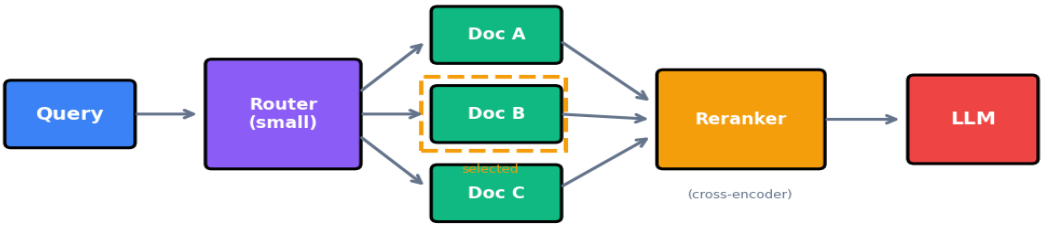
Query/Answer Strategies Comparison

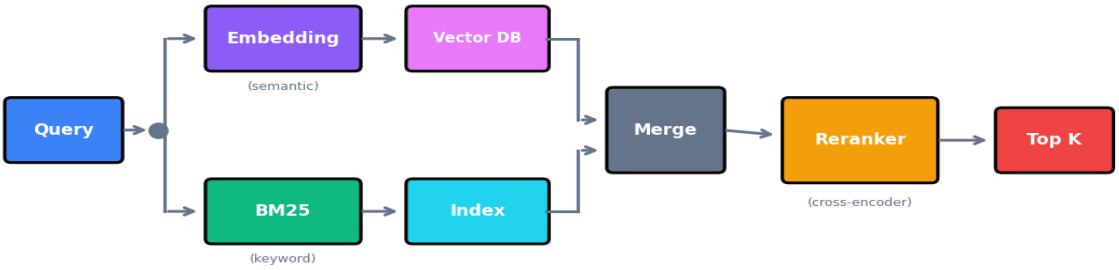
OpenRouter API Token:

```
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
```

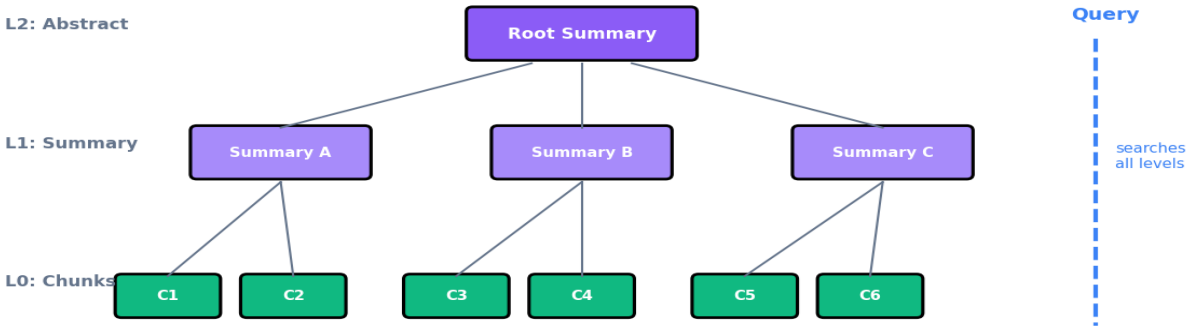
Strategy 1: Small Router Model



Strategy 2: RAG + BM25 Hybrid



Strategy 3: RAPTOR (Hierarchical Summarization)



Quick Comparison

Strategy 1 (Router): Fast, low cost. Best when documents are clearly separable by topic. Weakness: loses granularity within docs.

Strategy 2 (RAG+BM25): Balanced semantic + keyword matching. Good for mixed query types. Weakness: chunking artifacts, flat hierarchy.

Strategy 3 (RAPTOR): Best for synthesis/abstraction queries. Handles multi-hop reasoning. Weakness: high preprocessing cost, summary drift risk.

Code Examples

ChromaDB — Simple Vector Store

```
import chromadb

client = chromadb.Client()  # in-memory

collection = client.create_collection("docs")
collection.add(
    documents=["doc1", "doc2"],
    ids=["1", "2"]
)

results = collection.query(
    query_texts=["search term"],
    n_results=2
)
```

txtai — RAG with OpenRouter

```
import os
from txtai import Embeddings, LLM, RAG

os.environ["OPENROUTER_API_KEY"] = "your-key-here"

# LLM via OpenRouter (use litellm/ prefix)
llm = LLM("litellm/openrouter/meta-llama/llama-3.3-70b-instruct")

# Simple generation
response = llm("What is RAG?")
print(response)
```

RAPTOR — Hierarchical Retrieval with OpenRouter

```
import os
os.environ["OPENROUTER_API_KEY"] = "your-key"

from raptor import (
    BaseSummarizationModel, BaseQAModel, BaseEmbeddingModel,
    RetrievalAugmentationConfig, RetrievalAugmentation
)
from sentence_transformers import SentenceTransformer
from litellm import completion

# Embedding (local, free)
class LocalEmbedding(BaseEmbeddingModel):
    def __init__(self):
        self.model = SentenceTransformer("all-MiniLM-L6-v2")

    def create_embedding(self, text):
        return self.model.encode(text)

# Summarizer via OpenRouter
class OpenRouterSummarizer(BaseSummarizationModel):
    def summarize(self, context, max_tokens=150):
        resp = completion(
            model="openrouter/meta-llama/llama-3.3-70b-instruct",
            messages=[{"role": "user", "content": f"Summarize concisely:\n\n{context}"}],
            max_tokens=max_tokens
        )
        return resp.choices[0].message.content

# QA via OpenRouter
class OpenRouterQA(BaseQAModel):
    def answer_question(self, context, question):
        resp = completion(
            model="openrouter/meta-llama/llama-3.3-70b-instruct",
            messages=[{
                "role": "user",
                "content": f"Answer based on context only.\n\nContext: {context}\n\nQuestion: {question}"
```

```
            }]
        )
        return resp.choices[0].message.content

# Setup
config = RetrievalAugmentationConfig(
    summarization_model=OpenRouterSummarizer(),
    qa_model=OpenRouterQA(),
    embedding_model=LocalEmbedding()
)
RA = RetrievalAugmentation(config=config)

# Index document
text = open("your_doc.txt").read()
RA.add_documents(text)

# Query
answer = RA.answer_question("What are the key points?")
print(answer)
```