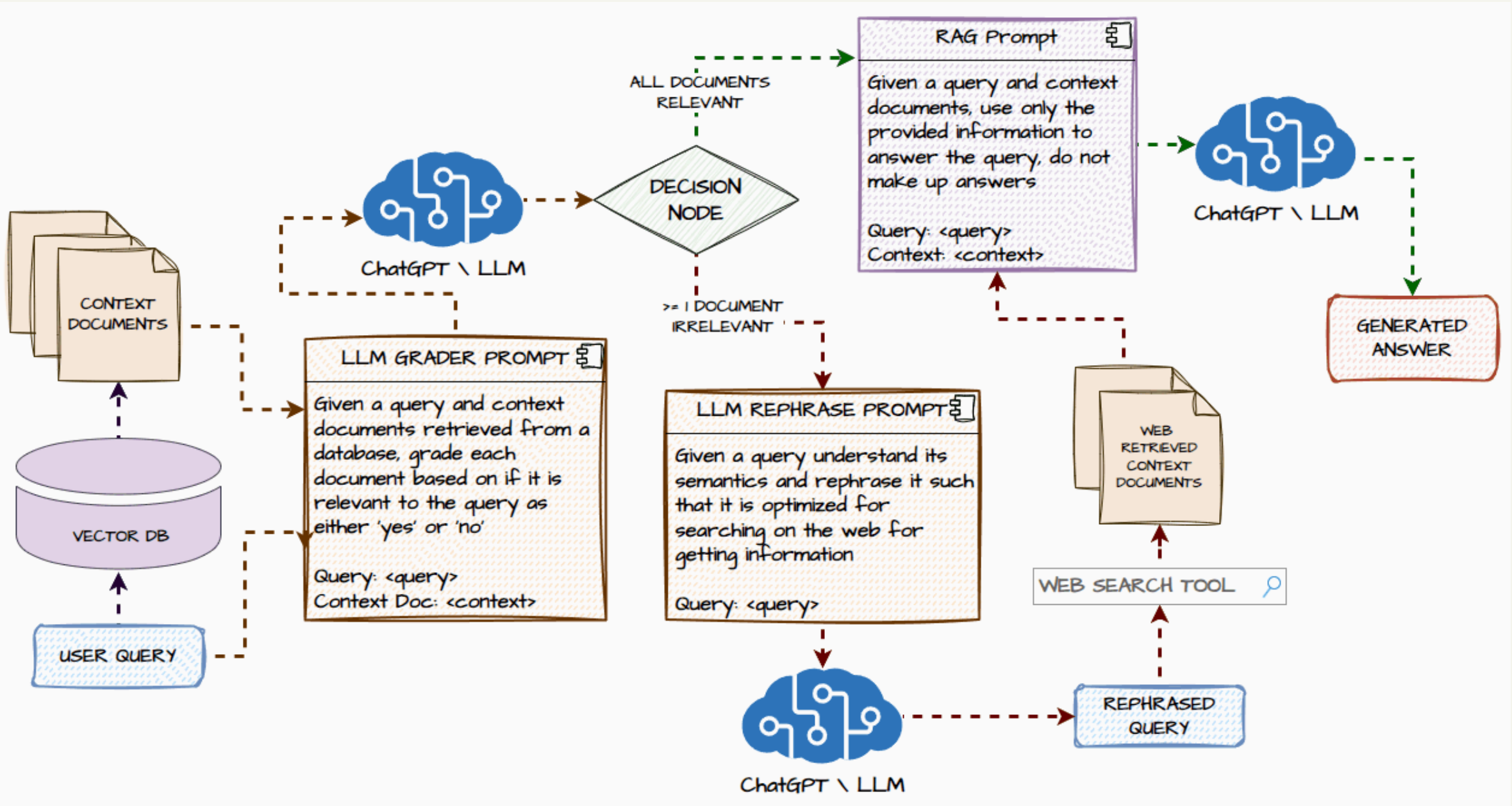
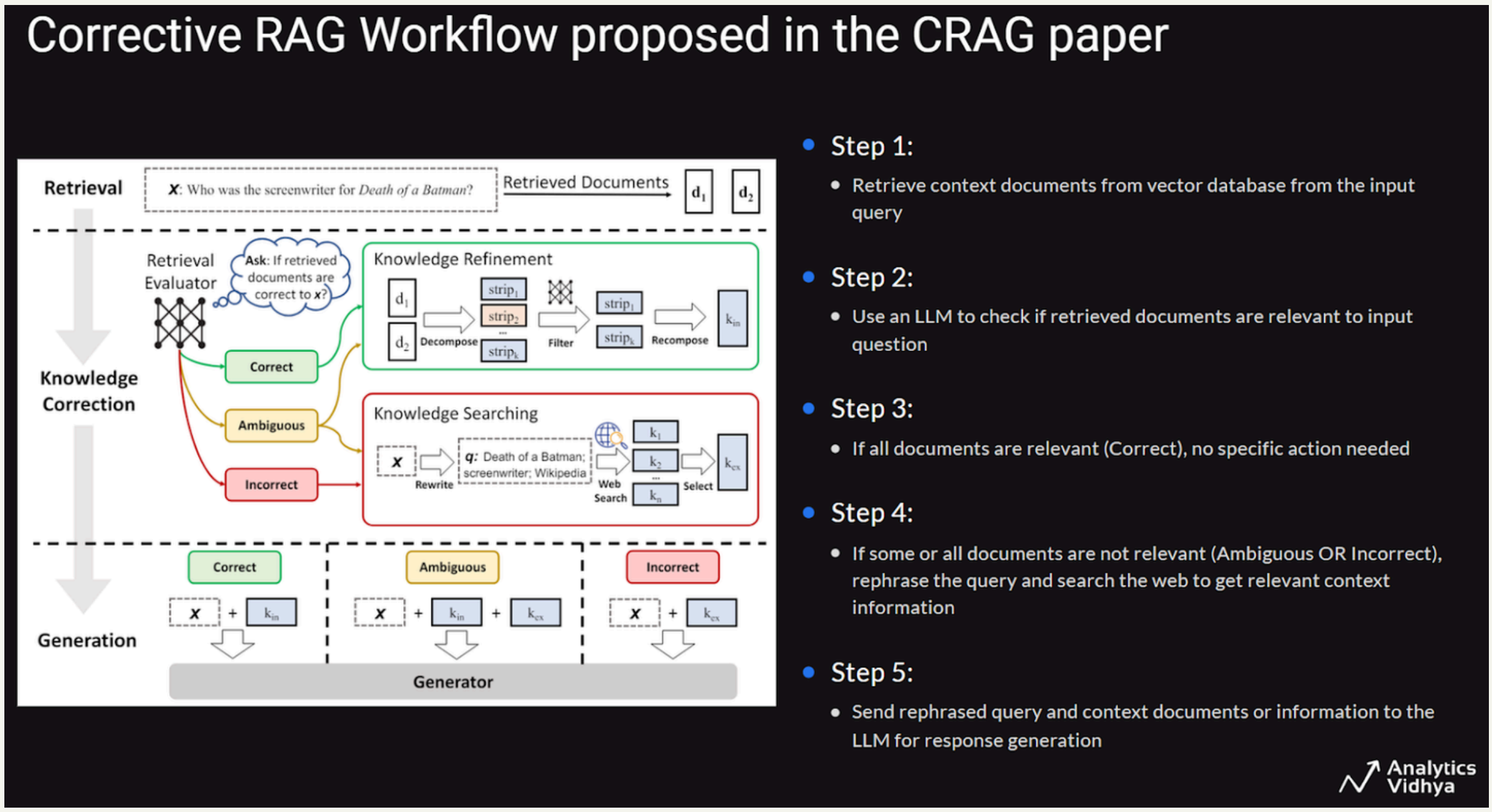
Hands-on Guide to Agentic Corrective RAG Systems



