RAG (Retrieval Augmented Generation) Cheatsheet

Stages in RAG:

1. Loading:
   - Import your data (text files, PDFs, databases, APIs) using LlamaHub's extensive range of connectors.
2. Indexing:
   - Create searchable data structures, primarily through vector embeddings and metadata strategies, enabling efficient context retrieval.
3. Storing:
   - Securely store your indexed data and metadata for quick access without the need to re-index.
4. Querying:
   - Utilize LLMs and LlamaIndex data structures for diverse querying techniques, including sub-queries and hybrid strategies.
5. Evaluation:
   - Continuously assess the effectiveness of your pipeline to ensure accuracy, faithfulness, and response speed.

Application Types:

1. Query Engines:
   - For direct question-answering over your data.
2. Chat Engines:
   - Enables conversations with your data for an interactive experience.
3. Agents:
   - Automated decision-makers that interact with external tools, adaptable for complex tasks.

Key Concepts:

1. Nodes and Documents:
   Fundamental units in LlamaIndex, where Documents encapsulate data sources and Nodes represent data "chunks" with associated metadata.
1. Connectors:
   Bridge various data sources into the RAG framework, transforming them into Nodes and Documents.
1. Indexes:
   The backbone of RAG, enabling the storage of vector embeddings in a vector store along with crucial metadata.
1. Embeddings:
   Numerical representations of data, facilitating the relevance filtering process.
1. Retrievers:
   Define efficient retrieval strategies, ensuring the relevancy and efficiency of data retrieval.
1. Routers:
   Manage the selection of appropriate retrievers based on query specifics and metadata.
1. Node Postprocessors:
   Apply transformations or re-ranking logic to refine the set of retrieved nodes.
1. Response Synthesizers:
   Craft responses from the LLM, utilizing user queries and retrieved text chunks for enriched answers.

