Introduction to Graph RAG (Retrieval Augmented Generation)

Yogesh Haribhau Kulkarni

YHK

About Me

YHK

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)

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?

YHK

Solutions

- Few Shots
- RAG
- Fine-tuning

YHK

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.

YHK

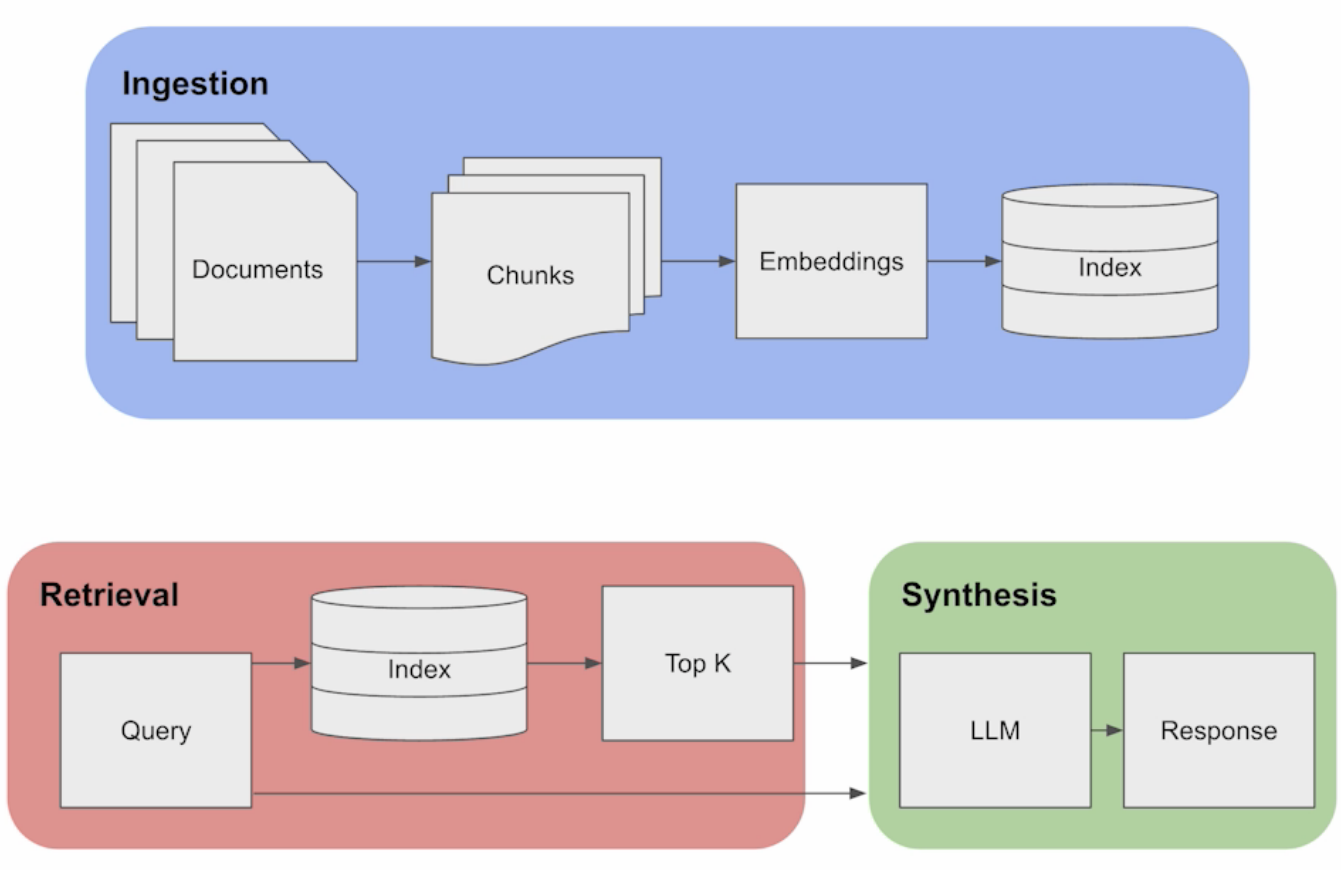
Important Questions

- 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?

