# RAG from scratch: Query Translation (Decomposition)
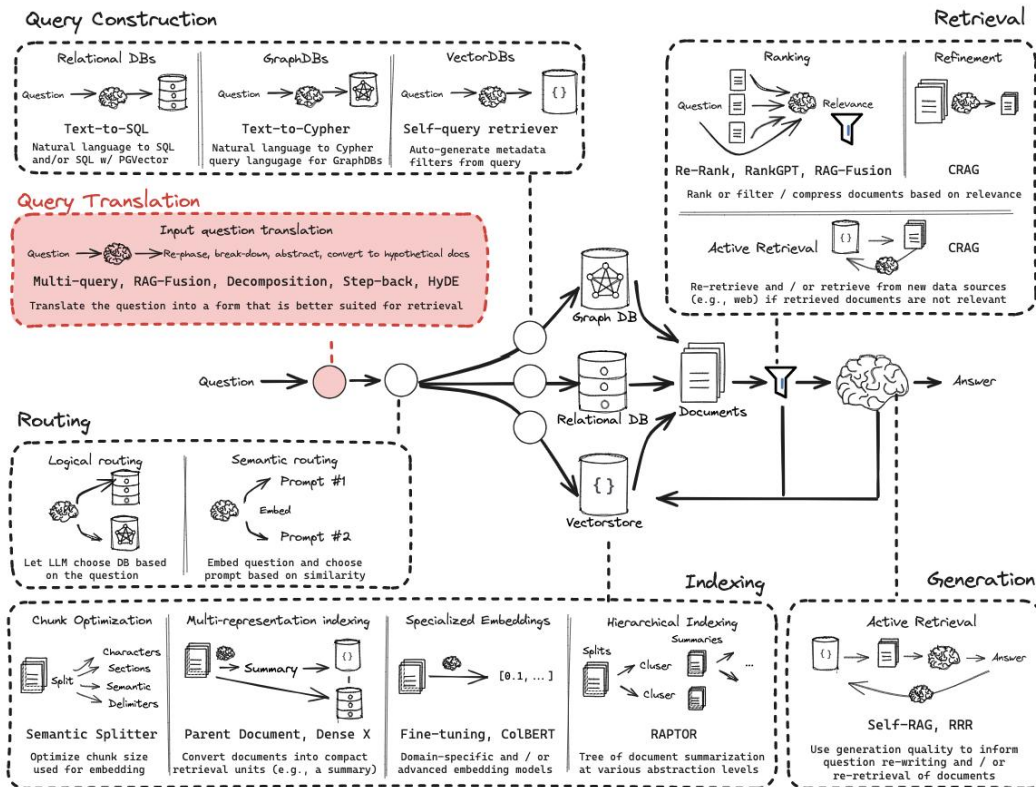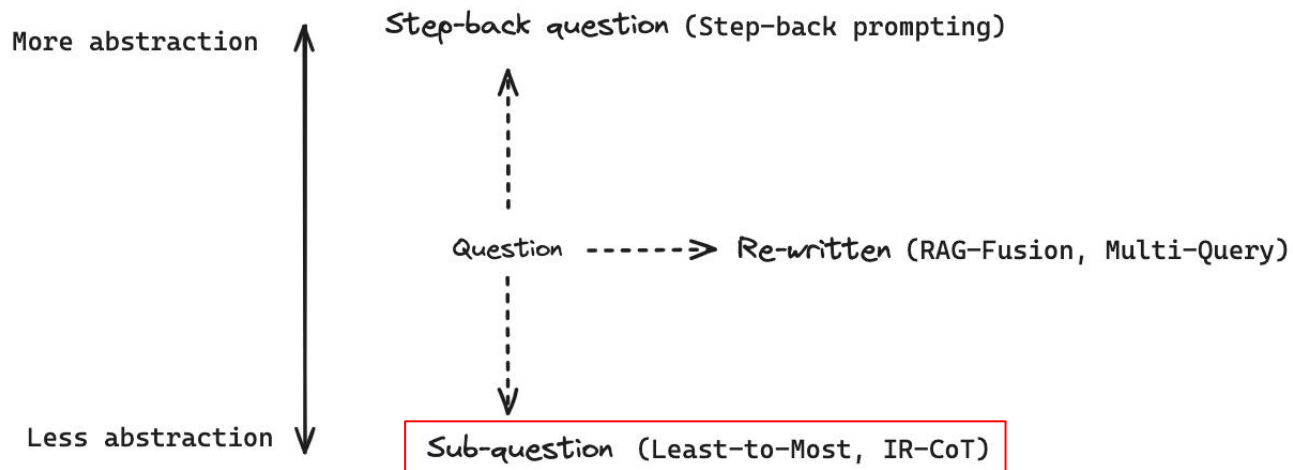
Lance Martin
Software Engineer, LangChain
@RLanceMartin