Dipanjan (DJ)

Corrective RAG System



- **The inspiration for our agentic RAG system will be based on the solution proposed in the paper, Corrective Retrieval Augmented Generation, Yan et al. , where they propose a workflow as depicted in the following figure to enhance a regular RAG system.**

- **The key idea here is to retrieve document chunks from the vector database as usual and then use an LLM to check if each retrieved document chunk is relevant to the input question.**

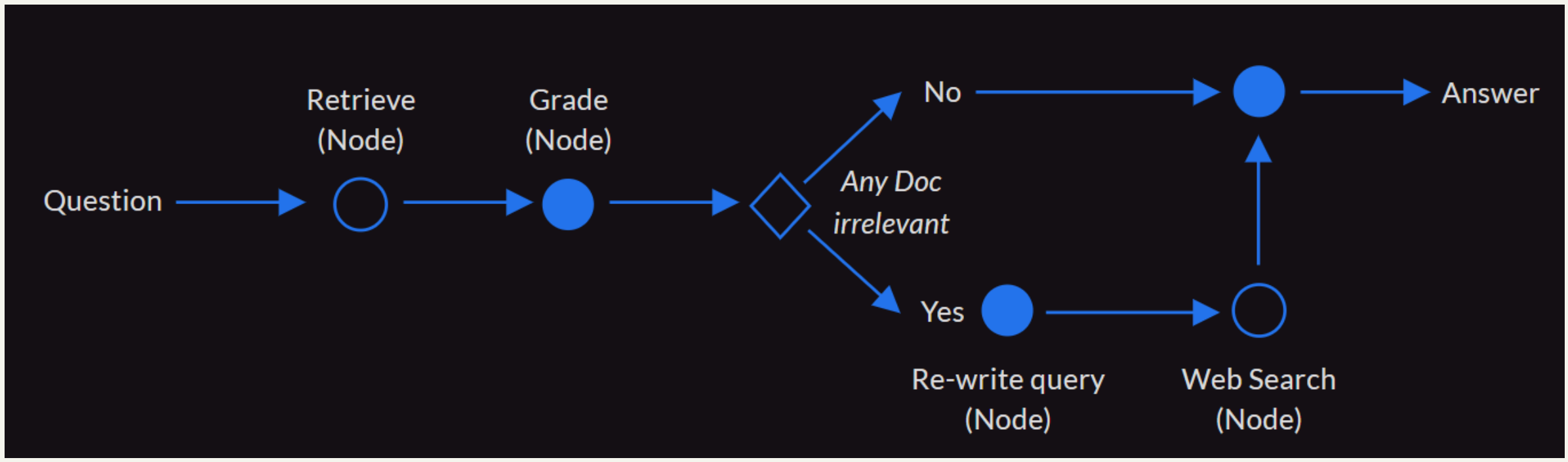
- **If all the retrieved document chunks are relevant, then it goes to the LLM for a normal response generation like a standard RAG pipeline.**

- **However, if some retrieved documents are not relevant to the input question, we rephrase the input query, search the web to retrieve new information, and send it to the LLM to generate a response.**

Source: A Comprehensive Guide to Building Agentic RAG Systems with LangGraph

Created by: Dipanjan (DJ)

