Introduction to Retrieval Augmented Generation (RAG)

Yogesh Haribhau Kulkarni

Outline

Retrieval Augmented Generation (RAG)

Quiz

- How to do domain adaptation? ie
- How to build custom LLM solution? meaning
- How can LLM based chatbot answer from my data?

Answers??

YHK

Need for Domain Adaptation

- Industrial settings prioritize cost, privacy, and reliability of solutions.
- LLMs are prone to hallucinations, factual errors, and looping.
- LLMs need to work on own/private/streaming/latest data
- Speed/latency/efficiency Concerns
- Solution: To build LLM from scratch?? needs??
- Extensive training data, high computational resources and $$$$

So, what to do?

Solutions

- Few Shots
- RAG
- Fine-tuning

Why RAG?

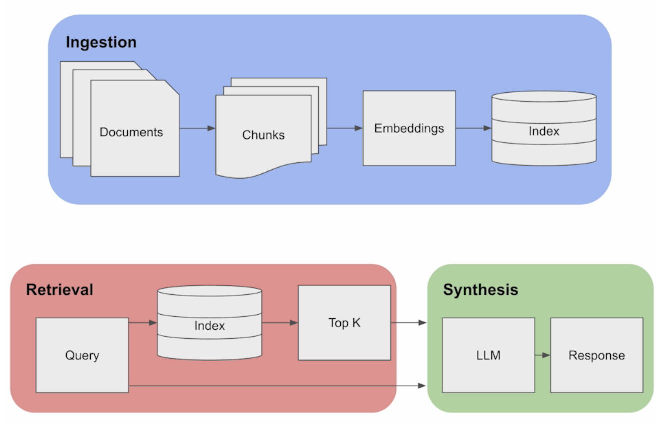
- **Unlimited Knowledge:**
  - RAG enables access external sources, surpassing limitations of its data.
  - Allows exploration of proprietary documents and internet searches
- **Easier to Update/Maintain:**
  - Offers a cost-effective way to update and maintain
  - Building a knowledge base minimizes ongoing maintenance financial burden.
- **Confidence in Responses:**
  - Enhances confidence by providing extra context for more accurate responses.
  - Practical boost to overall intelligence in generating responses.
- **Source Citation:**
  - RAG provides access to sources, improving transparency in LLM responses.
  - A step towards building trust in LLM systems.
- **Reduced Hallucinations:**
  - RAG-enabled LLMs exhibit reduced creative misfires.
  - Solid foundation of information keeps models focused and grounded.

Important Questions

- Does the use case require external data access?
- Does the use case require changing foundation model style?
- Does the use case require addressing hallucinations?
- Is labeled training data available?
- Is citing the source of information important?
- How critical is system latency?
- What are the cost implications?
- What are the scalability requirements?
- Do we have the necessary expertise?

Retrieval Augmented Generation (RAG) Workflow

RAG Components



(Ref: [Week 4] Retrieval Augmented Generation -Aishwarya Naresh Reganti)

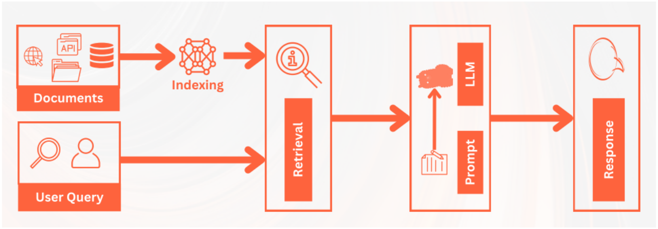
RAG Components

- Ingestion:
  - Documents undergo segmentation into chunks, and embeddings are generated from these chunks, subsequently stored in an index.
  - Chunks are essential for pinpointing the relevant information in response to a given query, resembling a standard retrieval approach.
- Retrieval:
  - Leveraging the index of embeddings, the system retrieves the top-k documents when a query is received, based on the similarity of embeddings.
- Synthesis:
  - Examining the chunks as contextual information, the LLM utilizes this knowledge to formulate accurate responses.

