Introduction to Retrieval Augmented Generation (RAG)

Yogesh Haribhau Kulkarni

Outline

About Me

Yogesh Haribhau Kulkarni

Bio:

- 20+ years in CAD/Engineering software development
- Got Bachelors, Masters and Doctoral degrees in Mechanical Engineering (specialization: Geometric Modeling Algorithms).
- Currently doing Coaching in fields such as Data Science, Artificial Intelligence Machine-Deep Learning (ML/DL) and Natural Language Processing (NLP).
- Feel free to follow me at:
  - Github (github.com/yogeshhk)
  - LinkedIn (www.linkedin.com/in/yogeshkulkarni/)
  - Medium (yogeshharibhaukulkarni.medium.com)
  - Send email to yogeshkulkarni at yahoo dot com

Office Hours: Saturdays, 2 to 5pm (IST); Free-Open to all; email for appointment.

YHK

Retrieval Augmented Generation (RAG)

Introduction

- 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.

Need for RAG

- Limitations of Pure Generation: Prone to hallucinations, factual errors, and repetitive outputs.
- Importance of Factuality: Crucial for tasks like summarization, question answering, and report generation.
- Efficiency Concerns: Large generative models require extensive training data and high computational resources.
- Industrial settings prioritize cost-conscious, privacy-respecting, and reliable solutions.
- Companies, including startups, seek solutions with a return on investment rather than investing in talent or training models from scratch.
- Recent research and chatbot releases demonstrate the capability to leverage knowledge beyond model weights.
- One approach is to iteratively call another neural network or language model (LM) to extract required information.
- Iteratively calling LM involves multiple interactions to extract information as shown in the provided image.g, as separating reasoning from factual information is not straightforward.

Retrieval Augmented Generation (RAG) for LLMs

- **Unlimited Knowledge:**
  - RAG enables LLMs to access external sources, surpassing limitations of internal data.
  - Allows exploration of proprietary documents and internet searches, broadening understanding.
  - Introduces both Parametric and Non-Parametric Memory, enhancing models' knowledge.
  - Read the seminal paper on RAG - https://lnkd.in/gN2cAcXx.
- **Easier to Update/Maintain:**
  - Offers a cost-effective way to update and maintain non-parametric memory in LLMs.
  - Building a solid knowledge base minimizes ongoing maintenance financial burden.
  - Pre-training expenses are mitigated, and fine-tuning is tailored to specific use cases.
  - Read Gopi's blog on cost implications of LLMs - https://lnkd.in/gJ-bT4RH.

Retrieval Augmented Generation (RAG) for LLMs (Contd.)

- **Confidence in Responses:**
  - RAG enhances LLM confidence by providing extra context for more accurate responses.
  - Practical boost to overall intelligence in generating responses.
- **Context Awareness:**
  - RAG makes LLMs adept at understanding context through added information.
  - Results in responses that are not only accurate but also contextually fitting.
- **Source Citation:**
  - RAG provides access to sources, improving transparency in LLM responses.
  - A step towards building trust in LLM systems.
  - Explore text generation with citations in LLMs - https://lnkd.in/g2VEwRBK.

Retrieval Augmented Generation (RAG) for LLMs (Contd.)

- **Reduced Hallucinations:**
  - RAG-enabled LLMs exhibit reduced creative misfires.
  - Solid foundation of information keeps models focused and grounded.
  - Pinecone explains more in this post - https://lnkd.in/gCfkDZYp.
- **Practical Empowerment:**
  - RAG shifts how we empower LLM systems practically.
  - Focuses on expanding knowledge, building confidence, and maintaining grounded responses.

RAG & SFT: Complementary Techniques

- RAG & SFT are complementary, not competing methods.
- RAG enhances non-parametric memory without changing parameters.
- SFT changes parameters, impacting parametric memory.

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).