YHK

Retrieval Augmented Generation (RAG) Workflow

RAG Components



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

YHK

RAG Components

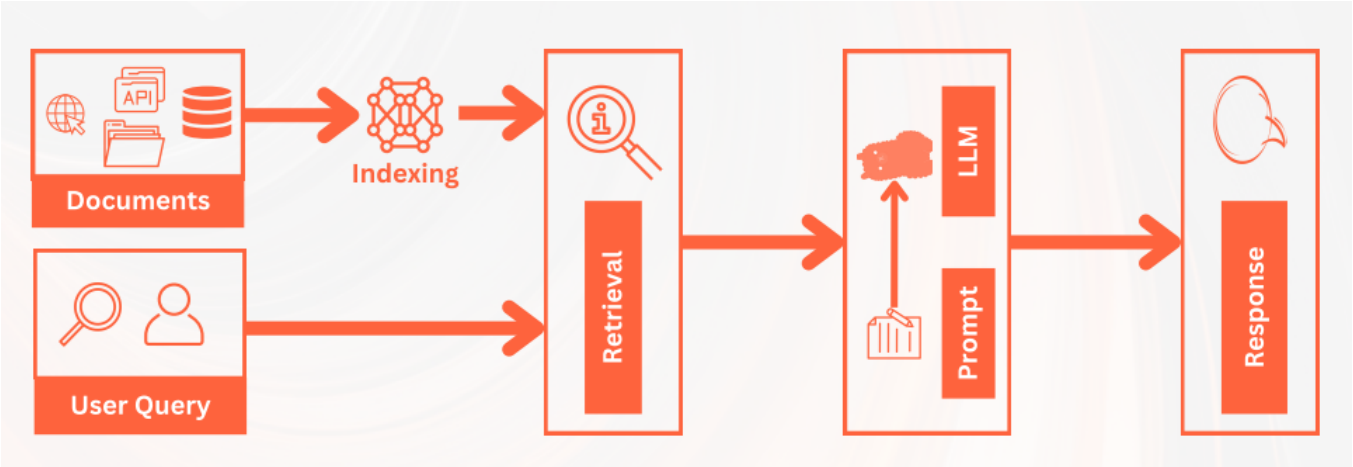
- Ingestion:
  - Documents undergo segmentation into chunks, and embeddings are generated from these chunks, subsequently stored in an index.
  - Chunks are essential for pinpointing the relevant information in response to a given query, resembling a standard retrieval approach.
- Retrieval:
  - Leveraging 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.

YHK

Naive RAG Approach

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

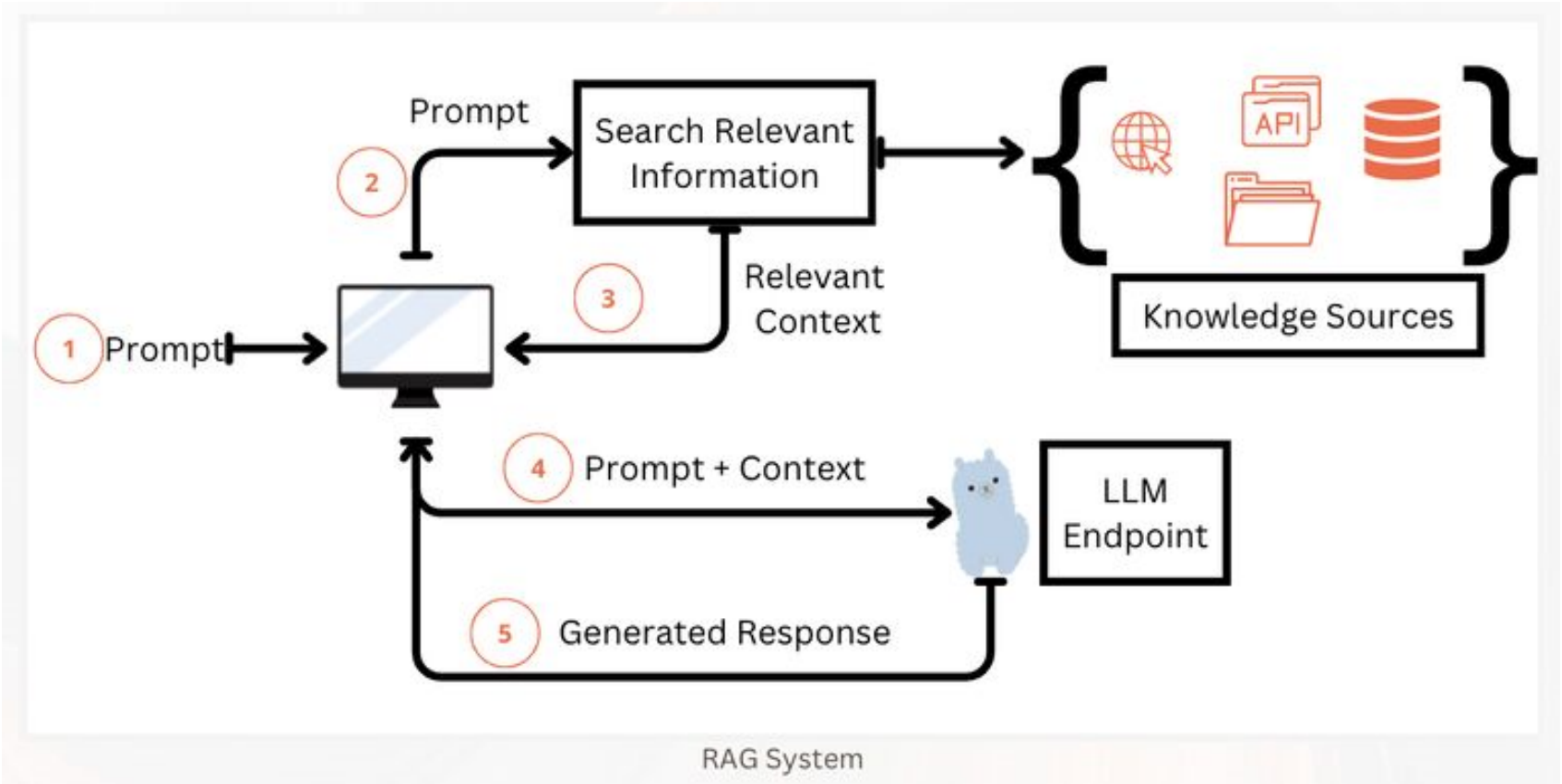
- Concept: Retrieve relevant documents from an external corpus before generation.
- Steps: 1. Input query/topic. 2. Retrieve documents. 3. Generate summary/response based on retrieved documents.
- Pros: Simple and effective for factual tasks.
- Cons: May lack fluency and originality due to heavy reliance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