LlamaIndex SingleStore LangChain OpenAI

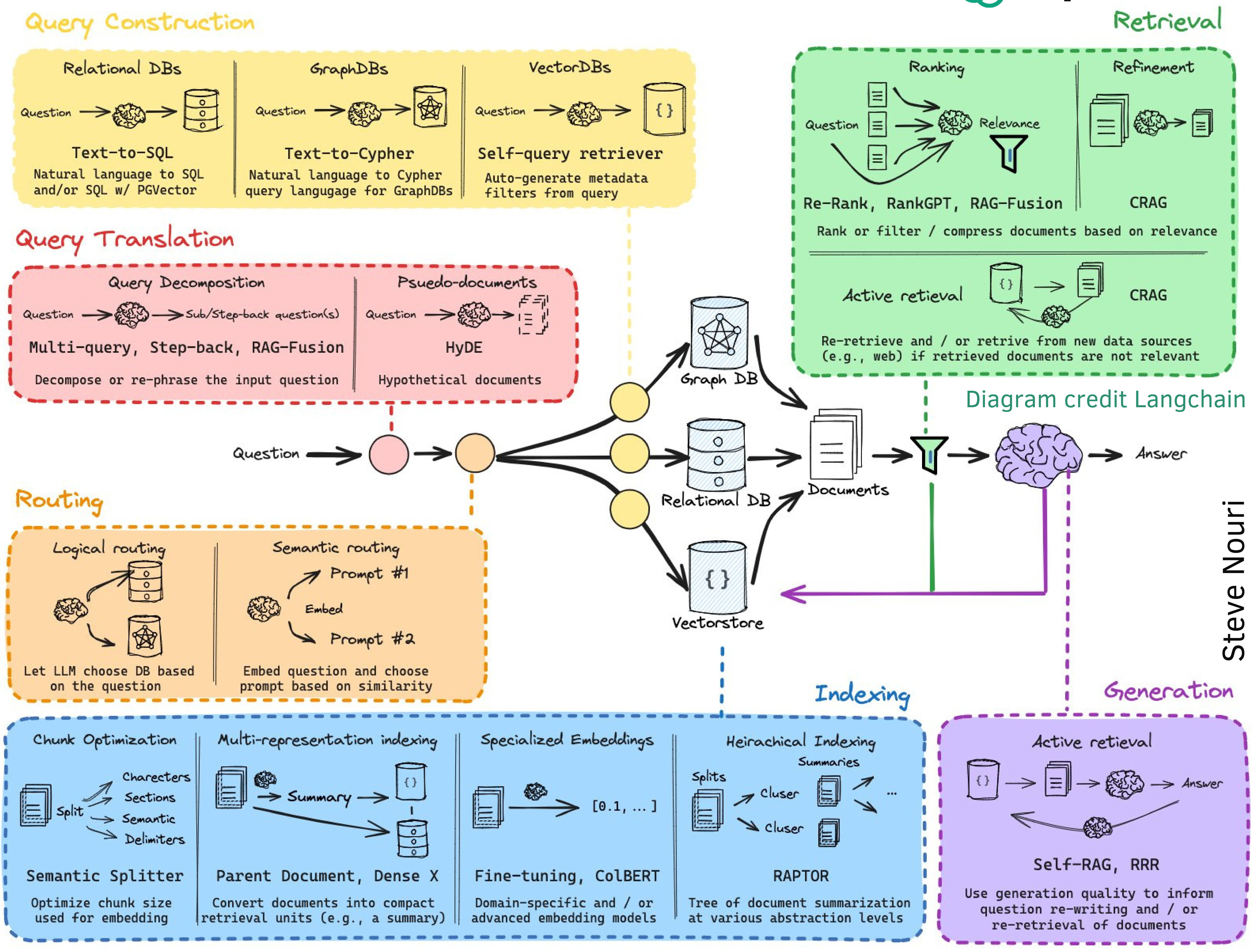
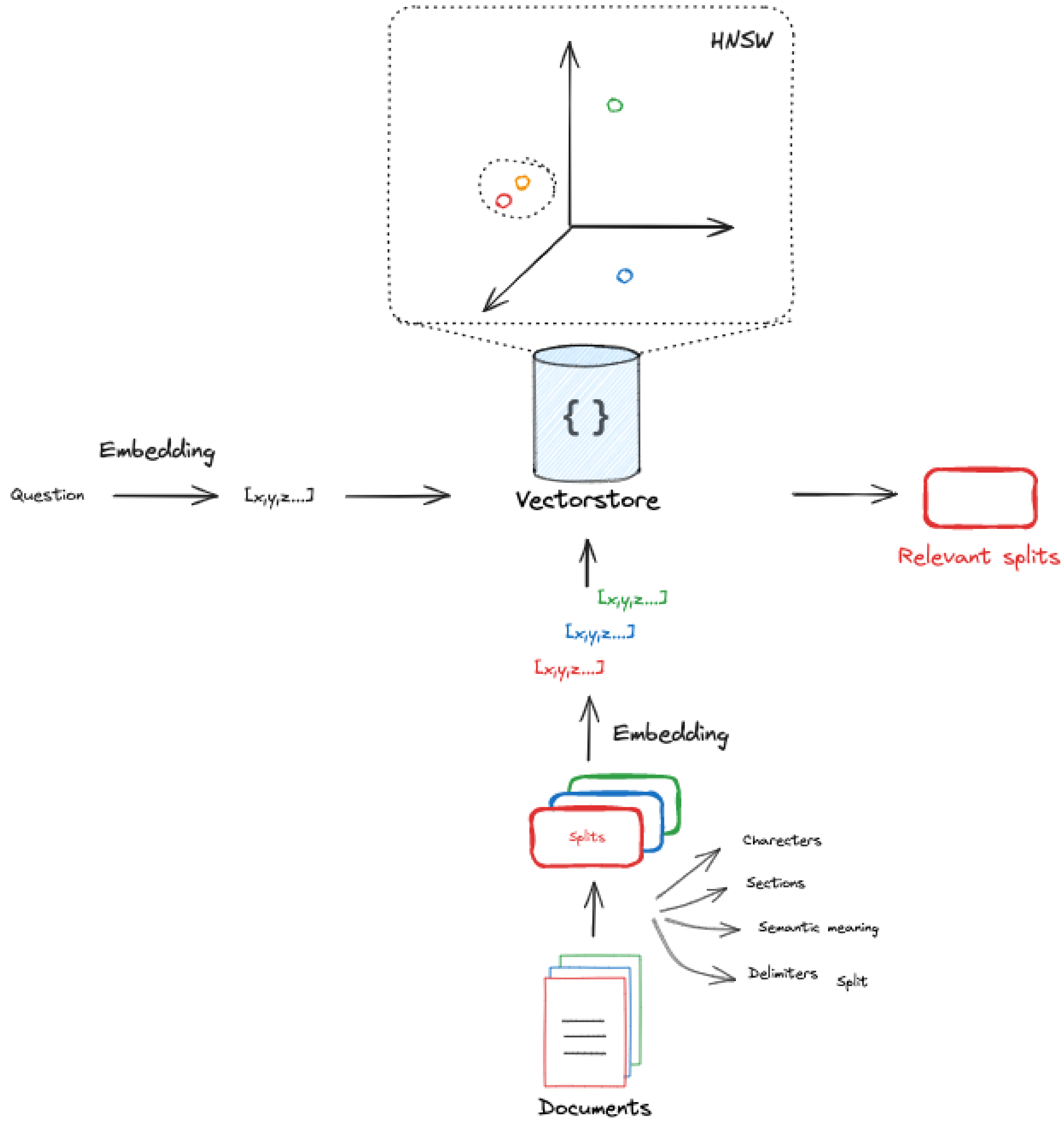


