Building Production-Ready RAG Applications: Jerry Liu

https://www.youtube.com/watch?v=TRjq7t2Ms5I

~ Prepared by Riya Kharade

➢ Introduction

The video starts with Jerry, the co-founder and CEO of LlamaIndex, explaining how to build production-ready Retrieval-Augmented Generation (RAG) applications. He highlights that RAG is useful for enhancing language models to work with data outside their training. Popular use cases include knowledge search, question-answering systems, conversational agents, workflow automation, and document processing. These applications are becoming essential for developers and businesses to create smarter and more efficient AI tools.

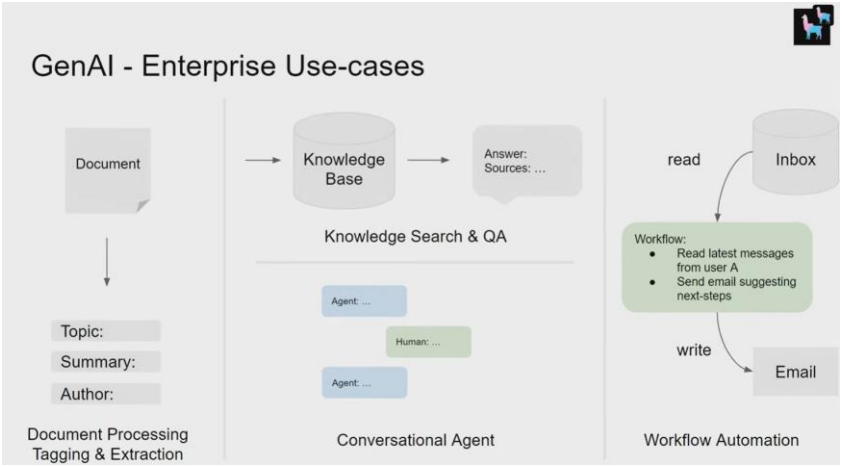
➢ Background and Motivation

The video emphasizes that while language models (LLMs) have become highly capable, they often struggle with data they have not been trained on. To overcome this limitation, developers use RAG (Retrieval-Augmented Generation), which allows models to incorporate external information dynamically.

The motivation behind building production-ready RAG systems is to improve response quality, accuracy, and relevance when handling real-world queries. Common challenges include hallucinations, irrelevant outputs, outdated data, and incomplete context. By

optimizing retrieval, embeddings, and synthesis processes, developers can build more reliable AI applications for tasks like knowledge search, QA systems, and document processing.

➢ GenAI – Enterprise Use-cases



1. **Document Processing, Tagging & Extraction**
   - Input: Documents.
   - Processes: Extract key information such as topic, summary, and author.
   - Purpose: Structure unstructured data for further processing.

2. **Knowledge Search & Question Answering (QA) / Conversational Agent**
   - Input: Knowledge Base built from processed documents.
   - Processes:
     - Search relevant information.
     - Generate answers based on sources.
     - Enable human-agent interactions for conversations.

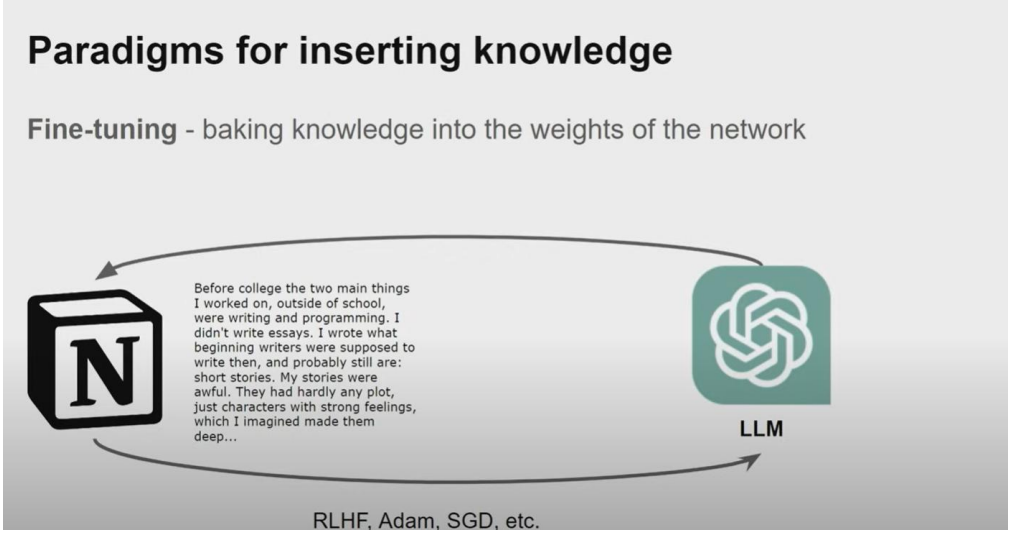
   Purpose: Improve decision-making, provide instant responses, and facilitate interactive assistance.