(Ref: Progression of RAG Systems - Abhinav Kimothi)

YHK

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)

YHK

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).

YHK

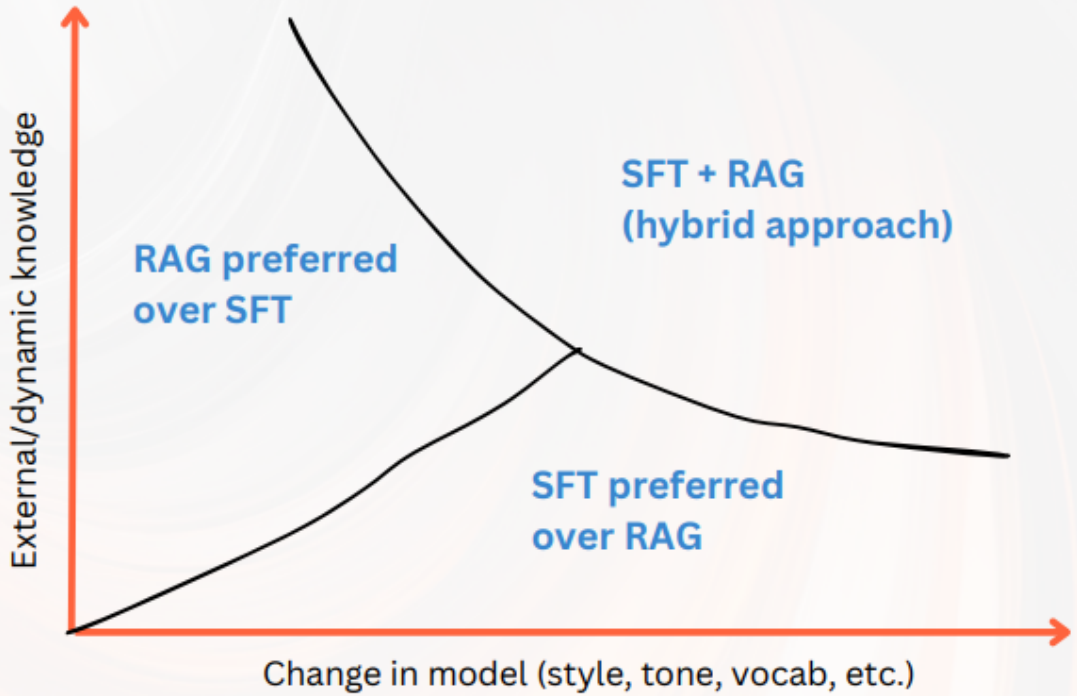
Important Use Case Considerations

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



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

YHK

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.

