководителю проекта бывает непросто найти правильную конфигурацию моделей и ресурсов RAG для конкретного проекта.

---

[1] Ретриверами (от англ. retriever).

Универсального решения здесь нет. Задача действительно серьезная, но, если ее решить, это принесет огромные дивиденды.

Мы начнем эту главу с общего рассмотрения механизма RAG. Затем определим три основные конфигурации этой системы: простейшую, продвинутую и модульную RAG. Помимо этого, сравним RAG с тонкой настройкой и определим, когда и какой из этих подходов использовать. RAG может существовать только внутри экосистемы, которую мы в текущей главе спроектируем и опишем. Данные должны откуда-то поступать и где-то обрабатываться. Для их поиска и извлечения необходима организованная среда, а модели генеративного ИИ имеют свои ограничения на ввод.

Наконец, в этой главе мы разберем и более практический аспект, а именно создадим на Python программу для выполнения простейшей RAG на базе поиска и сопоставления ключевых слов. Кроме того, напишем продвинутую систему RAG с векторным поиском и извлечением по индексу. А в завершение создадим модульную RAG, которая будет включать в себя простейшую и продвинутую модели механизма. К концу главы вы будете не только знать стоящую за RAG теорию, но и обладать опытом создания генеративного ИИ на ее основе. Такой подход позволит вам лучше понять тему и подготовит к прочтению последующих глав.

Что такое RAG

Когда модель генеративного ИИ не может дать точного ответа, говорится, что она галлюцинирует или дает искаженный результат (bias). Если проще, то она выдает чушь. Тем не менее все сводится к невозможности дать адекватный ответ, что может быть вызвано как классическими проблемами с настройкой модели, так и тем, что процесс ее обучения просто не включал запрошенную информацию. Это часто ведет к выдаче случайной череды наиболее вероятных, а не наиболее достоверных результатов.

И начинается RAG там, где генеративный ИИ предоставляет информацию, на которую модель LLM не может ответить точно. Этот принцип в качестве дополнения LLM в 2020 году разработал специалист по ИИ Льюис (Lewis) в сотрудничестве с коллегами. Механизм RAG выполняет оптимизированные задачи по извлечению информации, а система генерации добавляет ее к вводу (запросу пользователя или автоматизированному промту) для выдачи более точного результата. Схематично RAG можно описать, применяя аналогию, изображенную на рис. 1.1.

Представьте, что вы студент в библиотеке и вам нужно составить реферат с помощью RAG. Аналогично ChatGPT или другому ИИ-ассистенту вы умеете читать и писать. Как и любая *большая языковая модель*, вы достаточно образованны для того, чтобы изучить расширенную информацию, обобщить ее и записать нужное. Однако, как и любой ИИ, созданный компаниями Hugging Face, Certex AI или OpenAI, многого по теме вы не знаете.