3. **Workflow Automation**
    - Input : User inbox or other sources.
    - Processes:
        - Read latest messages or data.
        - Suggest next steps (e.g., send emails automatically).
    - Purpose: Automate repetitive tasks and increase efficiency.

Overall Insight:

Generative AI transforms enterprise workflows by automating data extraction, enhancing information retrieval, and enabling smart automation.

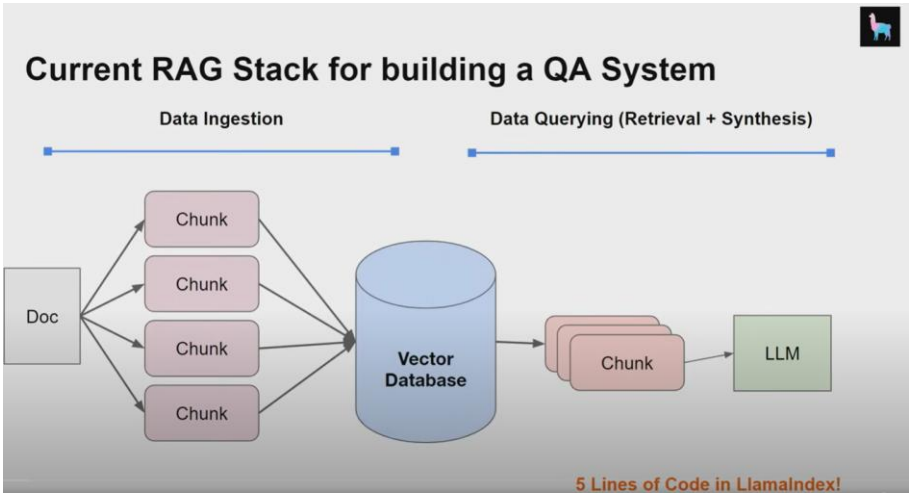
➢ Paradigms for inserting knowledge:



Two main paradigms for integrating data into language models:

1. **Retrieval Augmentation**: Adding external context into prompts using databases like vector stores, SQL, or unstructured text.
2. **Fine-Tuning**: Updating the model's weights by training it with new data for better knowledge retention.

➢ Current RAG Stack for building a QA System



- Core components: Data Ingestion and Data Querying.
- Engineers are encouraged to learn underlying systems, such as how data is retrieved from vector databases and synthesized by LLMs.

Common Challenges

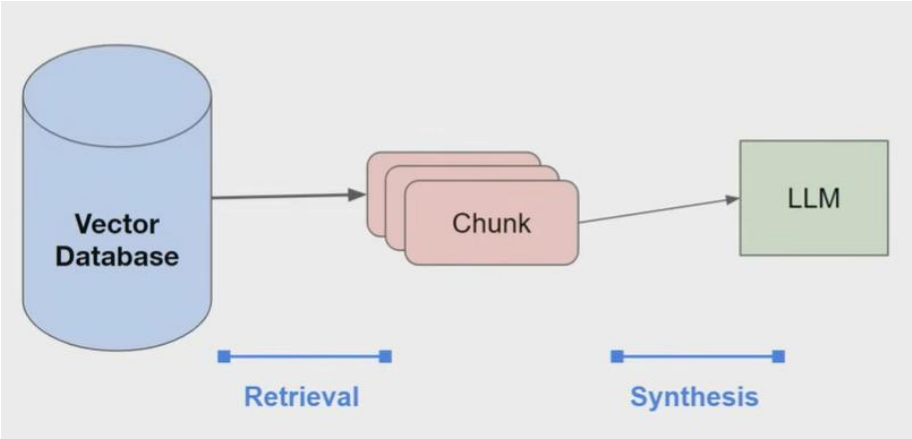
- Poor response quality due to:
    - Low precision and irrelevant chunks.
    - Hallucinations and fluff.
    - Low recall or missing key information.
    - Outdated data and bias.