YHK

Applications

Applications and Use Cases

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

YHK

Open Source Resources and Tools

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

YHK

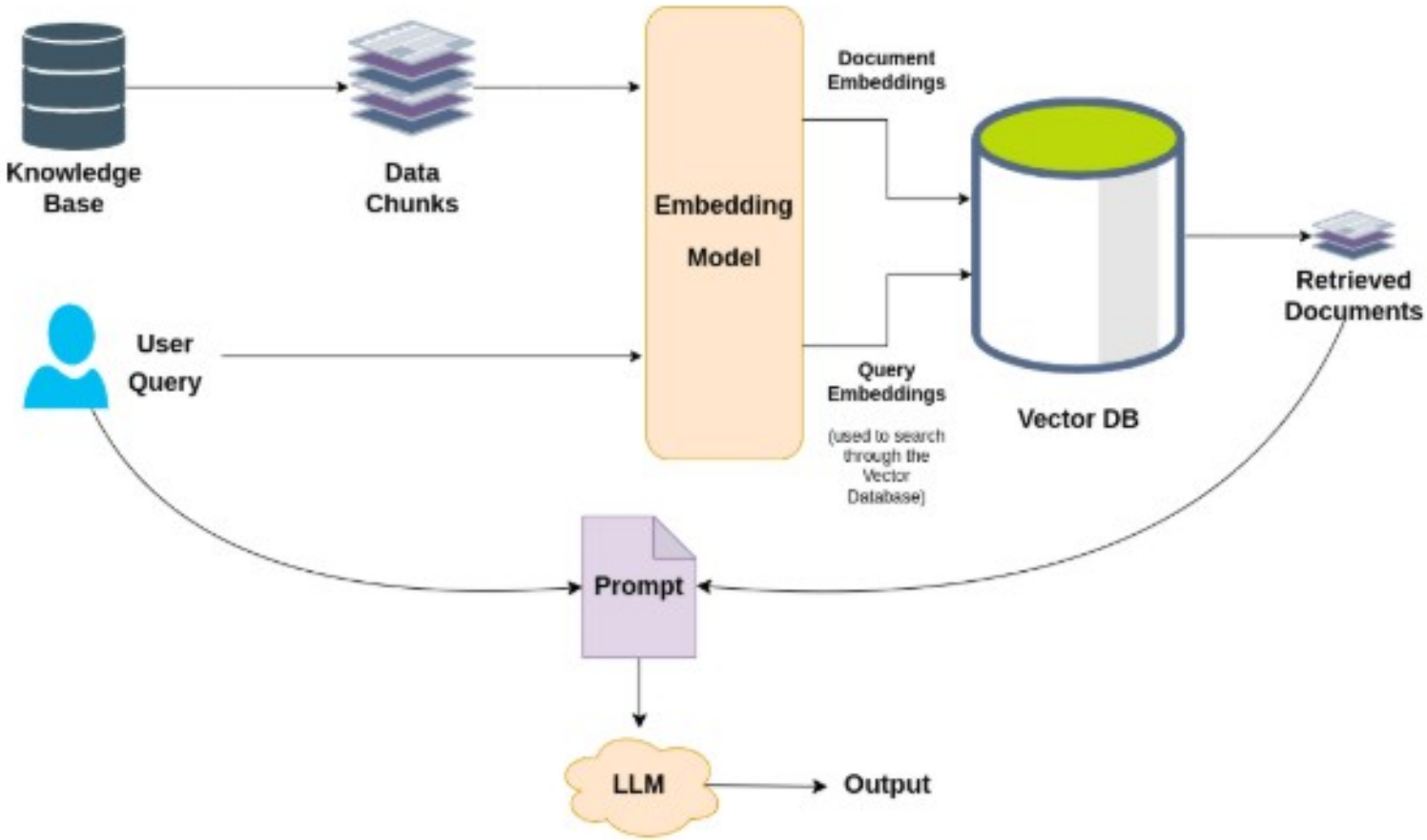
Introduction to Graph RAG

Why Graph RAG?

- Language models struggle with factual accuracy and real-world knowledge.
- Retrieval-Augmented Generation (RAG) improves accuracy using external text data.
- Traditional RAG has limitations in context understanding and scalability.
- GraphRAG leverages knowledge graphs for better retrieval and response generation.

YHK

Overview of Traditional RAG



(Ref: GraphRAG: The Practical Guide for Cost-Effective Document Analysis with Knowledge Graphs -Jaykumaran)

YHK

Limitations of Traditional RAG

- **Flat Retrieval:** Documents are treated as isolated entities.
- **Contextual Shortcomings:** Lacks deep semantic understanding.
- **Scalability Issues:** Slower retrieval with increasing data volume.
- Struggles with scalability due to flat data representation.
- Lacks global context over the entire data corpus.
- Inefficient for complex reasoning across multiple documents.
- **Lack of Explainability:** Hard to trace the source of retrieved information.
- **Local Window:** Limited chunk-level context leads to incomplete responses.
- **Loss of Structural Relationships:** Ignores hierarchical relationships in data.

YHK

What is GraphRAG?

- RAG on Graph: combines traditional RAG techniques with graph-based knowledge representations to leverage structural relationships between entities.
- Can be done multiple ways
  - English Query is converted to GraphQL or Cypher query then calls GraphDB to fetch the context/answers (Neo4j way). Uses few-shots or specialized parsers or fine-tuned LLMs for the conversion
  - Knowledge Graph is indexed, then English query then sent to index (which fetches, relevant triplets, also node's data) (Langchain/Llamaindex way)
  - From documents, populates the Knowledge Graph first, indexes it, then brings the relevant context from the index (Microsoft way)
  - Contextual Subgraph/Path-based Retrieval Approach: Extracts reasoning paths or connected subgraphs for the input query (mostly academic research)

YHK

Different Approaches

Approach	Description	Key Differentiators
Neo4j	LLM extracts entities, builds KG, translates queries to Cypher	Domain-specific entities + structured queries
LlamaIndex	Hierarchical graph + community detection + multi-level summaries	Layered context resolution
Microsoft	KG construction + Leiden clustering + community summaries	Dataset-wide insights
Hybrid	Vector search + graph traversal	Semantic + relational fusion