Naive RAG Approach

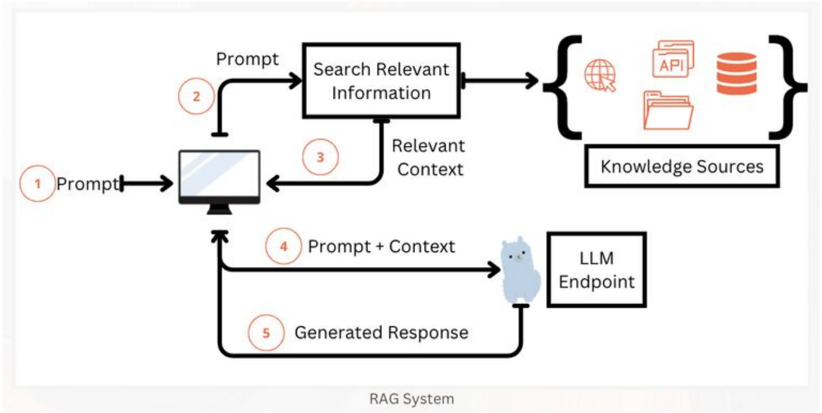
Simply retrieve texts and provide to language model as additional context during generation.

- Concept: Retrieve relevant documents from an external corpus before generation.
- Steps: 1. Input query/topic. 2. Retrieve documents. 3. Generate summary/response based on retrieved documents.
- Pros: Simple and effective for factual tasks.
- Cons: May lack fluency and originality due to heavy reliance on retrieved text.



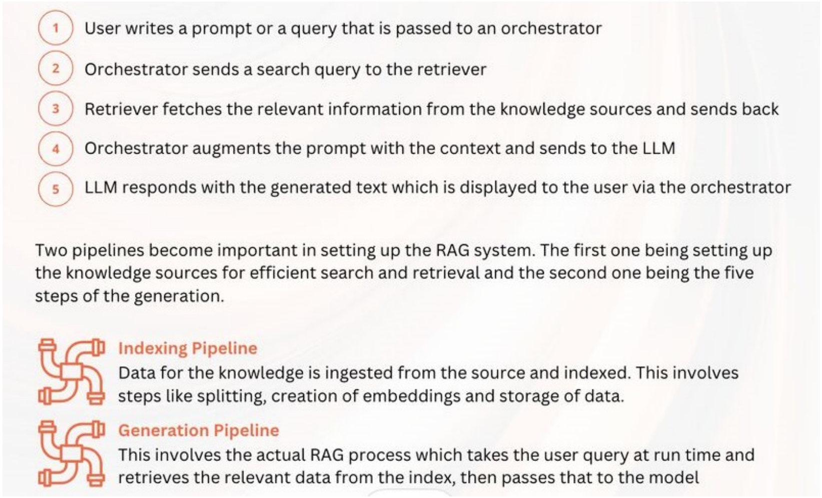
(Ref: Progression of RAG Systems - Abhinav Kimothi)

RAG Architecture



(Ref: RAG Architecture -Abhinav Kimothi)

RAG Workflow



1. User writes a prompt or a query that is passed to an orchestrator
2. Orchestrator sends a search query to the retriever
3. Retriever fetches the relevant information from the knowledge sources and sends back
4. Orchestrator augments the prompt with the context and sends to the LLM
5. LLM responds with the generated text which is displayed to the user via the orchestrator

Two pipelines become important in setting up the RAG system. The first one being setting up the knowledge sources for efficient search and retrieval and the second one being the five steps of the generation.

Indexing Pipeline
Data for the knowledge is ingested from the source and indexed. This involves steps like splitting, creation of embeddings and storage of data.