Diagram credit Langchain

Steve Nouri

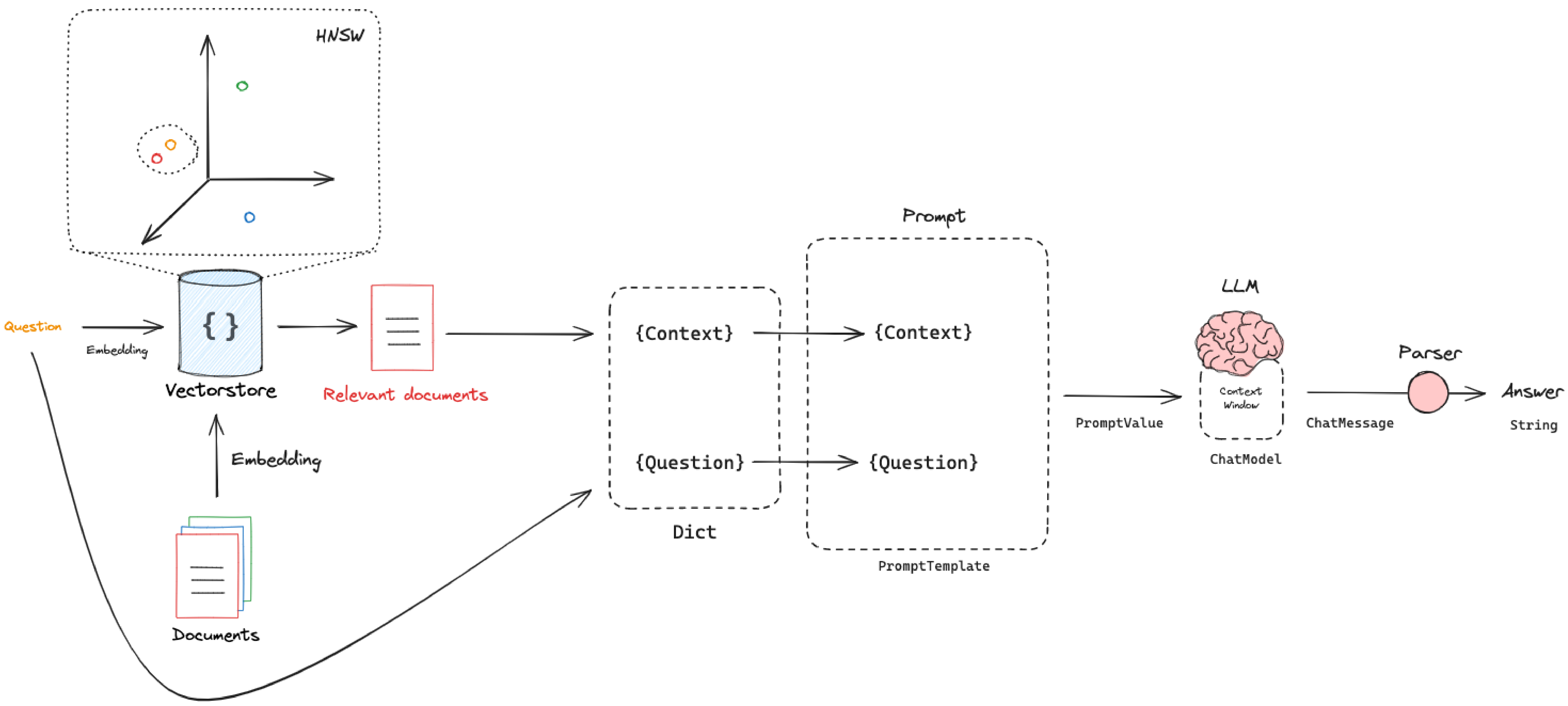
RAG (Retrieval Augmented Generation) Cheatsheet