YHK

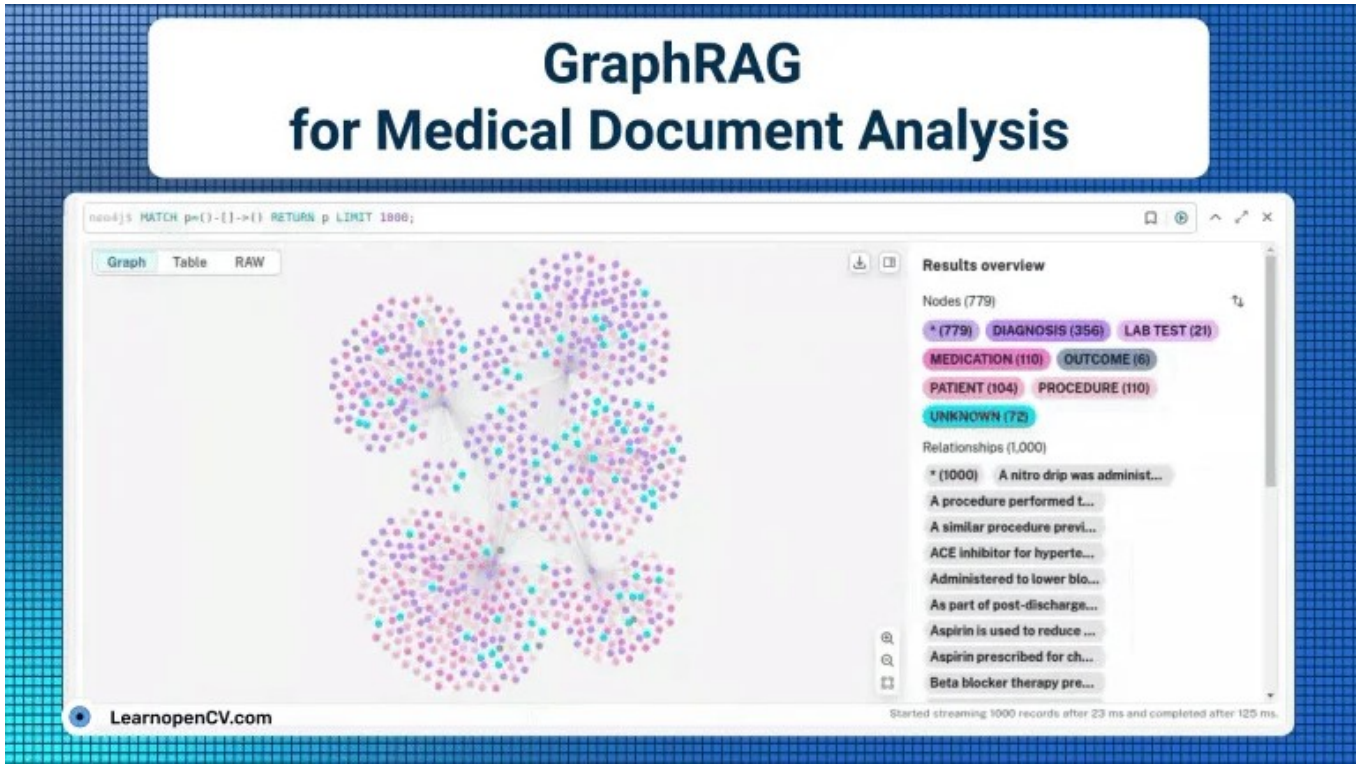
Key Features of GraphRAG

- **Structured Representation:** Uses knowledge graphs.
- **Contextual Retrieval:** Understands semantic relationships.
- **Efficient Processing:** Reduces computational cost.
- **Multi-Faceted Queries:** Synthesizes data from multiple sources.
- **Explainability:** More transparent than black-box models.
- **Continuous Learning:** Expands knowledge over time.

YHK

Example: Medical Query Challenge

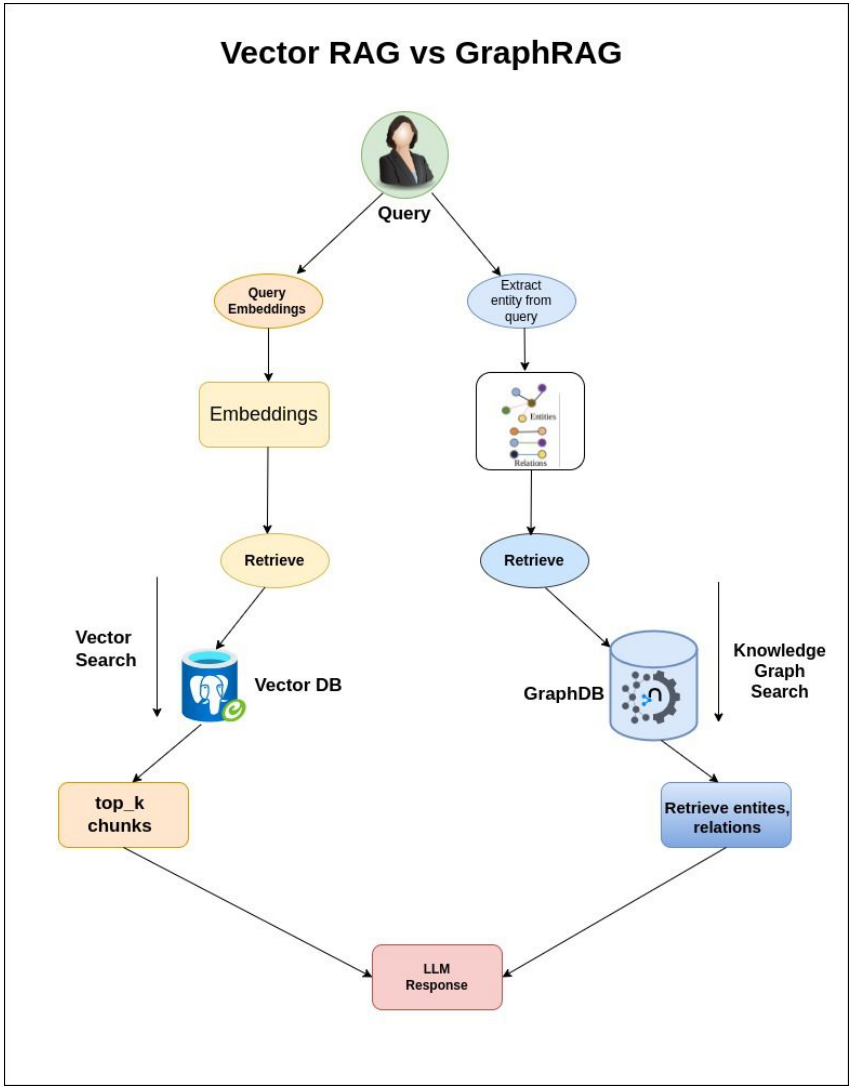
- Query: How does Medication A in Patient Record 1 affect Condition B in Patient Record 2?
- LLM needs to infer relationships across multiple records.
- Standard RAG struggles with such complex dependencies.
- Scaling this to millions of patient records is infeasible.