Generation Pipeline
This involves the actual RAG process which takes the user query at run time and retrieves the relevant data from the index, then passes that to the model

(Ref: RAG Architecture -Abhinav Kimothi)

Challenges in Naive RAG

- Retrieval Quality
  - Low Precision leading to Hallucinations/Mid-air drops
  - Low Recall resulting in missing relevant info
  - Outdated information
- Augmentation
  - Redundancy and Repetition when multiple retrieved documents have similar information
  - Context Length challenges
- Generation Quality
  - Generations are not grounded in the context
  - Potential of toxicity and bias in the response
  - Excessive dependence on augmented context

(Ref: Progression of RAG Systems - Abhinav Kimothi)

YHK

RAG vs SFT (Supervised Fine - Tuning)

RAG & SFT: Complementary Techniques

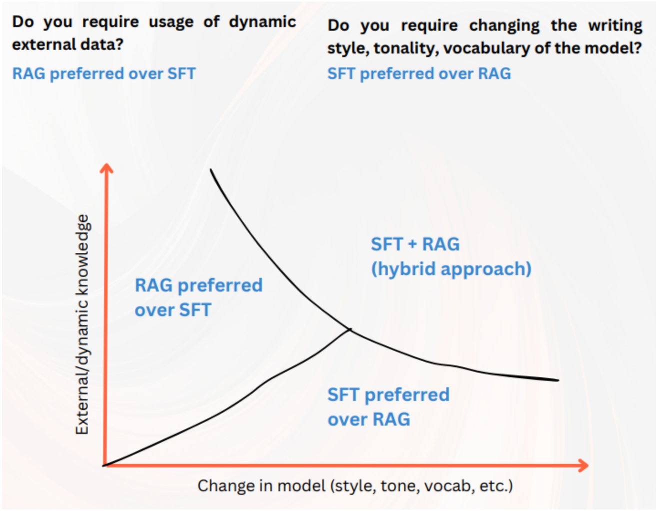
RAG Features

- Connects to dynamic external data sources.
- Reduces hallucinations.
- Increases transparency in source of information.
- Works well with very large foundation models.
- Does not impact style, tone, vocabulary.

SFT Features

- Changes style, vocabulary, tone of foundation model.
- Can reduce model size.
- Useful for deep domain expertise.
- May not address hallucinations.
- No improvement in transparency (black box models).

Important Use Case Considerations


Do you require usage of dynamic external data?
RAG preferred over SFT
Do you require changing the writing style, tonality, vocabulary of the model?
SFT preferred over RAG
External/dynamic knowledge
RAG preferred over SFT
SFT + RAG (hybrid approach)
SFT preferred over RAG
Change in model (style, tone, vocab, etc.)

(Ref: Generative AI with Large Language Model - Abhinav Kimothi)

Other Considerations

- **Latency:** RAG introduces inherent latency due to search and retrieval.
- **Scalability:** RAG pipelines are modular and scalable; SFT requires retraining.
- **Cost:** Both methods require upfront investment; costs vary.
- **Expertise:** RAG pipelines are moderately simple with frameworks; SFT needs deep understanding and training data creation.

Applications

Applications and Use Cases

- Code generation in software development
- Creative writing and storytelling
- Educational material generation
- Personalized product descriptions
- many many more . . .

Open Source Resources and Tools

- LangChain, LlamaIndex like frameworks
- Transformers library, Hugging Face model hub
- Datasets and evaluation benchmarks

RAG Advanced Concepts

Advanced RAG Approaches

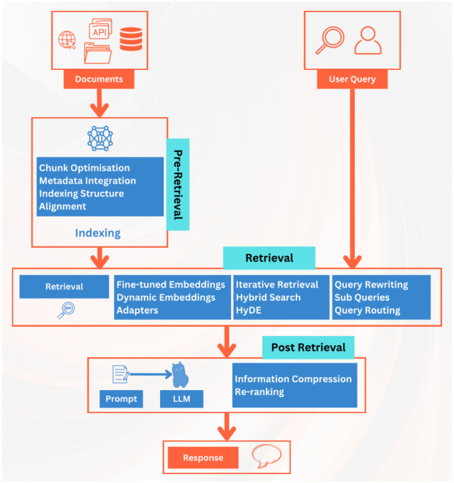
To address the inefficiencies of the Naive RAG approach, Advanced RAG approaches implement strategies focused on three processes:

- Pre-Retrieval
- Retrieval
- Post Retrieval

More complex integration through various processing layers between retriever and generator modules.