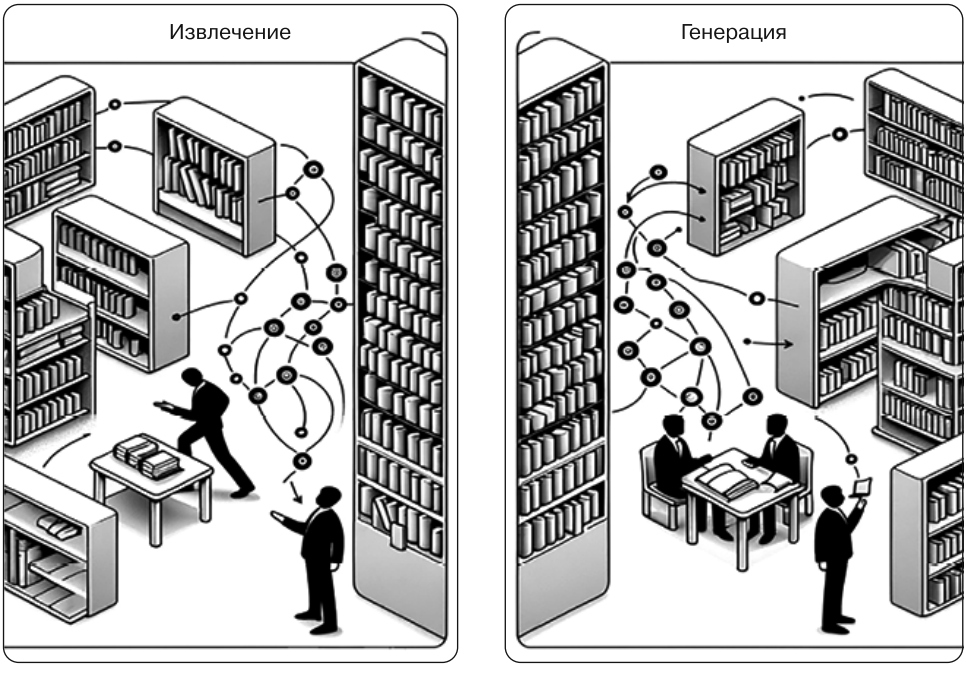

Рис. 1.1. Два новых компонента генеративного ИИ на базе RAG

На этапе *извлечения* вы изучаете библиотеку в поиске книг по интересующей теме (см. рис. 1.1, *слева*). Затем возвращаетесь за свой стол, находите в собранных книгах нужную информацию сами или поручаете поиск помощнику и извлекаете ее. На этапе *генерации* (см. рис. 1.1, *справа*) приступаете к написанию реферата. В описанной схеме вы выступаете в роли генеративного агента-человека, использующего RAG, то есть действуете аналогично генеративному фреймворку ИИ на базе RAG.

В процессе написания реферата вы сталкиваетесь с рядом сложных тем, но у вас нет времени на то, чтобы физически изучить всю эту информацию. Вы, как генеративный человек-агент, оказываетесь в тупике, в каком оказалась бы и модель генеративного ИИ. Тогда можете попробовать написать хоть что-то, как это и делает модель, когда выдает некорректный результат. При этом вы, как и модель ИИ, не поймете, где написанное вами правда, а где — нет, пока кто-нибудь не скорректирует реферат и не присвоит ему оценку.

На этом этапе вы достигли своего предела и решаете вернуться к ИИ-ассистенту на базе RAG, чтобы убедиться в корректности ответов. Но оказываетесь озадачены большим числом доступных LLM-моделей и их конфигураций. Для начала потребуется разобраться, какие ресурсы вам доступны и как вообще организована RAG. Далее мы пройдемся по основным видам конфигурации LLM.

Простая, продвинутая и модульная конфигурации RAG

Фреймворк RAG всегда включает в себя два основных компонента: модуль извлечения информации (ретривер) и генератор результата. Генератором может выступать любая LLM, мультимодальная платформа ИИ либо модель, например GPT-4o, Gemini, Llama или одна из сотен вариаций базовых архитектур. Ретривером же может быть любой из новейших фреймворков, методов и инструментов, например Activeloop, Pinecone, LlamaIndex, LangChain, Chroma и многие другие.

Выбор одного из трех типов RAG (Gao et al., 2024) будет зависеть от задач проекта. Все эти три подхода мы разберем в коде текущей главы.

- *Простая RAG*. Не подразумевает сложного встраивания данных и индексирования. Ее можно эффективно использовать для обработки адекватных объемов данных с помощью ключевых слов, например, для дополнения запроса пользователя и получения желаемого ответа.
- *Продвинутая RAG*. Такие механизмы способны обрабатывать более сложные сценарии с применением векторного поиска и извлечения информации на основе индексирования. Продвинутую RAG можно реализовать множеством разных способов. Этот фреймворк сможет обрабатывать различные типы данных, в том числе мультимодальных, которые бывают структурированными или неструктурированными.
- *Модульная RAG*. Модульные решения расширяют горизонт возможностей, включая сценарии, где задействованы простая RAG, продвинутая RAG, машинное обучение и любой другой алгоритм, необходимый для реализации комплексного проекта.

Но прежде чем идти дальше, нам нужно решить, будем ли мы реализовывать именно RAG или же тонко настраивать (fine-tune) модель.

RAG или тонкая настройка?

RAG не всегда может заменить тонкую настройку, как и тонкая настройка не всегда заменит RAG. Если мы соберем в датасетах RAG слишком много данных, системой станет очень сложно управлять. В то же время мы не можем тонко настроить модель с динамическими, постоянно меняющимися данными, такими как прогнозы погоды, стоимость ценных бумаг, корпоративные новости и всевозможные виды каждодневных событий.