(Ref: GraphRAG: The Practical Guide for Cost-Effective Document Analysis with Knowledge Graphs -Jaykumaran)

YHK

Vector RAG vs GraphRAG



(Ref: GraphRAG: The Practical Guide for Cost-Effective Document Analysis with Knowledge Graphs -Jaykumaran)

Traditional RAG vs GraphRAG

Feature	**Traditional RAG**	**GraphRAG**
Data Representation	Flat Vectors	Knowledge Graph
Query Scope	Local context	Global Reasoning
Scalability	Low	High
Citation Transparency	Low	High (Traceable sources)
Response Coherence	Fragmented	Relevant and Context-Rich

YHK

How GraphRAG Solves the Problem

- GraphRAG combines graph structures with vector search.
- Traverses multi-hop connections to infer relationships.
- Offers more reliable responses than vector-only RAG.

Graph-based RAG helps agentic AI make human-like decisions. – May Habib, CEO of Writer.com

YHK

Applications of GraphRAG

- **Healthcare:** Assists in diagnoses and treatment decisions.
- **Banking:** Detects fraudulent transactions using knowledge graphs.
- **Customer Service :** quickly answer customer questions from thousands of pages of policy documentation
- **Recommendations:** understand customer behavior and preferences better, to provide personalized services.
- **Supply Chain:** product recall and associated quality control checking, internal documentation search

YHK

Advantages of GraphRAG

- Structured knowledge representation.
- Context-aware and efficient retrieval.
- Handles complex queries effectively.
- Provides explainability and transparency.