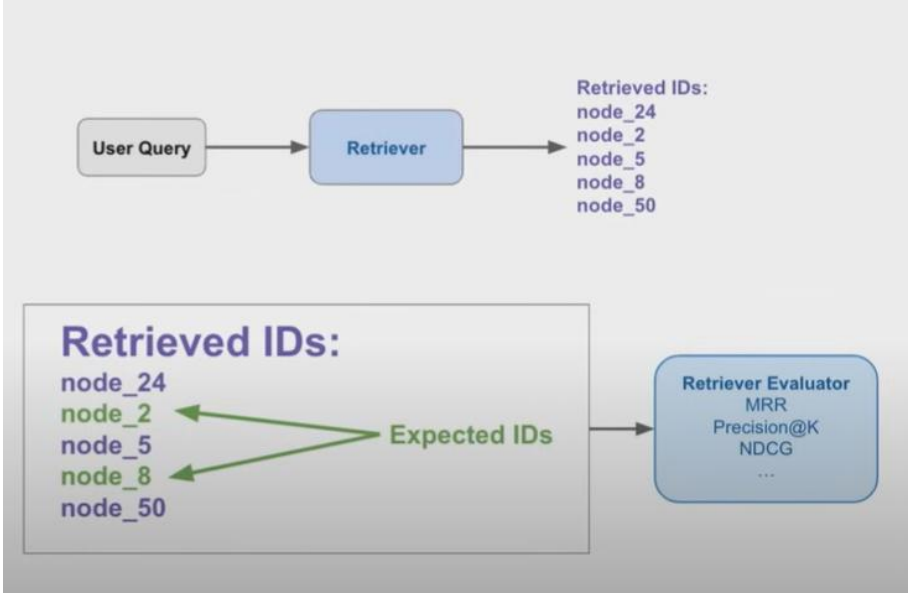
Improving RAG Systems

- Suggestions to optimize performance:
    - Store additional metadata.
    - Adjust chunk sizes.
    - Fine-tune embedding representations.
    - Optimize retrieval algorithms.
    - Use synthesis for reasoning instead of only generation.

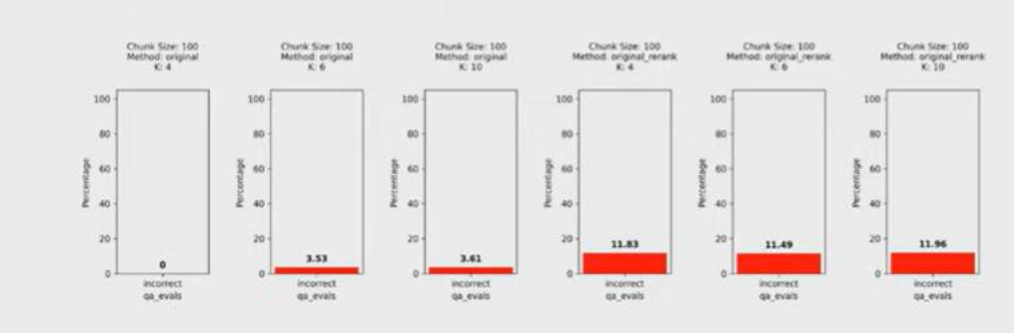
➢ Evaluation Metrics

- Importance of defining benchmarks.
- Metrics include success rate, hit rate, MRR, nDCG.
- Use human feedback, synthetic datasets, or automated LM evaluation methods.





➢ Table Stakes: Chunk Sizes



1. Chunk Size

- A chunk is a segment of your data (like a paragraph or a set of rows) that you feed to a model for retrieval.
- Chunk Size = 100 here means each chunk contains 100 tokens (words/subwords) from the data.
- Adjusting chunk size impacts performance. Too small can miss context; too large can dilute relevance.

2. Performance Impact

- The graph shows the percentage of incorrect answers (qa_evals) for different methods.
- Observations:
    - More retrieved tokens ≠ better performance. Even if you retrieve more context, it may not improve the model's accuracy.
    - Smaller chunks sometimes perform better because they are more focused and relevant.