Выбор между созданием RAG и тонкой настройкой модели будет зависеть от конкретной пропорции параметрической и непараметрической информации. Фундаментальное различие между моделью, которая обучалась с нуля или

прошла тонкую настройку, и RAG можно обобщить, опираясь на объем их параметрических и непараметрических знаний.

- *Параметрические.* В экосистеме генеративных ИИ на базе RAG параметрический аспект относится к параметрам (весам) модели, усвоенным из обучающих данных. Это означает, что знания модели хранятся в «выученных» весах и смещениях. Изначальные обучающие данные преобразуются в математическую форму, которую мы называем параметрическим представлением. По сути, модель запоминает то, что выучила из этих данных, но сами данные явно не сохраняет.
- *Непараметрические.* И наоборот, непараметрический аспект экосистемы RAG связан с сохранением явных данных, к которым можно обращаться напрямую. Это означает, что данные остаются доступными и при необходимости их можно запрашивать. В отличие от параметрических моделей, где знания усваиваются косвенно в виде весов, непараметрические данные в RAG позволяют видеть и использовать для вывода фактическую информацию.

Выбор между RAG и тонкой настройкой основывается на количестве статических (параметрических) и динамических (непараметрических) данных, которые модели ИИ нужно обрабатывать. Система, которая излишне опирается на RAG, со временем может стать громоздкой и неудобной в управлении. Если же система чрезмерно опирается на тонкую настройку модели, она будет отличаться неспособностью адаптироваться к ежедневному обновлению информации.

На рис. 1.2 показана схема принятия решения на основе определенного порога. Опираясь на этот порог, руководителю проекта генеративного ИИ на базе RAG нужно оценить потенциал параметрической модели ИИ, прежде чем реализовывать непараметрический (с явными данными) механизм RAG. При этом потенциал компонента RAG также требует внимательной оценки.

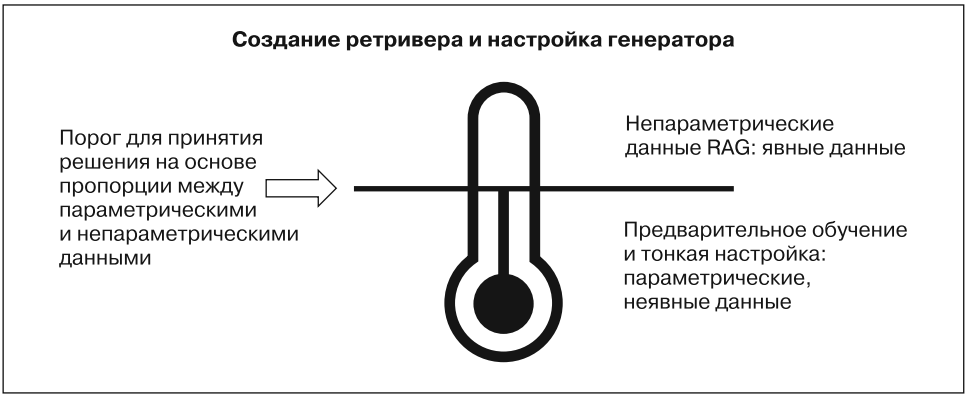

Рис. 1.2. Порог для принятия решения о выборе между расширением датасета RAG и настройкой LLM

Баланс между расширением ретривера и генератора в системе генеративного ИИ на базе RAG зависит от конкретных требований и целей проекта. RAG и тонкая настройка не являются взаимоисключающими.

Для повышения общей эффективности модели можно использовать RAG, дополняя ее тонкой настройкой, которая позволяет улучшить качество как извлечения информации, так и генерации ответа механизмом RAG. Настройкой датасета RAG мы займемся в главе 9.

Сейчас же рассмотрим то множество компонентов, которые включает в себя экосистема генеративного ИИ на базе RAG.

Экосистема RAG

Генеративный ИИ на базе RAG — это фреймворк, который можно реализовать в разных конфигурациях. На рис. 1.3 показана обширная система, внутри которой работает этот фреймворк. Но сколько бы фреймворков извлечения информации или генерации ответов вы ни встречали, их все можно разделить на четыре основные области, сопровождаемые вопросами.