Agentic Corrective RAG System Workflow



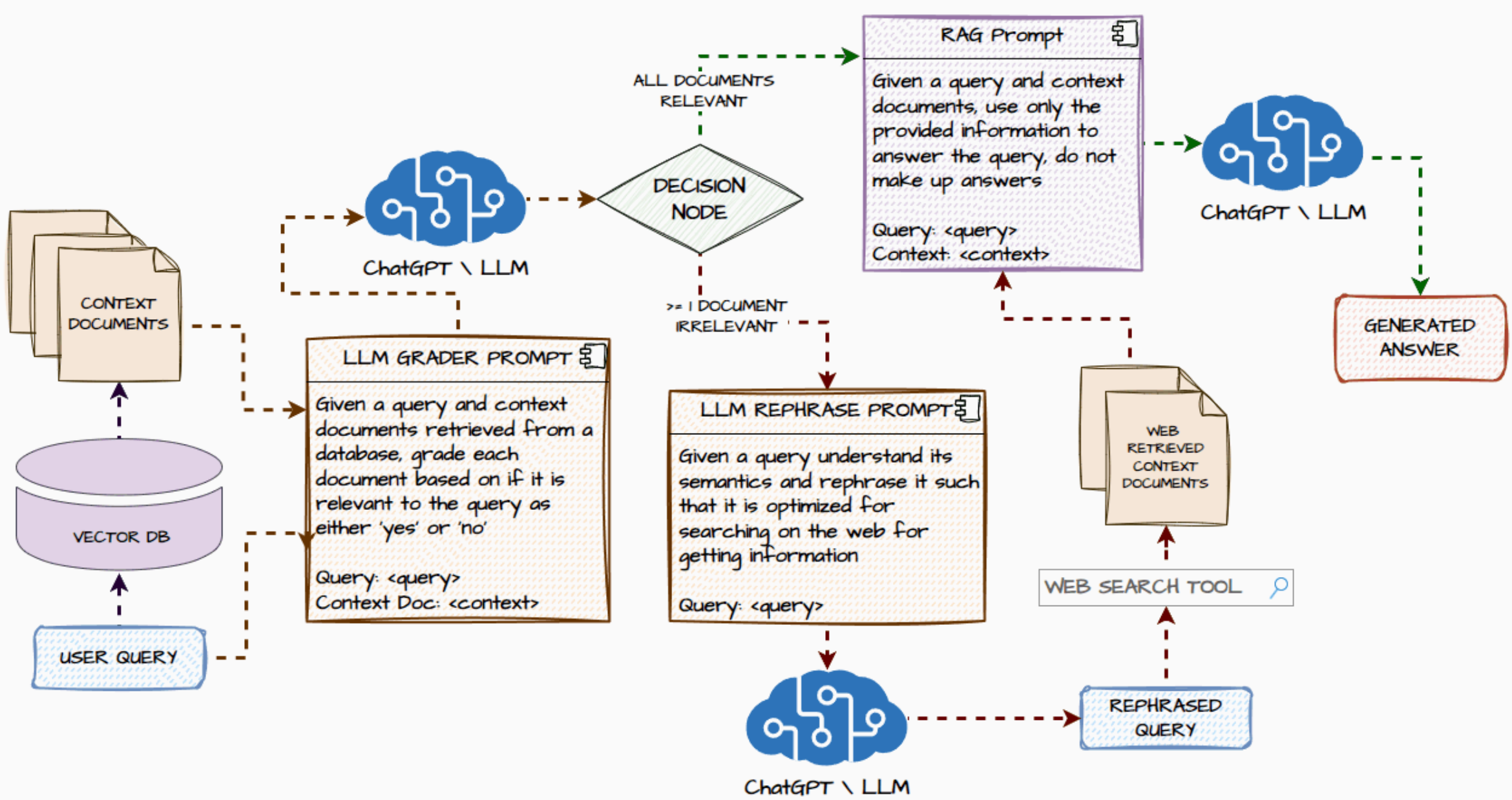
- **Main Flow**
  - **Step 1 - Retrieve Node**
    - Retrieves context documents from the vector database from the input query.
  - **Step 2 - Grade Node**
    - Use an LLM to grade if retrieved documents are relevant to the input question – yes or no.
  - **Step 3A - Generate Answer Node**
    - If all documents are relevant (all 'yes'), send them to an LLM for response generation.

- **Alternative Flow (if any retrieved documents are not relevant)**
  - **Step 3B - Rewrite Query Node**
    - If some or all documents are not relevant (at least one 'no'), rephrase the query.
  - **Step 3C - Web Search Node**
    - Search the web to get context information using the rephrased query.
  - **Step 3D - Generate Answer Node**
    - Send the rephrased query and context documents or information to the LLM for response generation.

Source: A Comprehensive Guide to Building Agentic RAG Systems with LangGraph

Created by: Dipanjan (DJ)

Detailed Agentic Corrective RAG System Architecture



- **Query and Context Retrieval**
  - User query sent to vector DB (e.g., Chroma) to retrieve context documents.
  - If no documents are retrieved, proceed to rephrase the query.
- **Document Grading**
  - LLM grades retrieved documents as 'Yes' (relevant) or 'No' (irrelevant).
- **Decision Node**
  - All Documents Relevant: Follow standard RAG flow; send query and documents to LLM for response.
  - Irrelevant/No Documents: Rephrase query using LLM for optimized web search.
- **Web Search**
  - Use web search tool (e.g., Tavily) to retrieve additional context documents.
- **Response Generation**
  - Send combined context documents and query to LLM to generate the final response.

Source: A Comprehensive Guide to Building Agentic RAG Systems with LangGraph

Created by: Dipanjan (DJ)

Hands-on Guide

Analytics Vidhya

Free Courses | Learning Paths | GenAI Pinnacle Program | Agentic AI Pioneer Program (New)

Interview Prep | Career | GenAI | Prompt Engg | ChatGPT | LLM | Langchain | RAG | AI Agents | Machine Learning | Deep Learning | GenAI Tools | LLMOps | Python | NLP

Home › LLMs › A Comprehensive Guide to Building Agentic RAG Systems with LangGraph