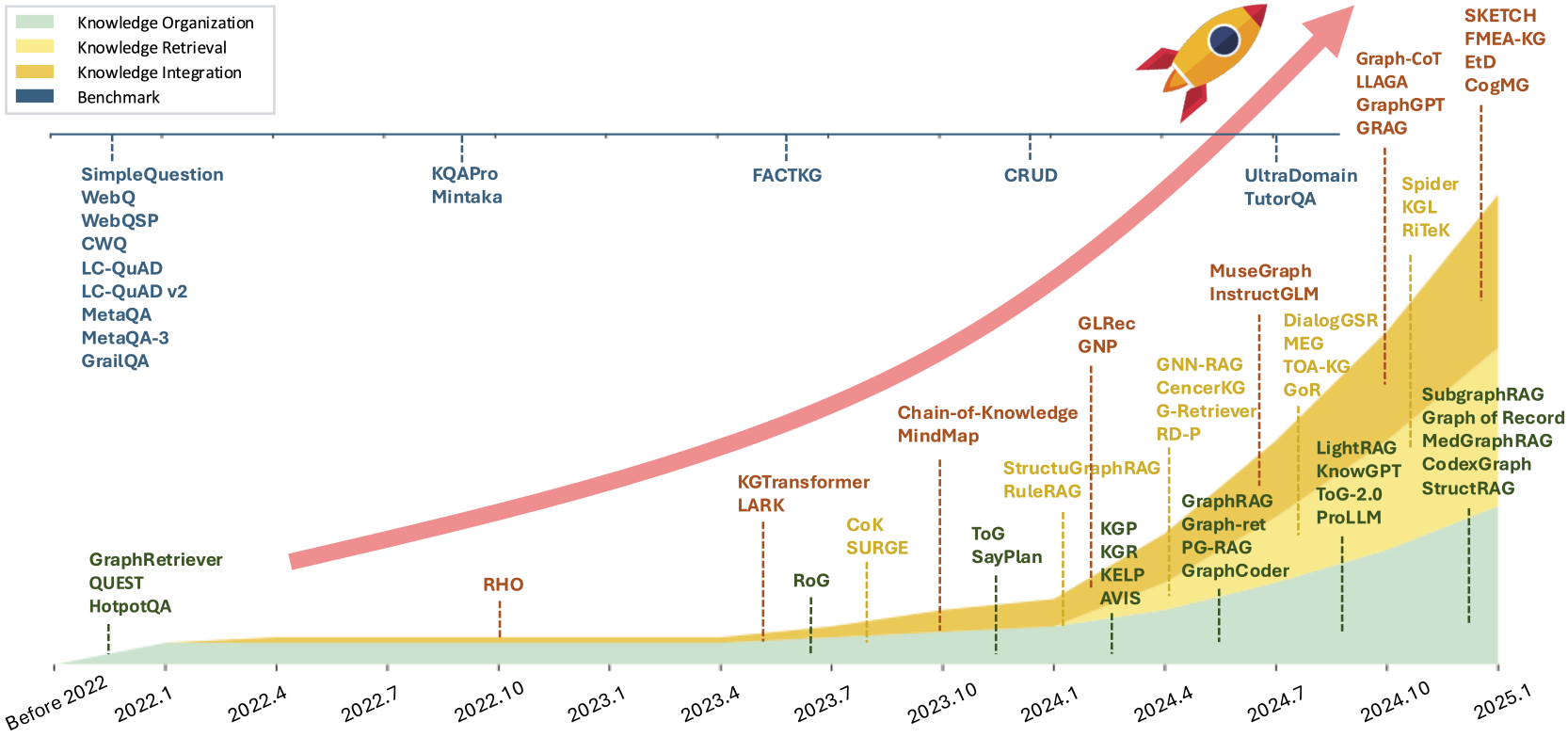
YHK

Challenges of GraphRAG

- **Complex Knowledge Graph Construction:** Requires sophisticated NLP techniques.
- **Data Dependency:** Performance relies on input data quality.
- **Scalability Issues:** Large graphs require significant computational resources.

YHK

Trend of GraphRAG Research



(Ref: Awesome-GraphRAG (GraphRAG Survey))

YHK

Conclusion

- GraphRAG enhances traditional RAG models using structured knowledge.
- Improves accuracy, context-awareness, and efficiency.
- Useful in various domains like healthcare and banking.
- A promising approach for future AI-powered knowledge retrieval.

YHK

Implementation using Microsoft Graph RAG

Source: Microsoft Research (https://microsoft.github.io/graphrag/) and GitHub repository (https://github.com/microsoft/graphrag)

YHK

Introduction to Microsoft's GraphRAG

- GraphRAG is Microsoft's framework for Retrieval-Augmented Generation on graphs
- Combines knowledge graphs with LLMs for enhanced context-aware responses
- Addresses limitations of traditional RAG by leveraging graph relationships
- Provides more structured, accurate, and explainable answers

YHK

GraphRAG: Key Concepts

- Knowledge Graph (KG): Structured representation of entities and relationships
- Graph Retrieval: Finding relevant graph sections for a given query
- Graph Reasoning: Using graph structure to enhance LLM reasoning
- Graph Augmentation: Dynamically enhancing the graph with new information
- End-to-end pipeline combining these components with LLM generation

YHK

GraphRAG Architecture

- Document Ingestion: Converting unstructured documents to graph format
- Graph Store: Neo4j or other graph databases for storing knowledge
- Embedding Engine: Creating vector representations of graph elements
- Query Processing: Converting natural language to graph queries
- Response Generation: Using retrieved subgraphs to augment LLM responses

YHK

Setting Up GraphRAG

- Install required dependencies
- Configure environment variables

```
pip install graphrag
pip install torch transformers neo4j

export NEO4J_URI="bolt://localhost:7687"
export NEO4J_USERNAME="neo4j"
export NEO4J_PASSWORD="password"
export OPENAI_API_KEY="your-key-here"
```

YHK

Document Ingestion Pipeline

Convert documents into graph-compatible format

```python
from graphrag import DocumentProcessor

processor = DocumentProcessor()
documents = processor.load_documents("path/to/docs")
chunks = processor.chunk_documents(documents)
entities = processor.extract_entities(chunks)
relations = processor.extract_relations(chunks, entities)
```

YHK

Building the Knowledge Graph

Populate Neo4j with extracted entities and relationships

```python
from graphrag import GraphBuilder

graph_builder = GraphBuilder(uri, username, password)
graph_builder.clear_database()  # Optional
graph_builder.create_schema()
graph_builder.add_entities(entities)
graph_builder.add_relationships(relations)
```

YHK

Knowledge Graph Embeddings

Generate embeddings for efficient graph retrieval

```
from graphrag import GraphEmbedder

embedder = GraphEmbedder(model="sentence-transformers/all-MiniLM-L6-v2")
embedder.generate_node_embeddings(graph_db)
embedder.generate_subgraph_embeddings(graph_db)
embedder.save_embeddings("embeddings.pkl")