Indexing:



https://github.com/langchain-ai/rag-from-scratch/blob/main/rag_from_scratch_1_to_4.ipynb

Generation:

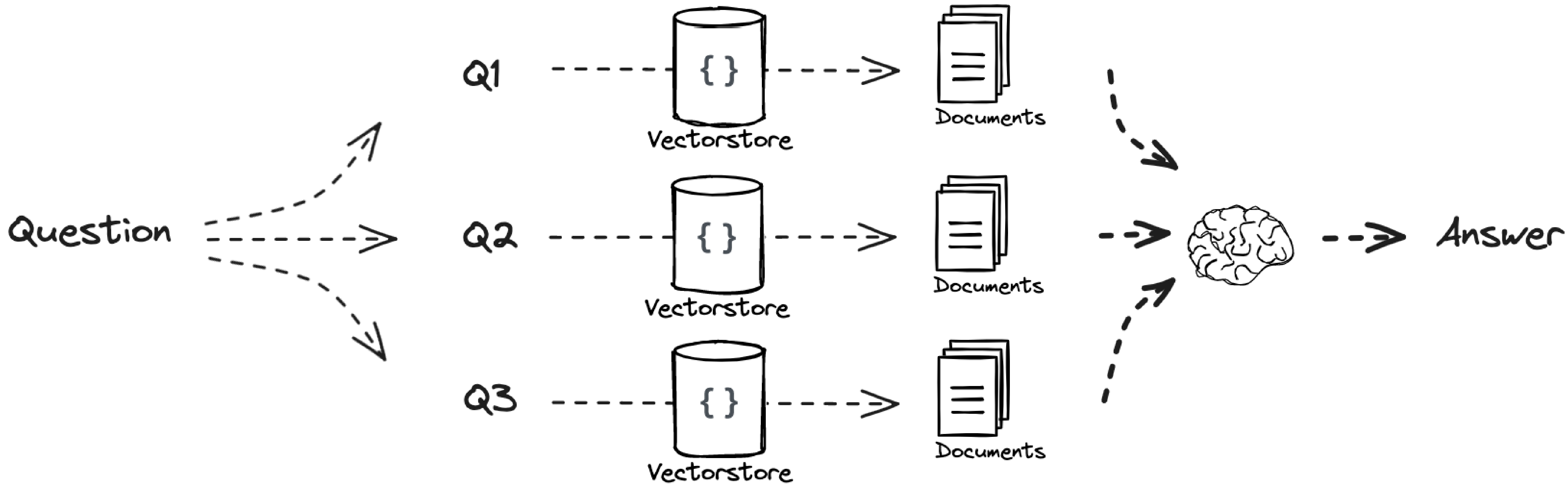


https://github.com/langchain-ai/rag-from-scratch/blob/main/rag_from_scratch_1_to_4.ipynb