Use Case Considerations for RAG & SFT

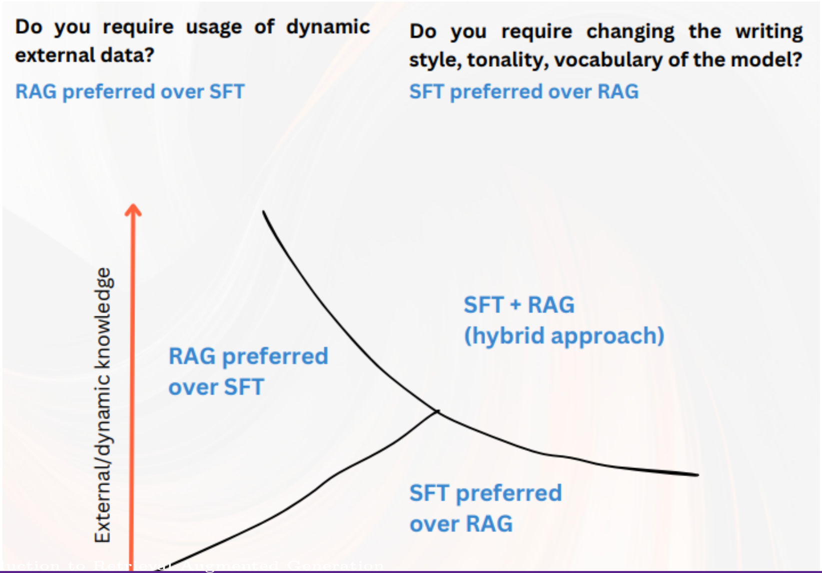
Use both if parametric and non-parametric memory changes are needed.

- Access to dynamic external data source.
- Resolving hallucinations.
- Transparency in source of information.
- SFT requires access to labeled training data.

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?

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?

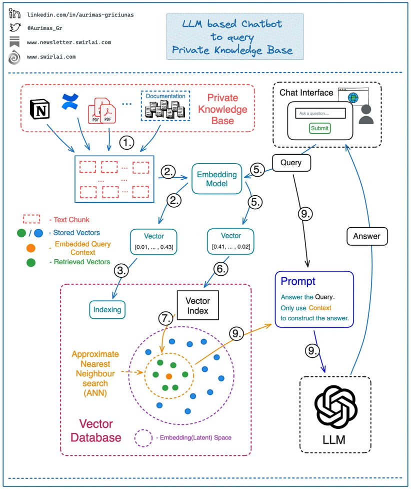
SFT preferred over RAG

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.

Retrieval Augmented Generation (RAG) Workflow

RAG Architecture



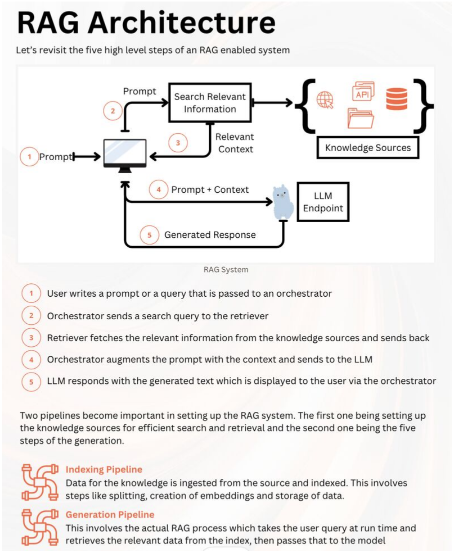
(Ref: Overview of Large Language Models - Aman AI)

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.

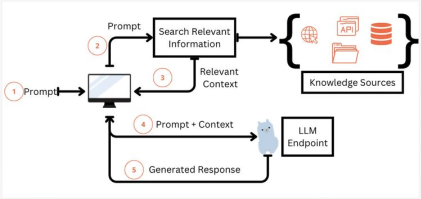
(Ref: RAG Architecture -Abhinav Kimothi)

RAG Architecture



RAG Architecture

Let's revisit the five high level steps of an RAG enabled system



RAG System

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

RAG in Steps

The first step is to convert internal documents into a query-friendly format by embedding them using an embedding model.