3. Reranking

- Reranking means shuffling or reordering the retrieved chunks based on relevance before passing them to the model.
- The note says reranking isn't always helpful. Sometimes shuffling context doesn't improve results and can even slightly hurt performance.

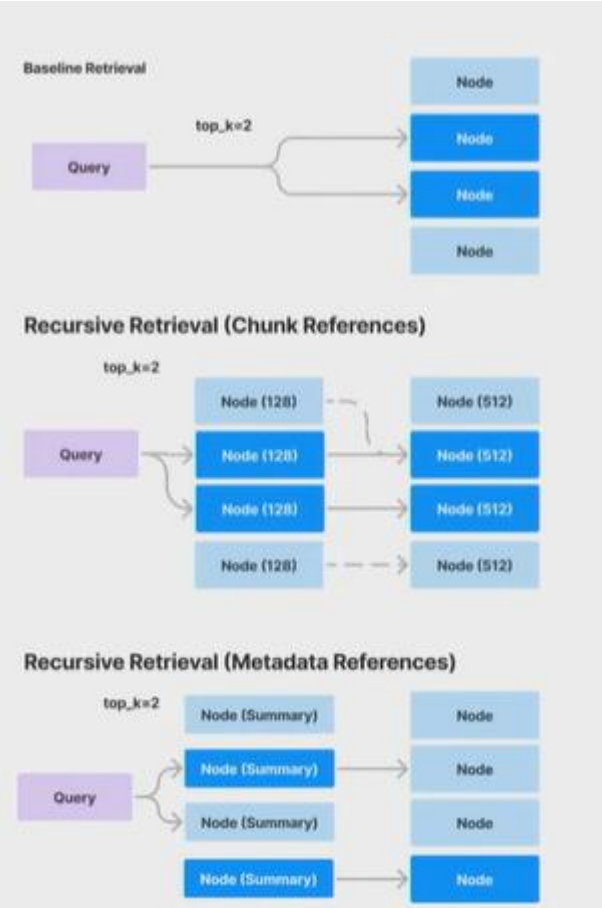
4. Methods and K Values

- Method: original → No reranking; chunks are retrieved as-is.
- Method: original_rerank → Chunks are reranked for relevance.
- K → Number of chunks retrieved per query (4, 6, 10 in this case).
- Observations from the graphs:
    - K = 4 has lower error than K = 6 or 10, suggesting more chunks don't always help.
    - Reranking slightly increases errors in this example, confirming the note.

✓ **Basic Improvements**

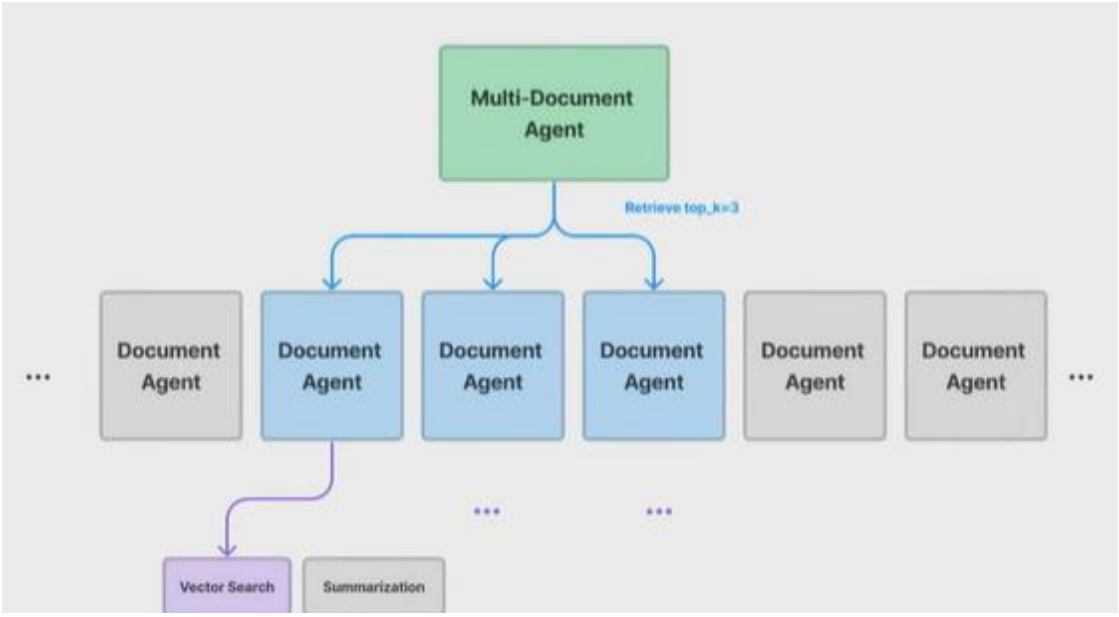
- Table Stakes: Start with basics like tuning chunk sizes and applying metadata filters.
- Hybrid searches and reranking methods improve retrieval.
- Embedding references instead of full chunks boosts efficiency.