Steve Nouri

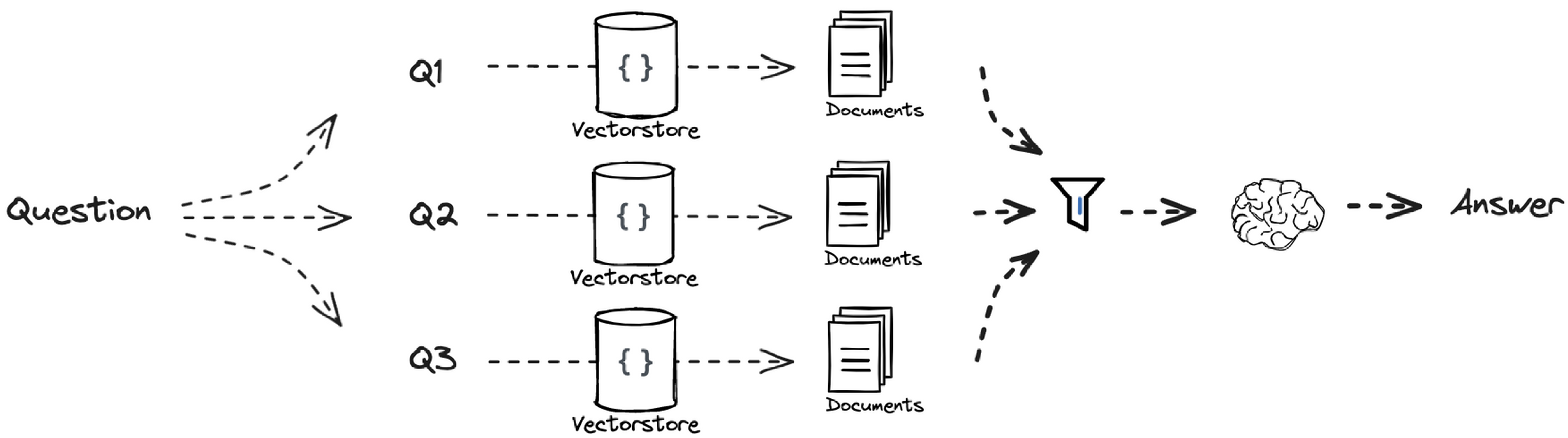
RAG (Retrieval Augmented Generation) Cheatsheet

Multi Query:



https://python.langchain.com/docs/modules/data_connection/retrievers/MultiQueryRetriever

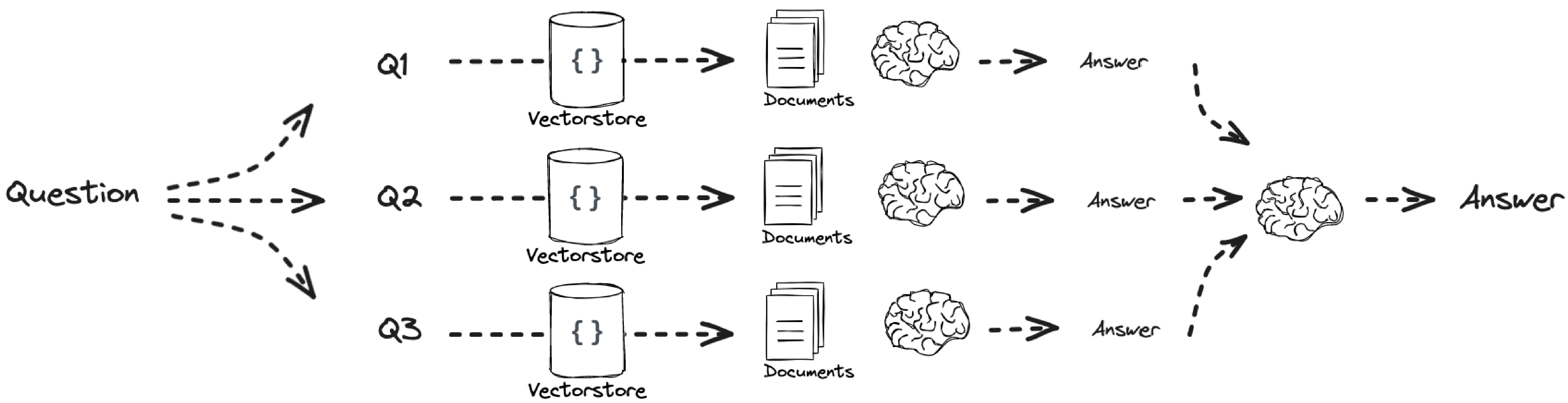
RAG-Fusion:



https://github.com/langchain-ai/langchain/blob/master/cookbook/rag_fusion.ipynb

Steve Nouri

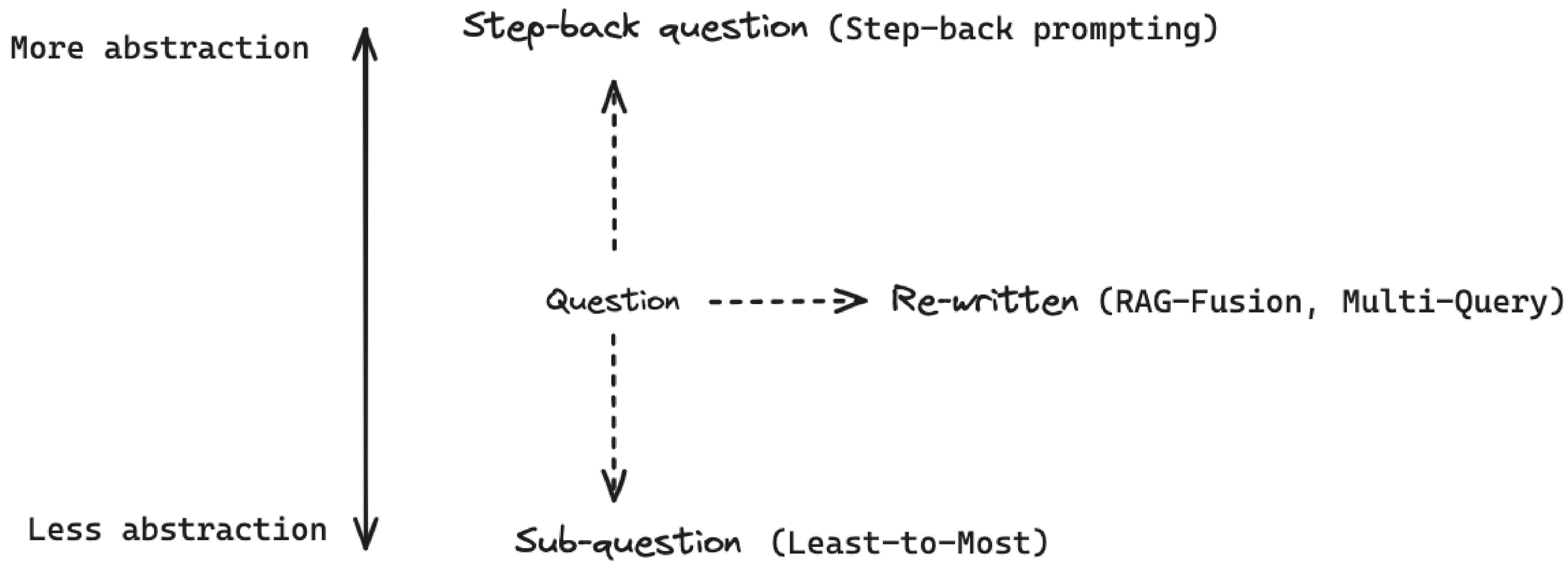
Decomposition:



https://arxiv.org/pdf/2205.10625.pdf

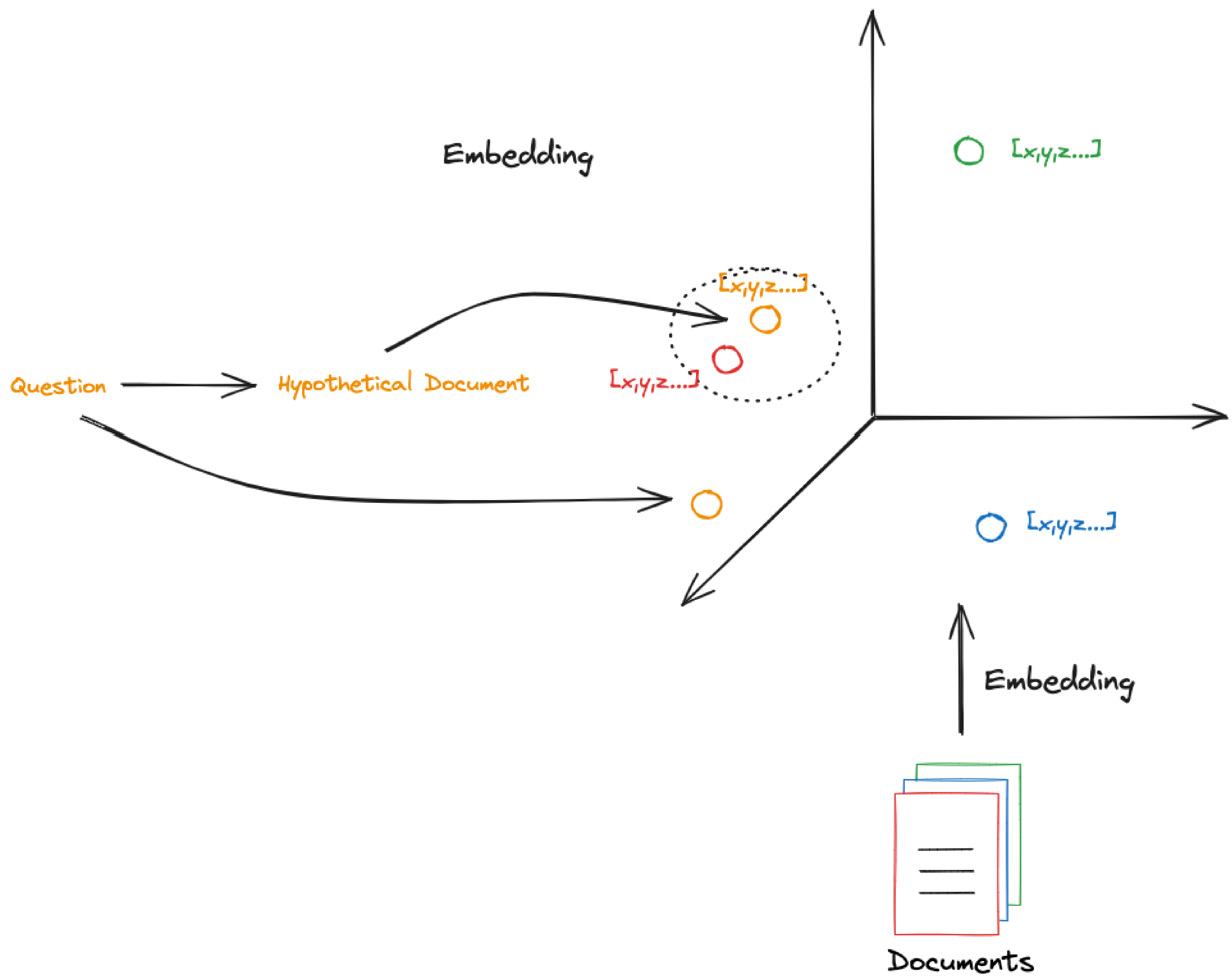
RAG (Retrieval Augmented Generation) Cheatsheet

Step Back:



https://arxiv.org/pdf/2310.06117.pdf

HyDE:



https://arxiv.org/abs/2212.10496

Steve Nouri

RAG (Retrieval Augmented Generation) Cheatsheet

Techniques and Tools:

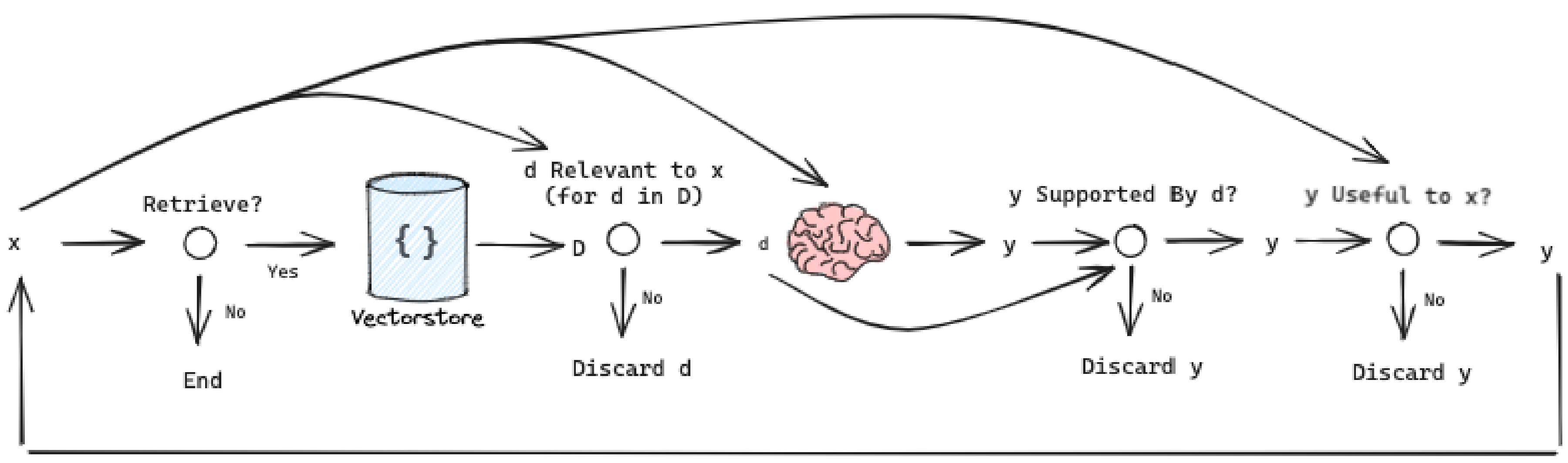
1. Data Ingestion and Querying:
   - Using tools like LlamaIndex for processing and querying data from various sources into the model's prompt.
2. Chunk Size Optimization:
   - Adjusting the size of data chunks for efficient processing and retrieval, improving response quality.
3. Metadata Filtering:
   - Enhancing retrieval by adding structured context to data, utilizing vector database capabilities for more relevant results.
4. Fine-Tuning Embeddings:
   - Customizing embedding models to better match query context with relevant data, improving precision and recall.
5. Advanced Retrieval Algorithms:
   - Implementing sophisticated retrieval methods like recursive retrieval and parent-child chunk retrieval to enhance context understanding and response accuracy.