- LLMs can gain external knowledge through information retrieval from memory units like external databases.
- There are two types of information retrievers: dense and sparse.
- Sparse retrievers use a sparse bag-of-words representation for documents and queries.
- Dense (neural) retrievers utilize dense query and document vectors obtained from a neural network.
- Retrieval-augmented LMs have shown strong performance in knowledge-intensive tasks, closing the performance gap compared to larger LMs with more parameters.
- Save the text representing each embedding along with its corresponding pointer for future reference.

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

RAG in Steps

Next we can start constructing the answer to a question/query of interest:

- Embed the question using the same Embedding Model as the knowledge base.
- Use the resulting Vector Embedding to query the Vector Database and specify the desired number of retrieved vectors, representing the context for answering the query.
- Perform an Approximate Nearest Neighbour (ANN) search in the Vector DB to find similar vectors in the Embedding/Latent space.
- Map the returned Vector Embeddings to the corresponding text chunks.
- Provide the question and retrieved context text chunks to the LLM via prompt, instructing it to use only the provided context for answering the question.
- Ensure that prompt engineering is applied to ensure the LLM's answers are within expected boundaries, avoiding fabricated answers when there is no relevant data in the retrieved context.

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

RAG in Steps

Next we can start constructing the answer to a question/query of interest:

- Use retrieval augmented generation (RAG) to enhance the knowledge base of an LM with relevant documents.
- Employ vector databases like Pinecone, Chroma, Weaviate, Milvus, or LlamaIndex to augment LLMs.
- RAG step-by-step process:
  - Chunk, embed, and index documents in a vector database (VDB).
  - Utilize (approximate) nearest neighbor techniques to match the query embedding of the claim advisor.
  - Retrieve the relevant context from the VDB.
  - Augment the LLM's prompt with the retrieved content.
- Consider LangChain or Google Vertex for prototyping or industrial applications, respectively.
- Another approach is leveraging the search engine itself, as demonstrated by WebGPT, which can interact with a web browser, refine queries, navigate webpages, follow links, and cite sources.e within expected boundaries, avoiding fabricated answers when there is no relevant data in the retrieved context.

YHK

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

RAG in Steps

Next we can start constructing the answer to a question/query of interest:

- Interact with the LLM through an API or direct interaction when working with prompts.
- Configure parameters to customize prompt results:
  - Temperature: Lower temperature values increase determinism, selecting the highest probable next token. Higher values introduce more randomness, encouraging diversity and creativity in outputs. Lower temperature can be suitable for fact-based QA, while higher temperature may benefit creative tasks like poem generation.
  - Top_p: Using nucleus sampling, top_p controls the determinism of the model's response generation. Lower values prioritize exact and factual answers, while higher values promote diverse responses.
- It is generally recommended to modify only one parameter at a time.
- Results may vary depending on the specific version of the LLM used.

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

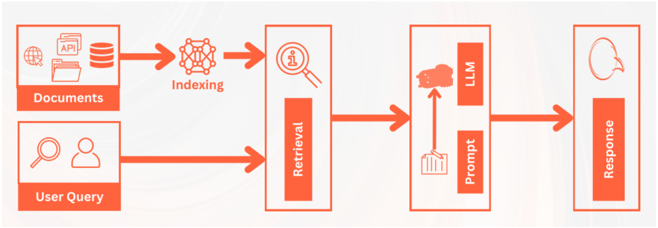
YHK

Approaches

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)

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)

Advanced RAG Approaches

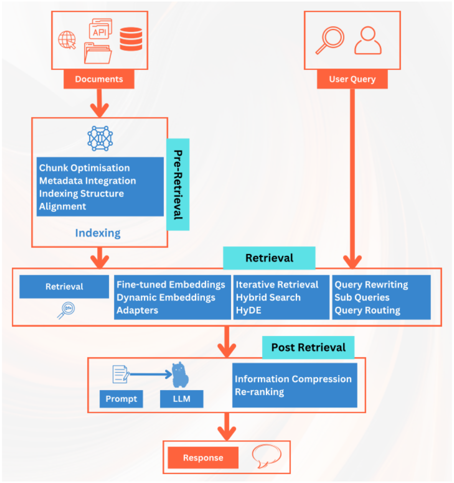
To address the inefficiencies of the Naive RAG approach, Advanced RAG approaches implement strategies focused on three processes:

- 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 weight 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

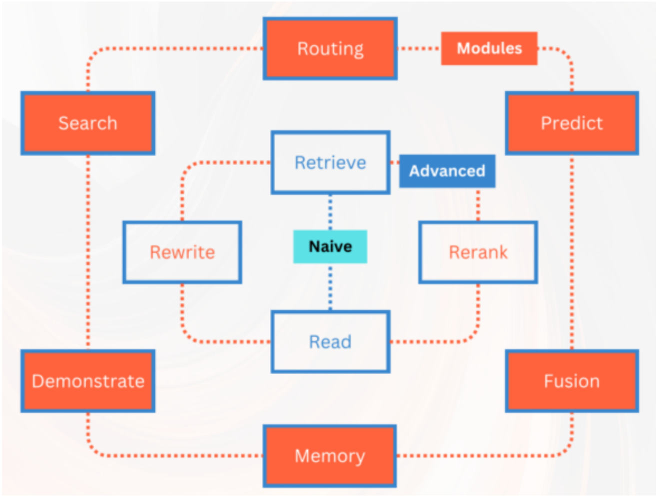
- Re-ranking model crucial for optimizing the document set retrieved by the retriever
- Rearrange document records to prioritize the most relevant ones at the top
- Manages the total number of documents effectively
- Resolves challenges related to context window expansion during retrieval
- Improves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

Modular RAG Architectures

Separate retriever and generator modules that can be independently improved.

- Decomposed Modules: Separate modules for retrieval, filtering, and generation, allowing for customization.
- Allows components like search, memory, and reranking modules to be configured
- Fine-tuning Flexibility: Adapting individual modules to specific tasks or data domains.
- Interpretability Advantages: Easier to analyze the contribution of each module to the final output.

Modular RAG Architectures



(Ref: Progression of RAG Systems - Abhinav Kimothi)

Some RAG Modules

- **Search:**
  - Perform search on different data sources
  - Customized for various data sources
  - Increase source data for better response generation
- **Memory:**
  - Leverage parametric memory capabilities of the Language Model (LLM)
  - Guide retrieval using retrieval-enhanced generator
  - Create an unbounded memory pool iteratively
- **Fusion (RAG-Fusion):**
  - Overcome limitations of traditional search systems
  - Multi-query approach, expanding user queries into diverse perspectives
  - Conduct parallel vector searches for original and expanded queries
  - Intelligently re-rank and optimize results
- **Extra Generation:**
  - Generate context using Language Model (LLM)
  - Address issues of repetition and irrelevant details in retrieved content
- **Task Adaptable Module:**
  - Make RAG adaptable to various downstream tasks
  - Develop task-specific end-to-end retrievers with minimal examples
  - Demonstrate flexibility in handling different tasks

Further RAG Variants and Advancements

- Dense Passage Retrieval
- Transformer-based RAG models
- Multi-stage RAG with diverse retrieval strategies
- Explainable RAG for interpretability
- RAG for low-resource languages

Applications and Use Cases

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

Open Source Resources and Tools

- LangChain framework
- Transformers library
- Hugging Face model hub
- Datasets and evaluation benchmarks

Evaluation and Metrics

- ROUGE for summarization
- BLEU for generation
- F1-score for question answering
- Additional metrics for factual accuracy, diversity, and coherence

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.

LangChain Implementation

- Open-source framework: Designed specifically for modular RAG architectures.
- Modular Components: Retrieval models, document ranking algorithms, and various generation models.
- Customizable Pipelines: Define the flow of information between modules for specific tasks.

LangChain Case Studies

- Text Summarization: Achieves state-of-the-art performance on benchmarks like CNN/Daily Mail and ROUGE.
- Question Answering: Improves factual accuracy and answerability compared to pure generative models.
- Other Applications: News writing, dialogue generation, and abstractive text editing.

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.

YHK

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

(Generated by Hugging Face QR-code-AI-art-generator, with prompt as "Follow me")