



RAG: Retrieval-Augmented Generation

The diagram illustrates the RAG (Retrieval-Augmented Generation) architecture. It features three main components: a retrieval component (blue box), a generation component (green box), and a fusion component (purple box). The retrieval component has an output arrow pointing to the generation component. The generation component has an output arrow pointing to the fusion component. The fusion component has an output arrow pointing to the retrieval component, forming a feedback loop. Additionally, there are direct connections from the retrieval component to the fusion component, and from the generation component to the fusion component.

О спикере

По науке:

- бак. ММФ НГУ
- маг. ФПМИ МФТИ feat Tinkoff
- асп. ММФ НГУ

По индустрии:

- из мира разработки
- ex-{ Promsoft, Tinkoff }
- Pretrain TechLead, GigaChat,
SberDevices



Advanced Prompting Strategies

Chain-of-Thought (CoT) Prompting

Стандартный подход:

Q: "У Маши было 23 яблока, она съела 7. Сколько осталось?"

A: "16"

Zero-shot CoT (добавьте "думай пошагово"):

Q: "У Маши было 23 яблока, она съела 7. Сколько осталось?"

Давай решим пошагово."

A: "Решим пошагово:

1. Изначально: 23 яблока
2. Съела: 7 яблок
3. Осталось: $23 - 7 = 16$ яблок

Ответ: 16"

Продвинутые техники промптинга

- **Tree of Thoughts (ToT):** Исследование альтернативных путей рассуждения
- **Self-Consistency:** Множественные генерации с мажоритарным голосованием
- **Meta-prompting:** Промпты для создания лучших промптов
- **Few-shot CoT:** Примеры рассуждений в контексте
- **Least-to-Most:** Разбиение сложных задач на подзадачи

Retrieval-Augmented Generation (RAG)

Концепция: Гибридная память

Параметрическая память

- Знания в весах модели
- Быстрый доступ
- Статичность
- Склонность к галлюцинациям

Непараметрическая память

- Внешняя база знаний
- Свежие данные
- Верифицируемость
- Дополнительная latency

Базовый RAG Pipeline

1. Query → Embedding model → Query vector
2. Vector search → Retrieve relevant documents
3. Context = Query + Retrieved docs → LLM → Response

Современные RAG техники

- **HyDE (Hypothetical Document Embeddings)**: Генерация гипотетического ответа для лучшего поиска
- **RAPTOR**: Иерархическая индексация с суммаризацией
- **Self-RAG**: Модель сама решает, когда искать и оценивает релевантность
- **Multi-hop reasoning**: Итеративные запросы для сложных вопросов
- **Corrective RAG**: Проверка и коррекция найденных документов

RAG: Работа с различными типами данных

Текстовые документы vs Структурированные данные

Неструктурированный текст

- PDF, Word, веб-страницы
- Естественное embeddings представление
- Простой semantic search
- Контекст в параграфах

Структурированные данные

- Таблицы (CSV, Excel, SQL)
- JSON/XML документы
- Графы знаний
- Требуют специальной обработки

Вызовы при работе с таблицами

1. Потеря структуры при эмбеддинге

- Колонки и строки теряют связи
- Заголовки отделяются от данных

2. Числовые данные

- Плохо представляются в векторном пространстве
- Требуют специальной нормализации

3. Контекст ячеек

- Значение зависит от заголовков строк/колонок
- Метаданные таблицы критичны

RAG для табличных данных

Стратегии индексации таблиц

1. Row-based Chunking

```
# Каждая строка как отдельный документ
doc = f"Product: {row['name']}, Price: ${row['price']},
      Category: {row['category']}, Stock: {row['stock']}"
```

2. Column Summaries

```
# Суммаризация колонок для семантического поиска
summary = f"Column 'revenue' contains quarterly revenue data
            from 2020-2024, ranging from $1M to $5M"
```

3. Table-to-Text

```
# Преобразование таблицы в естественный текст
text = "The sales report shows that iPhone 15 generated
       $2.3M in Q4 2024, representing 35% growth ..."
```

4. Hybrid Approach

Продвинутые техники RAG для таблиц

Text-to-SQL подход

```
# 1. Конвертируем вопрос в SQL
question = "Какой товар принёс больше всего выручки в Q4?"
sql_query = "SELECT product_name, SUM(revenue) as total
            FROM sales WHERE quarter = 'Q4'
            GROUP BY product_name ORDER BY total DESC LIMIT 1"

# 2. Выполняем запрос
result = execute_sql(sql_query)

# 3. Генерируем ответ
answer = f"Больше всего выручки в Q4 принёс {result['product_name']}
с общей суммой ${result['total']}"
```

Table Understanding промпты

Given the following table structure:

Date	Product	Revenue	Units
...