- *Данные*. Откуда берутся данные? Надежны ли они? Достаточно ли этих данных? Связаны ли они с авторскими правами, конфиденциальностью или вопросами безопасности?
- *Хранилище*. Как данные будут храниться до или после их обработки? Какой объем данных будет сохранен?
- *Извлечение*. Как нужные данные будут извлекаться для дополнения ввода пользователя достаточным для генеративной модели объемом информации? Какой вид фреймворка RAG окажется оптимальным для проекта?
- *Генерация*. Какая модель генеративного ИИ впишется в выбранный фреймворк RAG?

Сегменты данных, хранилища и генерации во многом определяются выбранным механизмом RAG. Прежде чем делать этот выбор, необходимо оценить пропорцию параметрических и непараметрических данных в создаваемой экосистеме. На рис. 1.3 показан механизм RAG, включающий перечисленные компоненты без учета реализуемого вида RAG.

- *Ретривер (D)* отвечает за сбор, обработку, хранение и извлечение данных.
- *Генератор (G)* отвечает за дополнение ввода, промт-инжиниринг и генерацию результата.
- *Модуль оценки (E)* обрабатывает вычисление метрик, оценку человеком и обратную связь от него.
- *Учитель (T)* обрабатывает изначально обученную модель и выполняет ее тонкую настройку.

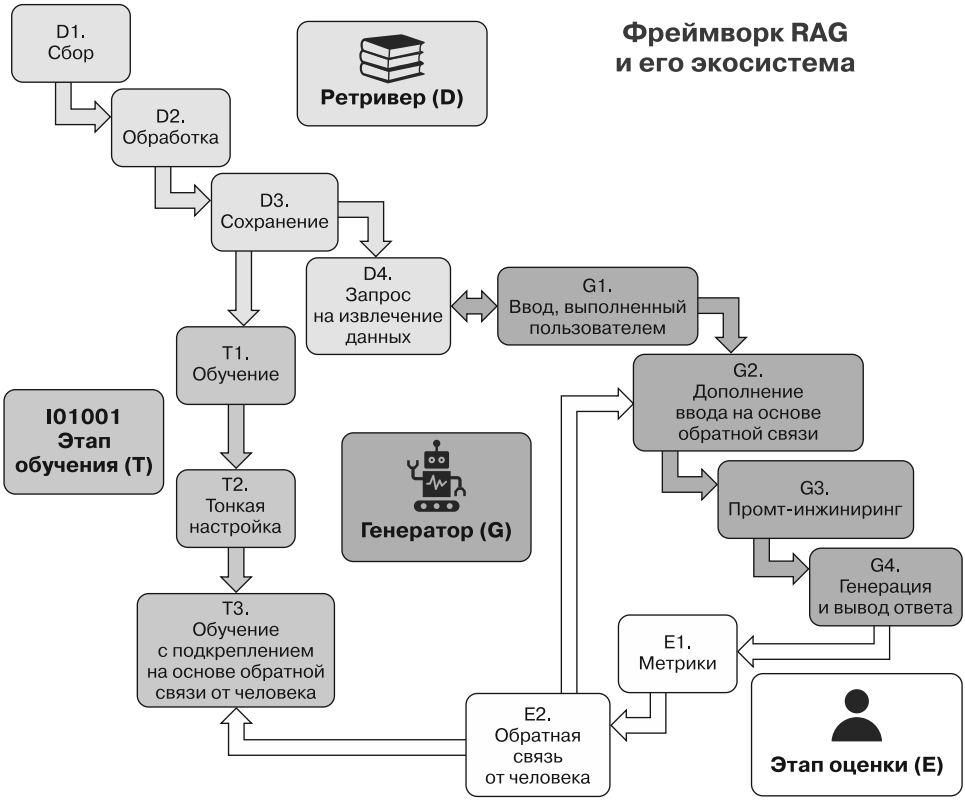

Рис. 1.3. Экосистема генеративного RAG

Каждый из этих четырех компонентов опирается на соответствующие системы, которые формируют общий пайплайн генеративного ИИ на базе RAG. В последующих разделах мы подробнее поговорим о компонентах D, G, E и T. Начнем с ретривера.

Ретривер (D)

Модуль поиска информации в системе RAG собирает, обрабатывает, сохраняет и извлекает данные. В связи с этим стартовой точкой системы выступает процесс поглощения и переработки данных, начинающийся с их сбора.