- Hybrid Model: Jointly train retrieval and generation models for stronger synergy.
- Conditional Attention: Dynamically weigh retrieved documents based on their relevance to the current generation stage.
- Controllable Generation: Introduce mechanisms to steer generated content towards retrieved information.

Advanced RAG Approaches



(Ref: Progression of RAG Systems - Abhinav Kimothi)

Advanced RAG Stages: Pre-retrieval

Chunk Optimization

- Break external documents into appropriately sized chunks
- Decision factors: content type, user queries, and application needs
- No one-size-fits-all strategy; flexibility is crucial
- Current research explores techniques like sliding windows and ”small2big” methods

Metadata Integration

- Embed information like dates, purpose, chapter summaries, etc., into chunks
- Improves retriever efficiency by assessing similarity to metadata

Advanced RAG Stages: Pre-retrieval

Indexing Structure

- Introduction of graph structures enhances retrieval
- Leverages nodes and their relationships
- Multi-index paths aimed at increasing efficiency

Alignment

- Understanding complex data, like tables, can be tricky for RAG
- Improve indexing with counterfactual training
- Create hypothetical (what-if) questions to increase alignment
- Reduce disparity between documents

Advanced RAG Stages: Retrieval

Query Rewriting

- Use rewriting approaches for better alignment between user queries and documents
- LLMs create pseudo-documents from the query for improved matching
- LLMs perform abstract reasoning; multi-querying for solving complex user queries

Hybrid Search Exploration

- RAG system employs different types of searches: keyword, semantic, and vector
- Search type depends on user query and available data

Advanced RAG Stages: Retrieval

Sub Queries

- Break down complex queries into sub-questions for each relevant data source
- Gather intermediate responses and synthesize a final response

Query Routing

- Query router identifies downstream task and decides subsequent action for RAG system
- During retrieval, query router identifies most appropriate data source

Iterative Retrieval

- Collect documents repeatedly based on query and generated response
- Create a more comprehensive knowledge base

Advanced RAG Stages: Retrieval

Recursive Retrieval

- Iteratively retrieves documents and refines search queries based on previous results
- Continuous learning process

Adaptive Retrieval

- Enhance RAG framework by empowering LLMs to proactively identify suitable moments and content for retrieval
- Improve efficiency and relevance of information obtained
- Dynamically choose when and what to retrieve for more precise and effective results

Advanced RAG Stages: Retrieval

Hypothetical Document Embeddings (HyDE)

- LLM-based HyDE forms a hypothetical document in response to a query
- Embeds it and retrieves real documents similar to this hypothetical one
- Emphasizes similarity between embeddings of different answers

Fine-tuned Embeddings

- Tailor embedding models to improve retrieval accuracy
- Particularly useful in specialized domains dealing with uncommon or evolving terms
- Fine-tuning utilizes training data generated with language models grounded in document chunks

Advanced RAG Stages: Post-retrieval

Information Compression

- Retriever proficient in extracting relevant information from extensive knowledge bases
- Challenge: Managing vast amount of information within retrieval documents
- Compress retrieved information to extract the most relevant points
- Information passed to LLM after compression

Reranking

- Re-ranking model crucial for optimizing the document set retrieved by the retriever
- Rearrange document records to prioritize the most relevant ones at the top
- Manages the total number of documents effectively
- Resolves challenges related to context window expansion during retrieval
- Improves efficiency and responsiveness

Still, Challenges in RAG

- **Data Ingestion Complexity:** Handling extensive knowledge bases involves engineering challenges such as parallelizing requests, managing retries, and scaling infrastructure.
- **Efficient Embedding:** Embedding large datasets requires addressing rate limits, implementing robust retry logic, and managing self-hosted models for optimal efficiency.
- **Vector Database Considerations:** Storing data in a vector database introduces challenges like understanding compute resources, monitoring, sharding, and addressing potential bottlenecks.
- **Fine-Tuning and Generalization:** Fine-tuning RAG models for specific tasks while ensuring generalization across diverse knowledge-intensive NLP tasks requires a careful balance.
- **Knowledge Update Mechanisms:** Developing mechanisms to update non-parametric memory as real-world knowledge evolves is crucial for adapting to changing information.