Challenges and Solutions:

- Missing Data:
  Addressed by expanding the document corpus or integrating external knowledge bases.
- The issue with Ranking:
  Overcome by using advanced retrieval techniques like rerankers.
- Consolidation Issues:
  Solved by employing strategies that ensure relevant documents are included in the final context.
- Formatting Issues:
  Addressed by ensuring the system correctly interprets and responds to format-specific queries.
- Incorrect Specifics and Incomplete Answers:
  Mitigated by adjusting the detail level of responses to match user queries.
- Extraction Challenges:
  Overcome by refining the system's ability to accurately extract information from the selected context.

Self-RAG

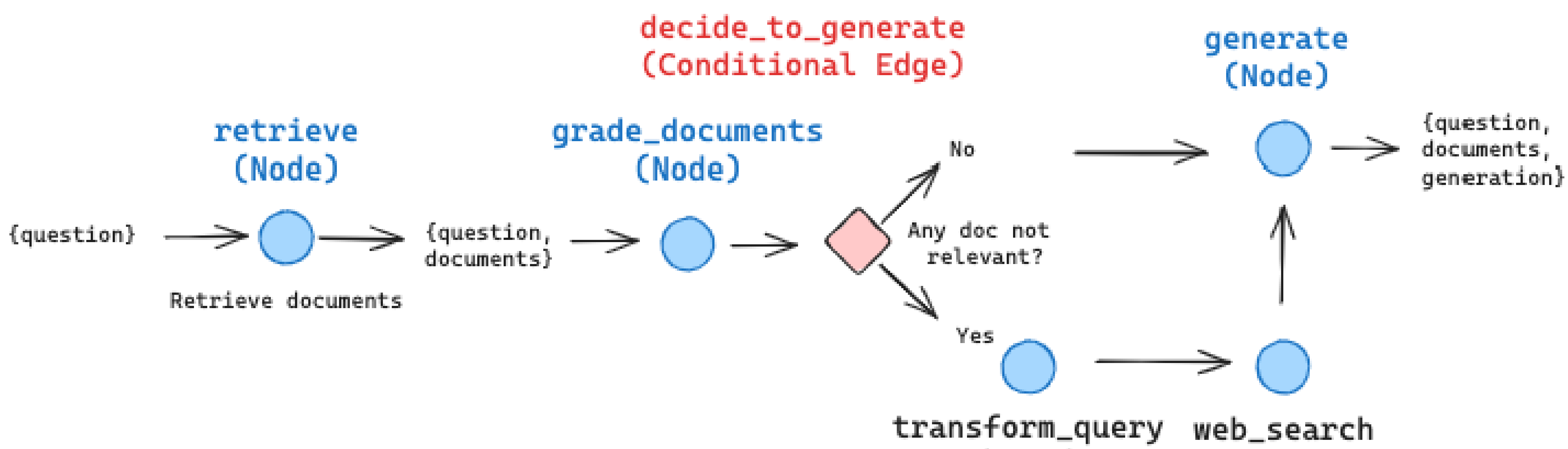
Self-reflection can enhance RAG, enabling correction of poor quality retrieval or generations.



https://arxiv.org/abs/2310.11511

Corrective RAG

Corrective-RAG (CRAG) is a recent paper that introduces an interesting approach for self-reflective RAG.



https://arxiv.org/pdf/2401.15884.pdf

Steve Nouri