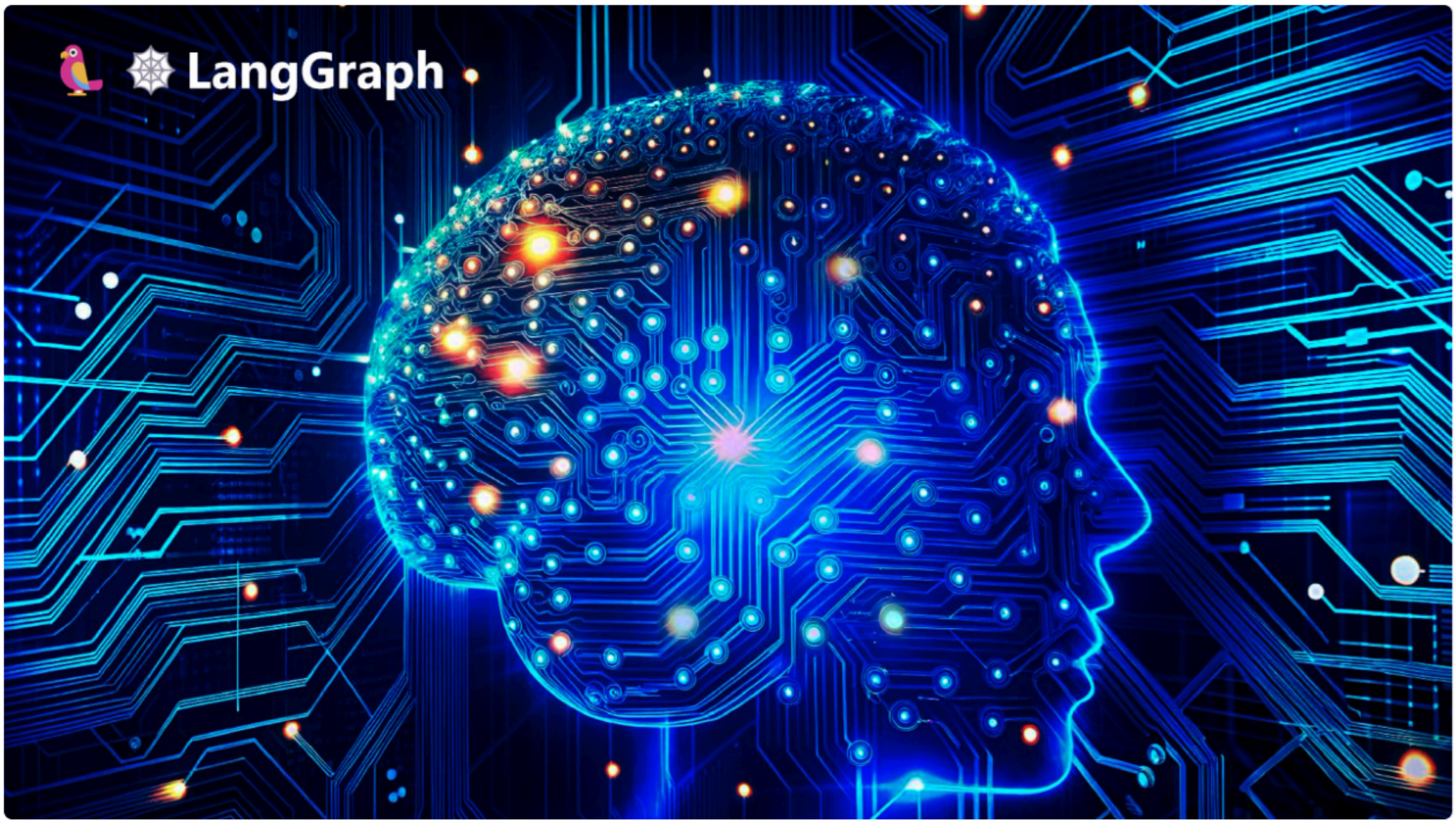
A Comprehensive Guide to Building Agentic RAG Systems with LangGraph

Dipanjan (DJ) Sarkar
Last Updated : 11 Sep, 2024

20 min read | 204

Introduction

Retrieval Augmented Generation systems, better known as RAG systems, have quickly become popular for building Generative AI assistants on custom enterprise data. They avoid the hassles of expensive fine-tuning of Large Language Models (LLMs). One of the key advantages of RAG systems is you can easily integrate your data, augment your LLM's intelligence, and give more contextual answers to your questions. However, a whole set of problems can make RAG systems underperform and, worse, give wrong answers to your questions! In this guide, we will look at a way to see how AI Agents can augment the capabilities of a traditional RAG system and improve on some of its limitations.



CHECK OUT THE HANDS-ON GUIDE HERE