Failure Points of RAG Systems

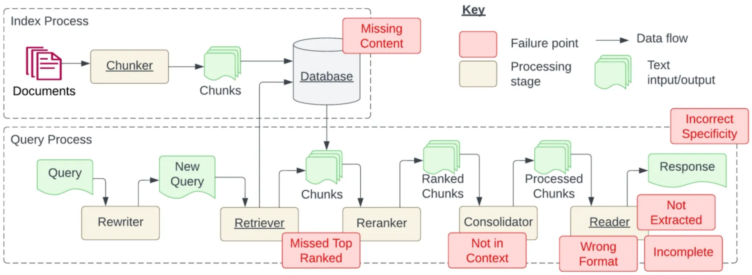


Figure 1: Indexing and Query processes required for creating a Retrieval Augmented Generation (RAG) system. The indexing process is typically done at development time and queries at runtime. Failure points identified in this study are shown in red boxes. All required stages are underlined. Figure expanded from [19].

(Ref: Mastering RAG: How To Architect An Enterprise RAG System - Pratik Bhavsar)

Failure Points

- **Missing Content (FP1):**
  - Question without answer in available documents.
  - Ideal response: "Sorry, I don't know."
  - Risk of system being misled for questions without clear answers.
- **Missed Top Ranked Documents (FP2):**
  - Answer in document, but not ranked high enough.
  - Only top K documents returned (K based on performance).
  - Theoretical ranking vs. practical limitations.
- **Not in Context - Consolidation Strategy Limitations (FP3):**
  - Relevant answer retrieval hindered during consolidation.
  - Occurs with a substantial number of returned documents.
  - Challenge in incorporating retrieved documents into context.
- **Not Extracted (FP4):**
  - Answer present, but model fails to extract correct information.
  - Occurs with excessive noise or conflicting information.

(Ref: Mastering RAG: How To Architect An Enterprise RAG System - Pratik Bhavsar)

Failure Points

- **Wrong Format (FP5):**
  - Question involves specific format (table/list), but model disregards.
  - Failure to follow instruction for information extraction.
- **Incorrect Specificity (FP6):**
  - Response lacks required specificity or is overly specific.
  - Predetermined outcomes may lead to incorrect specificity.
  - Unclear user questions may result in overly general responses.
- **Incomplete (FP7):**
  - Accurate answers lacking some information present in context.
  - Example: Key points covered in multiple documents.
  - Approach of asking separate questions for better results.

(Ref: Mastering RAG: How To Architect An Enterprise RAG System - Pratik Bhavsar)

Vertex AI RAG Engine

(Ref: https://cloud.google.com/vertex-ai/generative-ai/docs/rag-engine/rag-overview)

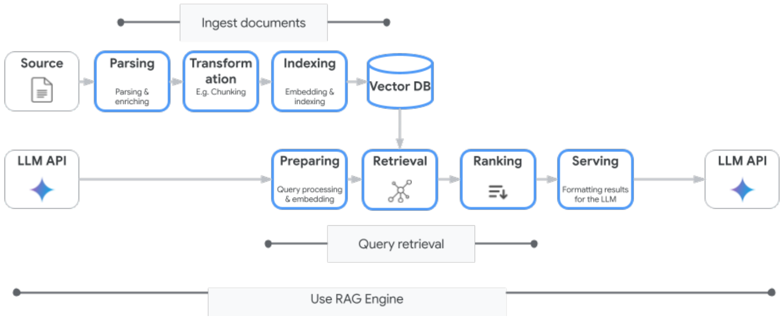
Vertex AI RAG Engine: Introduction

- Component of Vertex AI Platform facilitating Retrieval-Augmented Generation
- Data framework for developing context-augmented LLM applications
- Addresses common LLM limitation: lack of knowledge about private organizational data
- Enriches LLM context with additional private information
- Reduces hallucination and improves answer accuracy
- Combines external knowledge sources with LLM's existing knowledge