# Query Translation

## Query Construction

**Relational DBs**

Question → 🧠 → 🗄️

**Text-to-SQL**
Natural language to SQL and/or SQL w/ PGVector

**GraphDBs**

Question → 🧠 → ⭐

**Text-to-Cypher**
Natural language to Cypher query language for GraphDBs

**VectorDBs**

Question → 🧠 → {}

**Self-query retriever**
Auto-generate metadata filters from query

## Query Translation

**Input question translation**

Question → 🧠 → Re-phase, break-down, abstract, convert to hypothetical docs

**Multi-query, RAG-Fusion, Decomposition, Step-back, HyDE**

Translate the question into a form that is better suited for retrieval

## Routing

**Logical routing**

🧠 → 🗄️
→ ⭐

Let LLM choose DB based on the question

**Semantic routing**

🧠 → Prompt #1
Embed
→ Prompt #2

Embed question and choose prompt based on similarity

Question → 🔴 → ⚪ → Graph DB / Relational DB / Vectorstore → Documents → 🧠 → Answer

## Retrieval

**Ranking**

Question → 🧠 → Relevance

**Re-Rank, RankGPT, RAG-Fusion**

Rank or filter / compress documents based on relevance

**Refinement**

CRAG

**Active Retrieval**

{} → 

CRAG

Re-retrieve and / or retrieve from new data sources (e.g., web) if retrieved documents are not relevant

## Indexing

**Chunk Optimization**

Split → Characters, Sections, Semantic, Delimiters

**Semantic Splitter**
Optimize chunk size used for embedding

**Multi-representation indexing**

→ Summary → {}

**Parent Document, Dense X**
Convert documents into compact retrieval units (e.g., a summary)

**Specialized Embeddings**

🧠 → [0.1, ...]

**Fine-tuning, ColBERT**
Domain-specific and / or advanced embedding models

**Hierarchical Indexing**

Splits → Cluser, Cluser → Summaries → ...

**RAPTOR**
Tree of document summarization at various abstraction levels

## Generation

**Active Retrieval**

{} → → 🧠 → Answer

**Self-RAG, RRR**

Use generation quality to inform question re-writing and / or re-retrieval of documents

🦜🔗 LangChain

# General approaches to transform questions

More abstraction

Step-back question (Step-back prompting)

Question ------> Re-written (RAG-Fusion, Multi-Query)

Less abstraction

Sub-question (Least-to-Most, IR-CoT)

# Least-to-most: Decompose problem into sub-problems, solve sequentially

**(1) Decompose the problem**

Q: "think, machine, learning"
A: "think", "think, machine", "think, machine, learning"

Table 1: Least-to-most prompt context (decomposition) for the last-letter-concatenation task. It can decompose arbitrary long lists into sequential subsists with an accuracy of 100%.

Q: "think, machine"
A: The last letter of "think" is "k". The last letter of "machine" is "e". Concatenating "k", "e" leads to "ke". So, "think, machine" outputs "ke".

Q: "think, machine, learning"
A: "think, machine" outputs "ke". The last letter of "learning" is "g". Concatenating "ke", "g" leads to "keg". So, "think, machine, learning" outputs "keg".

**(2) Solve each step using prior sub-solutions**

Table 2: Least-to-most prompt context (solution) for the last-letter-concatenation task. The two exemplars in this prompt actually demonstrate a base case and a recursive step.