Сбор (D1)

В современном мире данные ИИ так же разнообразны, как и наши плейлисты медиаконтента. Это может быть что угодно, начиная с фрагмента текста в посте и заканчивая мемом или даже последним эстрадным хитом, транслируемым прямо в наушники. И это далеко не все — файлы могут иметь всевозможные форматы

и размеры. К примеру, PDF, заполненный кучей информации, веб-страницы, простые текстовые файлы, четко организованные файлы JSON, запоминающиеся записи MP3, видео MP4, а также PNG- или JPG-изображения.

Хуже того, значительная часть этих данных не структурирована и встречается в непредсказуемых и сложных формах. К счастью, многие платформы, такие как Pinecone, OpenAI, Chroma и Activeloop, предоставляют готовые инструменты для обработки и хранения таких хаотичных данных.

Обработка (D2)

В схеме мультимодальной обработки на этапе сбора данных D1 различные их типы, такие как текст, изображения и видео, посредством веб-скрапинга извлекаются с сайтов или из других источников. Затем собранные объекты данных преобразуются в однородное представление в виде признаков. Например, данные могут разбиваться на чанки (небольшие части), преобразовываться в векторы (эмбеддинг) и индексироваться для облегчения их поиска и извлечения.

Сохранение (D3)

На этом этапе пайплайна мы уже собрали и начали обрабатывать большой объем разнообразных данных — видео, картинок, текста, да чего угодно. Как теперь поступить с этими данными, чтобы сделать их полезными?

Здесь на помощь приходят такие векторные базы данных, как Deep Lake, Pinecone и Chroma. Можете рассматривать их как умные библиотеки, которые не просто хранят ваши данные, но и преобразуют их в математические сущности — векторы, доступные для вычислительной обработки. Кроме того, в этих хранилищах для ускорения доступа к данным могут применяться различные методы индексирования и прочие техники.

Вместо хранения собранной информации в статических электронных таблицах и файлах мы преобразуем ее в динамическую систему с возможностью поиска, которая может лежать в основе различных инструментов, начиная с чат-ботов и заканчивая механизмами поиска.

Запрос данных (D4)

Процесс извлечения активируется пользовательским или автоматизированным вводом (G1).

Для быстрого извлечения данных после преобразования в подходящий формат мы загружаем их в векторные БД в виде датасетов. Затем, используя комбинацию из поиска по ключевым словам, создания эмбеддингов и индексирования, сможем эффективно эти данные извлекать. К примеру, с помощью косинусной меры сходства можно находить тесно связанные элементы, обеспечивая не только ускоренный поиск, но и выдачу актуальных результатов. После извлечения данных мы дополняем ими ввод.

Генератор (G)

Глядя на рис. 1.3, представляющий фреймворк и экосистему RAG, мы видим, что граница, разделяющая ввод и извлечение, сильно размыта. Ввод G1, будь то автоматизированный или выполненный человеком, взаимодействует с запросом данных (D4), из которого дополняется перед отправкой в генеративную модель.

Иначе говоря, поток генерации начинается с ввода.

Ввод (G1)

Ввод может быть представлен пакетом автоматизированных задач (к примеру, обработка электронных писем) или промтов, переданных человеком через *User Interface* (UI). Эта гибкость позволяет легко интегрировать ИИ в различные профессиональные среды, повышая продуктивность в соответствующих отраслях.

Дополнение ввода обратной связью от человека (G2)

Ввод можно дополнить *обратной связью* (human feedback, HF). Обратная связь позволяет значительно повысить адаптируемость экосистемы RAG, а также обеспечить полный контроль над извлечением данных и вводом для генеративного ИИ. В разделе «Создание гибридной адаптивной RAG на Python» главы 5 мы реализуем механизм дополнения ввода обратной связью от человека.

Промт-инжиниринг (G3)

При подготовке базового и дополненного сообщения, которое нужно будет обработать модели ИИ, ретривер (D) и генератор (G) активно опираются на промт-инжиниринг. В ходе этого процесса вывод ретривера и ввод, выполненный пользователем, объединяются.

Генерация ответа и его вывод (G4)

Выбор модели генеративного ИИ зависит от целей проекта. Llama, Gemini, GPT и прочие их варианты подходят для разных задач. При этом промт должен отвечать характеристикам любой выбранной модели. Фреймворки наподобие LangChain, который мы реализуем позже, помогают упростить интеграцию различных моделей в приложения, предоставляя гибкие интерфейсы и инструменты.