✓ Advanced Retrieval Techniques



- Small-to-big retrieval: Embed smaller pieces first, expand at synthesis time for more precision.
- Parent trunk embedding: Use summaries or references to guide retrieval.

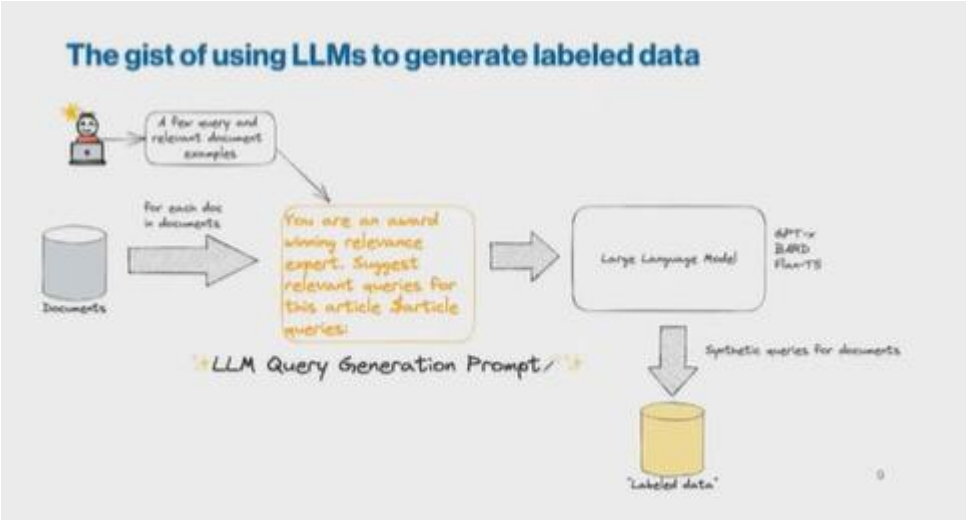
➢ Agents for Reasoning



Instead of only answering queries, agents can:

- Summarize multiple documents.
- Perform advanced reasoning across datasets.
- Retrieve structured tools to synthesize information.

➢ Fine-Tuning for Performance



- Use larger models like GPT-4 to generate synthetic datasets.
- Fine-tune weaker models like GPT-3.5 to improve reasoning and structure.
- Fine-tune embeddings or adapters to boost retrieval.

➢ Key Takeaways & Conclusion

1. **RAG Boosts LLMs** –

   Retrieval-Augmented Generation allows language models to access external knowledge, improving accuracy, relevance, and reliability.

2. **Production Challenges** –

   Common issues like hallucinations, irrelevant outputs, outdated data, and incomplete context need careful optimization.

3. **Core System Components** –

   Data ingestion, retrieval, and synthesis form the backbone of a RAG application.

4. **Optimization Strategies** –

   Tune chunk sizes, apply metadata filters, use advanced retrieval methods, fine-tune embeddings/models, and leverage agent reasoning.

5. **Enterprise Applications** –

   Useful in document processing, knowledge search/QA, conversational agents, and workflow automation.

6. **Evaluation is Key** –

   Measure performance using metrics like hit rate, success rate, MRR, and nDCG; combine automated evaluation with human feedback.

- **Conclusion:**

  RAG enables smarter, production-ready AI systems by combining LLM capabilities with external knowledge. Optimizing retrieval, chunking, embeddings, and reasoning ensures higher accuracy, efficiency, and applicability in real-world enterprise tasks.