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

# IR-CoT: Interleave retrieval with CoT

In what country was
Lost Gravity manufactured?

cumulate docs

The Lost Gravity was
manufactured by Mack Rides.

cumulate docs

Mack Rides is a company
from Germany.

cumulate docs
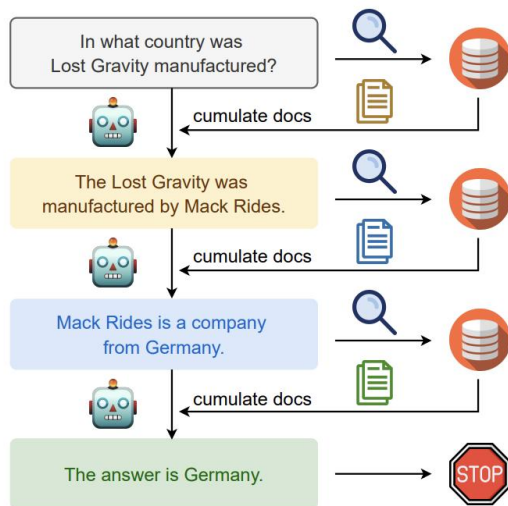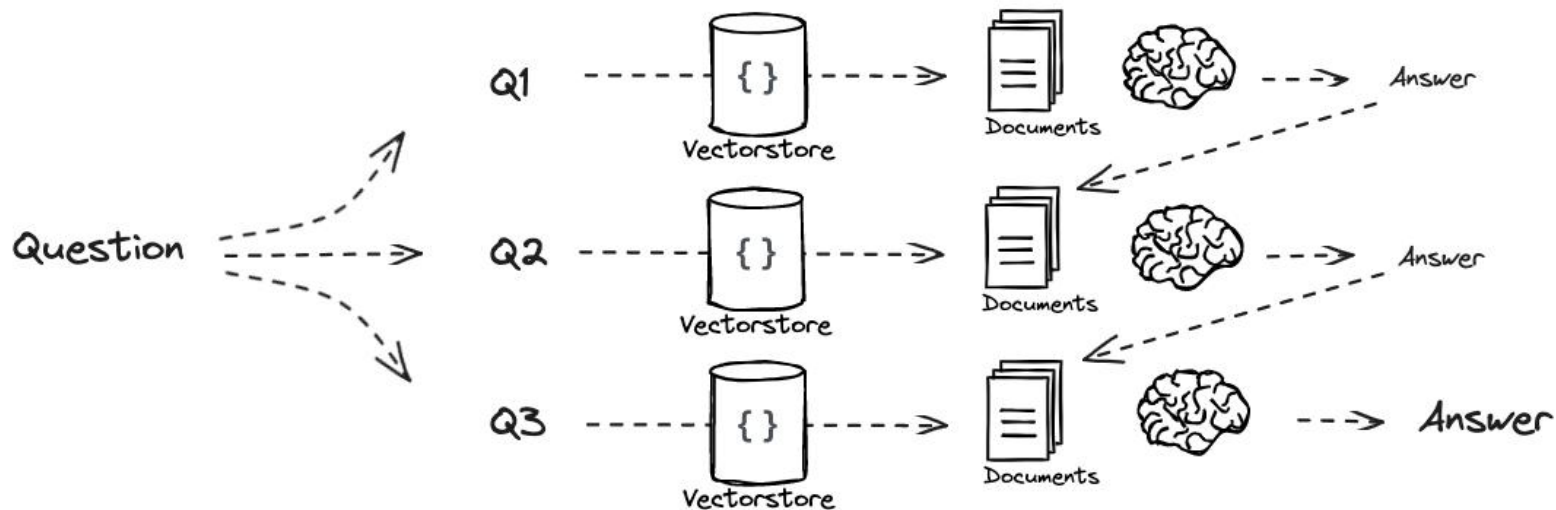
The answer is Germany.

STOP

Figure 1: IRCoT interleaves chain-of-thought (CoT) generation and knowledge retrieval steps in order to guide the retrieval by CoT and vice-versa. This interleaving allows retrieving more relevant information for later reasoning steps, compared to standard retrieval using solely the question as the query.

https://arxiv.org/pdf/2212.10509.pdf

**Combine ideas: Dynamically retrieve to aid in solving the subproblems**

Code walk-through