Модуль оценки (E)

При анализе качества модели генеративного ИИ мы зачастую опираемся на математические метрики. Однако они позволяют увидеть лишь часть картины. Здесь важно помнить, что финальной проверкой эффективности ИИ является оценка человеком.

Метрики (E1)

Как и любую систему ИИ, модель невозможно оценить без использования метрик, таких как косинусное сходство (оно же коэффициент Отиаи). Они обеспечивают актуальность и точность извлеченных данных. Количественно измеряя связанность и релевантность точек данных, метрики предоставляют прочную основу для анализа качества и надежности модели.

Обратная связь от человека (E2)

Любая система ИИ, как на базе RAG, так и нет, даже при, казалось бы, достаточном количестве метрик должна проходить этап оценки человеком. Именно он в конечном счете решает, заслуживает ли созданная для людей система похвалы или критики и нужно ли ее принять или отклонить.

Механизм адаптивной RAG вносит человеческую, основанную на реальности обратную связь, которая повышает качество экосистемы генеративного ИИ. Мы реализуем адаптивную RAG в главе 5.

Учитель (T)

Типичная модель генеративного ИИ предварительно обучается на обширных объемах общих данных. Затем эту модель можно настроить (T2), используя данные из конкретной предметной области.

Но мы пойдем еще дальше и в главе 9 интегрируем статические данные RAG в процесс тонкой настройки. Кроме того, включим в этот процесс технику *обучения с подкреплением на основе отзывов людей* (Reinforcement Learning from Human Feedback, RLHF), основанную на получении ценной обратной связи от человека-эксперта.

Теперь мы готовы приступить к написанию простой, продвинутой и модульной RAG на Python.

Простая, продвинутая и модульная RAG в коде

В этом разделе мы познакомимся с простой, продвинутой и модульной системами RAG, используя базовые обучающие примеры. Эта программа будет включать сопоставление по ключевым словам, векторный поиск и извлечение данных по индексу. Применяя GPT-модели от OpenAI, она будет генерировать ответы на основе входящих запросов и извлеченных документов.

Цель этого блокнота — создать диалоговый агент, который будет отвечать на вопросы по RAG в целом. Здесь мы с нуля напишем на Python полноценный ретривер и запустим генератор на базе GPT-4o, разбив все это на две части, включающие восемь секций кода.

Часть 1. Основы и базовая реализация

1. *Настройка среды* для интеграции OpenAI API.
2. *Функция генерации* на основе GPT-4o.
3. *Настройка данных* с помощью списка документов (`db_records`).
4. *Запрос ввода* от пользователя.

Часть 2. Продвинутые техники и оценка

1. *Метрики извлечения* для оценки его результатов.
2. *Простая RAG* на основе поиска по ключевым словам с помощью функции сопоставления.
3. *Продвинутая RAG* с использованием векторного поиска и поиска по индексу.
4. *Модульная RAG*, реализующая гибкие методы извлечения данных.

Для начала откройте `RAG_Overview.ipynb` в репозитории GitHub. Мы начнем с блокнота и познакомимся с базовой реализацией.

Часть 1. Основы и базовая реализация

В этом разделе мы настроим среду, создадим функцию генератора и функцию для вывода форматированного ответа, а также определим механизм создания запроса пользователем.

Первым шагом будет установка рабочей среды.

> **ПРИМЕЧАНИЕ**
>
> Названия разделов приведенной далее реализации блокнота соответствуют структуре кода. То есть вы можете как повторять код из блокнота, так и читать этот самодостаточный раздел.

1. Среда

Основным пакетом для доступа к GPT-4o через API будет OpenAI:

```
!pip install openai==1.40.3
```

Обязательно зафиксируйте установленную версию OpenAI. В экосистеме фреймворков RAG для работы с продвинутыми конфигурациями потребуется установить несколько пакетов. Закончив с настройкой, мы зафиксируем версию этих пакетов, чтобы минимизировать возможные конфликты между библиотеками и реализуемыми нами модулями.

После установки `openai` создайте аккаунт на платформе OpenAI (если у вас его еще нет) и получите ключ API. Прежде чем запускать API, ознакомьтесь с условиями и стоимостью предлагаемых тарифных планов.