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.

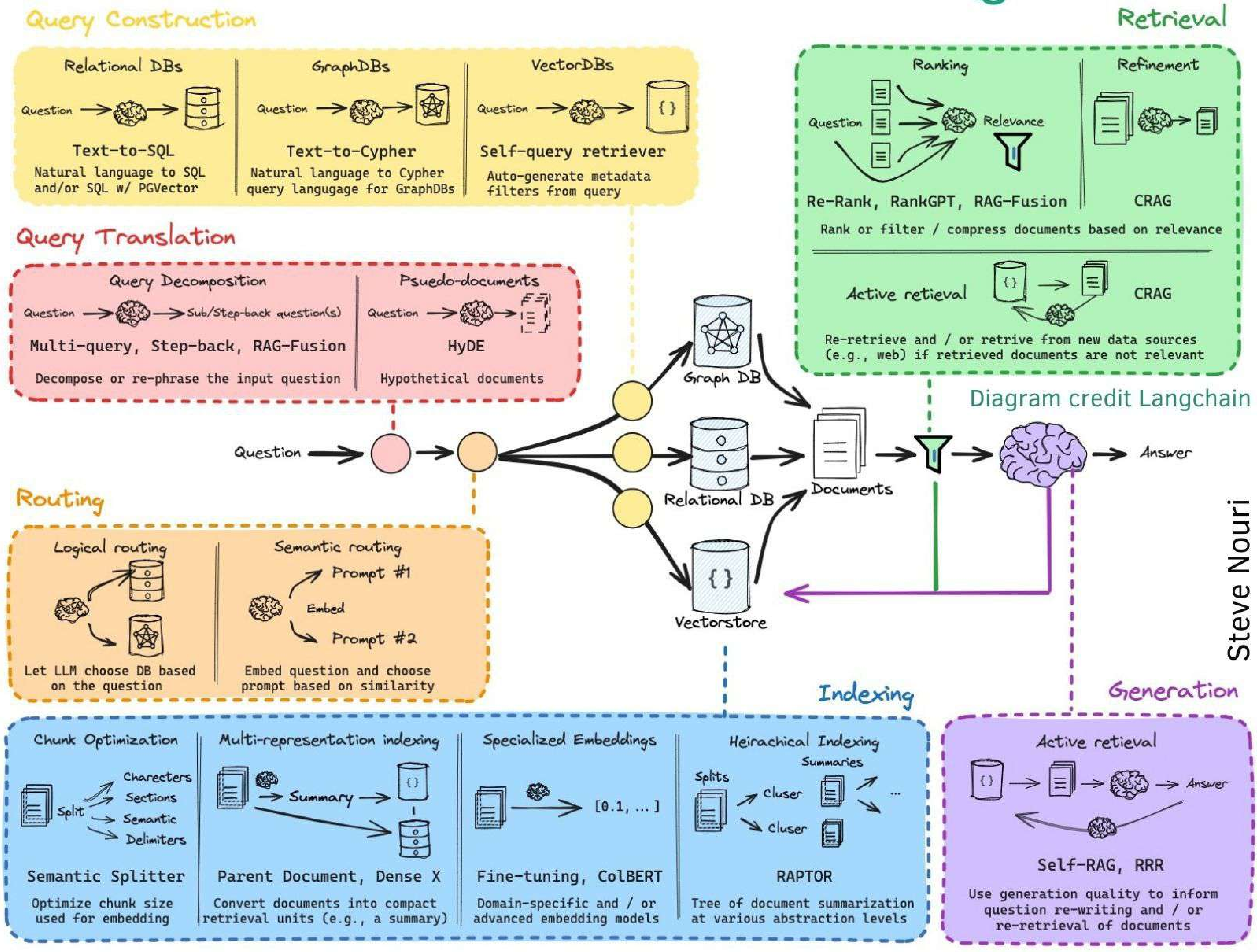
1. **Retrievers:**
Define efficient retrieval strategies, ensuring the relevancy and efficiency of data retrieval.

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



Query Construction

Relational DBs — Text-to-SQL: Natural language to SQL and/or SQL w/ PGVector

GraphDBs — Text-to-Cypher: Natural language to Cypher query langugage for GraphDBs

VectorDBs — Self-query retriever: Auto-generate metadata filters from query

Query Translation

Query Decomposition — Multi-query, Step-back, RAG-Fusion: Decompose or re-phrase the input question

Psuedo-documents — HyDE: Hypothetical documents

Routing

Logical routing — Let LLM choose DB based on the question

Semantic routing — Embed question and choose prompt based on similarity

Retrieval

Ranking — Re-Rank, RankGPT, RAG-Fusion

Refinement — CRAG

Rank or filter / compress documents based on relevance

Active retieval — CRAG: Re-retrieve and / or retrive from new data sources (e.g., web) if retrieved documents are not relevant

Indexing

Chunk Optimization — Semantic Splitter: Optimize chunk size used for embedding

Multi-representation indexing — Parent Document, Dense X: Convert documents into compact retrieval units (e.g., a summary)

Specialized Embeddings — Fine-tuning, ColBERT: Domain-specific and / or advanced embedding models

Heirachical Indexing — RAPTOR: Tree of document summarization at various abstraction levels

Generation

Active retieval — Self-RAG, RRR: Use generation quality to inform question re-writing and / or re-retrieval of documents

Diagram credit Langchain

Steve Nouri