User question: "Какова динамика продаж iPhone?"

Analyze the table to answer the question.

Chunking Strategies для RAG

Семантическое разделение

- **Sentence splitting:** По границам предложений
- **Semantic chunking:** По смысловым блокам
- **Document structure:** По заголовкам/разделам
- **Sliding window:** Фиксированные окна с перекрытием

Recursive Chunking

1. Большие чанки (2000 токенов) для контекста
2. Средние чанки (512 токенов) для поиска
3. Маленькие чанки (128 токенов) для точности

Гибридный поиск в RAG

Комбинирование методов поиска

Vector Search (Semantic)

- Embeddings similarity
- Хорошо для смысловых запросов
- "Найди документы про экономический рост"

Keyword Search (BM25/TF-IDF)

- Точное совпадение терминов
- Хорошо для имён, кодов, ID
- "Договор №12345"

Reranking и фильтрация

Two-stage Retrieval

Stage 1: Быстрый поиск (1000 кандидатов)

↓ Vector/BM25 search

Stage 2: Точный reranking (top 10)

↓ Cross-encoder model

Final: Отфильтрованные результаты

Cross-Encoder Reranking

Концепция: Bi-Encoder vs Cross-Encoder

Bi-Encoder (Dual Encoder)

Query: "машинное обучение"

↓ encode

Query_vector: [0.1, 0.3, -0.2, ...]

Document: "ML это область ИИ"

↓ encode

Doc_vector: [0.2, 0.4, -0.1, ...]

Similarity: cosine(Query_vector, Doc_vector)

Cross-Encoder

Input: "[CLS] машинное обучение [SEP]

ML это область ИИ [SEP]"

↓ joint encoding

Hidden states: [h1, h2, h3, ...]

↓ classifier

Relevance score: 0.89

Преимущества: Точное моделирование взаимодействий

Недостатки: Медленно, нужно вычислять для каждой пары

Преимущества: Быстро, можно предвычислить embeddings

Недостатки: Нет взаимодействия между query и document

Metadata Filtering

```
# Предварительная фильтрация по метаданным
# их можно генерировать с помощью LLM
filters = {
    "date": {"$gte": "2024-01-01"},
    "department": "sales",
    "confidential": False
}
results = vector_db.search(query, filters=filters)
```

Оценка качества RAG системы

Метрики retrieval

- **Recall@k:** Процент релевантных документов в топ-k
- **MRR (Mean Reciprocal Rank):** Позиция первого релевантного
- **NDCG:** Нормализованный DCG для ранжирования

Метрики генерации

- **Faithfulness:** Соответствие ответа источникам
- **Answer Relevance:** Релевантность ответа вопросу
- **Context Relevance:** Релевантность найденного контекста

RAG Evaluation Framework

```
# Автоматическая оценка с LLM-as-Judge
def evaluate_rag_response(question, context, answer):
    faithfulness = llm_judge(
        "Is the answer faithful to the context?",
        context, answer
    )
    relevance = llm_judge(
        "Does the answer address the question?",
        question, answer
    )
    return {
        "faithfulness": faithfulness,
        "relevance": relevance,
        "overall": (faithfulness + relevance) / 2
    }
```

Практические паттерны RAG

Query Transformation

```
# Multi-Query: генерируем варианты вопроса
original = "Выручка за последний квартал"
variants = [
    "Revenue in Q4 2024",
    "Продажи за октябрь–декабрь 2024",
    "Финансовые результаты 4 квартал"
]

# HyDE: генерируем гипотетический ответ
hypothetical = "Выручка компании за Q4 2024 составила $X млн,
    что на Y% больше чем в Q3 ..."
```