RAG Process Overview

- Data Ingestion: Intake from various sources (local files, Cloud Storage, Google Drive)
- Data Transformation: Convert and prepare data for indexing (chunking)
- Embedding: Create numerical representations capturing semantic meaning
- Data Indexing: Create a corpus optimized for searching
- Retrieval: Find relevant information based on user queries
- Generation: Use retrieved context with user query for informed responses

RAG Process Overview



Benefits of Vertex AI RAG Engine

- Provides factual grounding for LLM responses
- Allows integration of proprietary/private data
- Improves response accuracy and relevance
- Reduces LLM hallucinations
- Creates domain-specific AI applications
- Enables knowledge-based conversational experiences

Getting Started with Vertex AI RAG

- Create a new GCP Project (console.cloud.google.com)
- Enable the Vertex AI API
- Install Vertex AI SDK for Python
- Set up project authentication

```
pip install google-cloud-aiplatform

gcloud config set {project}
gcloud auth application-default login
```

YHK

Python SDK Imports (Example)

```python
import vertexai
from google.cloud.aiplatform.private_preview import rag
from vertexai.generative_models import GenerativeModel, Tool

# PROJECT_ID = "your-google-cloud-project-id"
# REGION = "us-central1" # Or your preferred region
# CORPUS_DISPLAY_NAME = "my_rag_corpus"
# GCS_BUCKET_NAME = "your-gcs-bucket-for-rag-docs"
# EMBEDDING_MODEL_PUBLISHER_PATH =
     "publishers/google/models/textembedding-gecko@003" # Verify model

# vertexai.init(project=PROJECT_ID, location=REGION)

```

Creating a RAG Corpus

- Initialize Vertex AI and configure embedding model
- Create a RAG corpus with appropriate configuration

```
vertexai.init(project=PROJECT_ID, location="us-central1")

embedding_model_config = rag.RagEmbeddingModelConfig(
    vertex_prediction_endpoint=rag.VertexPredictionEndpoint(
        publisher_model="publishers/google/models/text-embedding-005"
    )
)

rag_corpus = rag.create_corpus(
    display_name=display_name,
    backend_config=rag.RagVectorDbConfig(
        rag_embedding_model_config=embedding_model_config
    ),
)
```

Importing Files to RAG Corpus

- Import documents from Google Cloud Storage or Google Drive
- Configure chunking parameters for optimal retrieval

```
rag.import_files(
    rag_corpus.name,
    paths,  # GCS or Google Drive paths
    transformation_config=rag.TransformationConfig(
        chunking_config=rag.ChunkingConfig(
            chunk_size=512,
            chunk_overlap=100,
        ),
    ),
    max_embedding_requests_per_min=1000,  # Optional
)
```

Direct Context Retrieval

- Configure retrieval parameters (top-k results, relevance filters)
- Query corpus directly for relevant context

```python
rag_retrieval_config = rag.RagRetrievalConfig(
    top_k=3,  # Optional
    filter=rag.Filter(vector_distance_threshold=0.5),
)
response = rag.retrieval_query(
    rag_resources=[
        rag.RagResource(
            rag_corpus=rag_corpus.name,
            # Optional: specific file IDs
        )
    ],
    text="What is RAG and why it is helpful?",
    rag_retrieval_config=rag_retrieval_config,
)
print(response)
```

Enhancing LLM Generation with RAG

- Create a RAG retrieval tool for the LLM
- Initialize Gemini model with the retrieval tool
- Generate augmented responses

```python
rag_retrieval_tool = Tool.from_retrieval(
    retrieval=rag.Retrieval(
        source=rag.VertexRagStore(
            rag_resources=[rag.RagResource(rag_corpus=rag_corpus.name,)],
            rag_retrieval_config=rag_retrieval_config,),),))

rag_model = GenerativeModel( model_name="gemini-2.0-flash-001",
    tools=[rag_retrieval_tool])

response = rag_model.generate_content(
    "What is RAG and why it is helpful?")
print(response.text)