```

YHK

Query Translation

Convert natural language queries to graph queries

```
from graphrag import QueryTranslator

translator = QueryTranslator(llm="gpt-4")
query = "Who funded the Manhattan Project?"
cypher_query = translator.to_cypher(query)
print(cypher_query)
# MATCH (p:Project {name: 'Manhattan Project'})-
# [:FUNDED_BY]->(f) RETURN f.name
```

YHK

Graph Retrieval Strategies

- Multiple retrieval strategies for different query types
- Direct query execution for structured data retrieval
- Semantic search for similarity-based retrieval

```
results = graph_db.execute_query(cypher_query)


from graphrag import SemanticGraphRetriever

retriever = SemanticGraphRetriever(embeddings)
relevant_nodes = retriever.retrieve(query, top_k=5)

```

YHK

Path-based Retrieval

Extract logical paths between relevant entities

```
from graphrag import PathRetriever

path_retriever = PathRetriever(graph_db)
paths = path_retriever.find_paths(
  start_node="Albert Einstein",
  end_node="Manhattan Project",
  max_length=3
)

```

YHK

Subgraph Extraction

Extract contextual subgraphs for complex queries

```
from graphrag import SubgraphExtractor

extractor = SubgraphExtractor(graph_db)
subgraph = extractor.extract_neighborhood(
  seed_nodes=relevant_nodes,
  depth=2
)

```

Graph Reasoning

Apply graph algorithms to infer additional information

```
from graphrag import GraphReasoner

reasoner = GraphReasoner(graph_db)
inferences = reasoner.infer_relations(subgraph)
enhanced_graph = reasoner.enhance_subgraph(
  subgraph,
  inferences
)

```

YHK

Response Generation

Generate responses using retrieved subgraphs

```
from graphrag import GraphRAGGenerator

generator = GraphRAGGenerator(
  llm="gpt-4",
  prompt_template="Based on the graph: {graph_context}\n\nQuestion:
    {query}\n\nAnswer:"
)

response = generator.generate(
  query=query,
  graph_context=enhanced_graph.to_text()
)
print(response)
```

YHK

End-to-End GraphRAG Pipeline

Complete pipeline from query to response

```
from graphrag import GraphRAGPipeline

pipeline = GraphRAGPipeline(
  graph_db=graph_db,
  retriever=retriever,
  reasoner=reasoner,
  generator=generator
)

answer = pipeline.run(query="What was Einstein's role in the Manhattan
    Project?")
print(answer)

```

YHK

Advanced Features: Query Decomposition

Break complex queries into simpler sub-queries

```python
from graphrag import QueryDecomposer

decomposer = QueryDecomposer(llm="gpt-4")
complex_query = "How did Einstein's theories influence nuclear weapons
    development?"
sub_queries = decomposer.decompose(complex_query)
# ['What theories did Einstein develop?',
#  'How do these theories relate to nuclear physics?',
#  'How were these principles applied in weapons development?']

```

YHK

Advanced Features: Graph Augmentation

Dynamically enhance graph with new information

```python
from graphrag import GraphAugmenter

augmenter = GraphAugmenter(
  graph_db=graph_db,
  llm="gpt-4"
)

new_entities = augmenter.extract_missing_entities(
  query=query,
  context=retrieved_text
)
graph_db.add_entities(new_entities)
```

YHK

Advanced Features: Explainability

Generate explanations for responses based on graph paths

```python
from graphrag import ExplainableGraphRAG

explainable_rag = ExplainableGraphRAG(pipeline)
result = explainable_rag.generate(
  query=query,
  explain=True
)
print(f"Answer: {result['answer']}")
print(f"Explanation: {result['explanation']}")
print(f"Evidence paths: {result['evidence_paths']}")
```

YHK

Evaluation & Benchmarking

Evaluate GraphRAG performance against standard benchmarks

```python
from graphrag.evaluation import Evaluator

evaluator = Evaluator(
  pipeline=pipeline,
  metrics=["accuracy", "relevance", "faithfulness"]
)

results = evaluator.evaluate(
  benchmark_dataset="path/to/questions.json"
)
print(f"Accuracy: {results['accuracy']}")
print(f"Relevance: {results['relevance']}")
```

YHK

Case Studies & Applications

- Enterprise knowledge management
- Scientific literature analysis
- Customer support and troubleshooting
- Regulatory compliance and legal research
- Healthcare and biomedical information systems
- Financial analysis and risk assessment

YHK

References

Slides primarily borrowed from . . .

- End-to-End Implementation of GraphRAG(Knowledge Graphs + Retrieval Augmented Generation): Unlocking LLM Discovery on Narrative Private Data - Vinod Kumar G R
- GraphRAG tutorials by Data Science in your pocket
- GraphRAG: Enhancing LLMs with Knowledge Graphs - Vijayakumar Ramdoss
- Awesome-GraphRAG (GraphRAG Survey) https://github.com/DEEP-PolyU/Awesome-GraphRAG
- Build your hybrid-Graph for RAG & GraphRAG applications using the power of NL - Irina Adamchic
- The GenAI Stack - Andreas Kollegger - Neo4j
- GraphRAG: The Practical Guide for Cost-Effective Document Analysis with Knowledge Graphs JaykumaranJaykumaran

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

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

YHK