Contextual Compression

```
# Сжимаем найденный контекст до релевантных частей
def compress_context(query, documents):
    compressed = []
    for doc in documents:
        # Извлекаем только релевантные предложения
        relevant_sentences = extract_relevant(query, doc)
        compressed.append(relevant_sentences)
    return compressed
```

Iterative Refinement

1. Initial Query → RAG → Answer v1
2. "Уточни данные по региону Европа" → RAG → Answer v2
3. "Сравни с прошлым годом" → RAG → Answer v3

Релевантные проекты YaCamp

RAG-проекты в разработке

Оптимизация графа знаний для RAG

- Ускорение построения knowledge graphs
- Обучение малых LLM для извлечения триплетов
- Поэтапное извлечение: сущности → отношения → описания

RAG на табличных данных

- QA-система для работы с таблицами
- T5/LLM как ретриверы
- Участие в SemEval 2025 - Task 8

АнтиКоллизия: поиск противоречий

- Выявление внутренних/внешних противоречий
- Локальная система с LLM + RAG
- Работа на домашнем компьютере

Кросс-модальная RAG-система

- Any-to-Any retrieval (текст/изображение/аудио/видео)
- Unified vector space на базе Qwen2.5-Omni
- FastAPI + Faiss/Milvus deployment

RAG для специфичных доменов

Financial Data

- Таблицы с временными рядами
- Связи между показателями
- Требования к точности чисел
- Compliance и audit trail

Medical/Clinical Data

- Структурированные медкарты
- Терминология и онтологии
- Privacy (не индексируем PII)
- Связи диагноз-лечение

Legal Documents

- Иерархия документов
- Ссылки и цитирования
- Версионность
- Точное цитирование

Code Repositories

- Структура проекта
- Зависимости между файлами
- Комментарии + код
- Git history как контекст

Advanced RAG: Graph RAG

Концепция Graph RAG

Традиционный RAG: Documents → Chunks → Vectors

Graph RAG: Documents → Entities → Knowledge Graph → Traversal

Graph RAG

Преимущества для сложных запросов

1. Multi-hop reasoning

Q: "Кто руководит отделом, где работает Иван?"

Path: Иван → работает в → Отдел продаж → руководитель → Мария

2. Агрегация информации

Q: "Все проекты клиента X"

Graph: Клиент X → связан с → [Проект 1, Проект 2, ...]

3. Обнаружение связей

Q: "Как связаны продукт A и технология B?"

Graph: Кратчайший путь через промежуточные узлы

Реализация

- Neo4j + векторный поиск
- GraphRAG от Microsoft
- Knowledge Graph embeddings

Tool Use и Agent Capabilities

Function Calling (Tool Use)

Определение инструментов:

```
{  
  "name": "get_weather",  
  "description": "Get current weather for a location",  
  "parameters": {  
    "type": "object",  
    "properties": {  
      "location": {"type": "string"},  
      "unit": {"type": "string", "enum": ["celsius", "fahrenheit"]}  
    },  
    "required": ["location"]  
  }  
}
```

ReAct Framework (Reasoning + Acting)

Question: "Какая погода в Москве и стоит ли брать зонт?"

Thought: Нужно получить текущую погоду в Москве

Action: `get_weather(location="Moscow", unit="celsius")`

Observation: Температура 15°C, дождь

Thought: Идёт дождь, нужно рекомендовать зонт

Answer: В Москве сейчас 15°C, идёт дождь. Обязательно возьмите зонт!

Современные Agent фреймворки

- **LangChain/LangGraph:** Построение сложных цепочек и графов агентов
- **AutoGPT/AutoGen:** Автономные агенты с планированием
- **OpenAI Assistants API:** Встроенная поддержка инструментов
- **Reflexion:** Агенты с self-reflection и обучением на ошибках

Заключение по RAG

Ключевые принципы и эволюция

RAG как мост между статическими LLM и динамическим миром:

Решённые проблемы:

- Hallucinations через верифицируемые источники
- Свежесть данных без переобучения
- Специализация под домены

Направления роста:

- Интеграция с агентными системами
- Мультимодальный поиск и генерация
- Автономные системы принятия решений