```

Advanced RAG Capabilities

- Filter retrieval by specific document IDs
- Adjust vector distance thresholds for relevance
- Configure chunking parameters for optimal retrieval
- Scale with configurable embedding request rates
- Support for multiple document formats and sources
- Integration with other Vertex AI capabilities

Best Practices

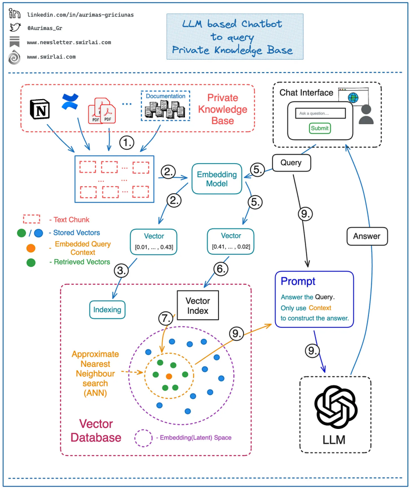
- Optimize chunk size for your specific content
- Balance chunk overlap for contextual coherence
- Tune top-k and relevance thresholds for your use case
- Consider multiple corpora for different knowledge domains
- Test queries against your corpus before deployment
- Monitor and refine based on generation quality

Conclusions

So, What is RAG?

- A new paradigm for generation tasks, combining retrieval and generation models.
- Motivation: Overcoming limitations of pure generative models in factual consistency, efficiency, and diversity.
- Impact: Improved performance in text summarization, question answering, and other NLP tasks.

RAG Architecture



(Ref: Overview of Large Language Models - Aman AI)

RAG Architecture

- **Powerful Enhancement:**
  - RAG provides additional memory and context.
  - Increases confidence in LLM responses.
- **Architecture:**
  - Two essential pipelines for an effective RAG system.
- **Indexing Pipeline:**
  - Retrieves knowledge from diverse sources.
  - Enables loading, splitting, and embedding creation for offline use.
  - Can be on-the-fly for lower volume and single-use scenarios.
- **Generation Pipeline:**
  - Retriever fetches information from the knowledge base.
  - Retrieved data augments user prompt and is sent to the LLM.
  - LLM generates text and sends the response back to the user.
- **Evaluation Pipeline:**
  - Optional pipeline for assessing groundedness and relevance of responses.

(Ref: RAG Architecture -Abhinav Kimothi)

Pros and Cons of RAG

- Pros: Improved factual accuracy, diversity, and efficiency compared to pure generation.
- Cons: Increased complexity, potential bias introduced by the retrieval component, and dependency on external corpus quality.

Challenges and Future Directions

- Bias Mitigation: Addressing potential biases introduced by the retrieval component.
- Domain Adaptation: Adapting RAG models to specific domains with limited training data.
- Interpretability Enhancement: Understanding the reasoning behind generated outputs for better model debugging.

References

Many publicly available resources have been refereed for making this presentation. Some of the notable ones are:

- Retrieval Augmented Generation - Medium blogs by Abhinav Kimothi
- Mastering RAG: How To Architect An Enterprise RAG System - Pratik Bhavsar
- Prompt Engineering Guide https://www.promptingguide.ai/techniques/rag
- 3 Day RAG Roadmap: Understanding, Building and Evaluating RAG Systems 2024 - Aishwarya Naresh Reganti
- Explanation of RAG by DeepLearning AI
- What is RAG by DataStax
- Advanced RAG series (6 videos) by Sam Witteveen
- LangChain101: Question A 300 Page Book (w/ OpenAI + Pinecone) by Greg Kamradt
- Blog on advanced RAG techniques by Akash
- RAG hands-on tutorials on GitHub gkamradt/langchain-tutorials/

Thanks . . .

- Search **"Yogesh Haribhau Kulkarni"** on Google and follow me on LinkedIn and Medium
- Office Hours: Saturdays, 2 to 5pm (IST); Free-Open to all; email for appointment.
- Email: yogeshkulkarni at yahoo dot com

(https://www.linkedin.com/in/yogeshkulkarni/, QR by Hugging Face QR-code-AI-art-generator, with prompt as "Follow me")