



RAG CHEAT SHEET

A comprehensive visual guide to Retrieval-Augmented Generation architectures,
implementation, and best practices

What is RAG?

Retrieval-Augmented Generation (RAG) is an AI architecture that enhances Large Language Models (LLMs) by combining them with external knowledge retrieval systems. Rather than relying solely on the model's internal parameters, RAG allows LLMs to access, retrieve, and use up-to-date information from external databases before generating responses.

Core Components:

Document Processing

Converting documents into embeddings

Vector Database

Storing embedded documents

Retriever

Finding relevant documents

When to Use RAG

- When you need factual accuracy beyond the LLM's training data.
- When working with domain-specific or proprietary information.
- When information needs to be up-to-date and verifiable.
- When transparency and citation of sources matters
- When you need to reduce hallucinations in LLM outputs

Basic RAG Flow:

Document Collection → Document

Chunking Embedding Generation → Vector Database Storage

User Query - Query Embedding →

Similarity Search → Relevant Document Retrieval

Retrieved Documents + Original Query - LLM Generation → Response

10 RAG Architectures Compared

1. Standard RAG

User Query → Query Processing →
Retrieval → Document Selection →
Context Integration → LLM Response

When to Use: For basic question-answering systems needing external knowledge

Real-World Example: Customer support chatbots that access product documentation

Implementation Tip: Start with smaller chunk sizes (512–1024 tokens) and adjust based on performance

2. Corrective RAG

User Query → Initial Response →
Error Detection → Retrieval →
Response Correction → Final Response

When to Use: High-precision use cases where accuracy is critical

Real-World Example: Medical information systems, legal documentation assistance

Implementation Tip: Implement a feedback loop with multiple verification passes

3. Speculative RAG

User Query → Small Model Draft → Parallel Retrievals → Large Model Verification → Response

When to Use: When balancing speed and accuracy is important

Real-World Example: Real-time customer service where response time impacts

Implementation Tip: Use a specialized domain-specific small model for draft generation

4. Fusion RAG

User Query → Multiple Retrieval Methods → Results
Fusion → Aggregated Context → LLM Response

When to Use: When dealing with multiple data sources of varying formats

Real-World Example: Research assistants accessing articles, patents, and databases

Implementation Tip: Weight different sources based on their reliability and relevance

5. Agentic RAG

Advanced → Intent Analysis Agent Selection → Parallel Retrievals → Strategy Coordination → Response

When to Use: Complex queries requiring multiple types of information

Real-World Example: Financial analysis tools accessing market data, company reports, and news

Implementation Tip: Design specialized agents for different query types and data sources

6. Self RAG

User Query → Initial Generation

Self-Critique → Additional

Retrieval → Refined Response

When to Use: For conversational systems requiring consistency

Real-World Example: Educational tutoring systems that build on previous explanations

Implementation Tip: Store conversation history as retrievable context

7. Hierarchical RAG

User Query Top-Level Retrieval
Sub-document Identification →
Focused Retrieval
Response

When to Use: With large, structured documents or knowledge bases

Real-World Example: Enterprise search across documentation hierarchies

Implementation Tip: Create multi-level embedding indexes for efficient navigation.

8. Multi-modal RAG

User Query Cross-modal
Understanding →
Multi-format Retrieval
Format Integration Response

When to Use: When information spans text, images, audio, or video

Real-World Example: E-commerce search using both product descriptions and images

Implementation Tip: Use specialized embeddings for each modality and create bridging mechanisms

9. Adaptive RAG

User Query → Query Analysis →
Retrieval Strategy Selection →
Dynamic Parameter Adjustment → Response

When to Use: For systems facing diverse query types and user needs

Real-World Example: Academic research assistants handling various disciplines

Implementation Tip: Implement real-time feedback to tune retrieval parameters per query

10. Fine-tuned RAG

User Query → Domain-Specific
Processing →
Specialized Retrieval →
Context-Aware LLM → Response

When to Use: For specialized domains requiring expert-level responses

Real-World Example: Technical support systems for complex products

Implementation Tip: Fine-tune both embeddings and LLM on domain-specific data

Best Practices for RAG Implementation

Document Processing

Chunking Strategy

Balance between semantic coherence and retrieval granularity

Embedding Selection

General Purpose

OpenAI ada-002, BERT-based models

Chunk Size

256-1024 tokens depending on content complexity

Specialized Domains

Consider domain-specific embedding models

overlap

10-20% chunk overlap to maintain context across chunks

Dimensions

Higher dimensions(768+) for complex information

Retrieval

Top – k selection

Usually 3–5 chunks for typical queries

Prompt Engineering

Template

“Based on the following information:
{content},please answer:{query}”

Re-ranking

Consider adding a re-ranking step after initial retrieval

Instruction

“Use only the provided information
if you don’t know,say so.”

Hybrid Search

Combine semantic and keyword search for better results

Source Attribution

“For each point in your answer,indicate which source it came from”

Common RAG Challenges & Solutions

challenge	solution
Hallucination	implement fact-checking and source Validation
Retrieval Latency	Use approximate nearest neighbor algorithms
Context Length Limits	Implement recursive Summarization of retrieved chunks
Irrelevant Retrieval	Add filtering and pre-processing of documents
Responce Consistency	implemenet conversation history as part of the context

Evaluating RAG Systems

Metric	Description	Target
Answer Relevance	How well the response answers the query	>85%
Factual Accuracy	correctness of facts in the response	>95%
Retrieval precision	relevance of retrieved documents	>80%
Response Time	Time from query to response	<2s
Source Coverage	Using multiple relevant sources	.>=2 sources

Real-World Use Cases

Enterprise Knowledge Base

Connect employees to internal documentation

Customer Support

Provide accurate product information and troubleshooting

Research Assistant

Aid in literature review and information synthesis

Compliance Monitoring

Ensure responses adhere to regulatory guidelines

Educational Tutoring

Provide accurate explanations with cited sources

Advanced RAG Optimizations

Query Reformulation

Rewrite user queries for better retrieval performance

// Example Query Reformulation
User: "Tell me about rockets"
Reformulated: "What are rockets, their history, types, and applications in space exploration?"

Retrieval Augmentation

Enhance retrieved context with related information

// Retrieved Context Augmentation
1. Original: "SpaceX Falcon 9 specifications"
2. Augmented: "Rocket propulsion systems"
3. Augmented: "Comparison with other launch vehicles"

Contextual Compression

Condense retrieved information to focus on relevance

// Context Compression Pipeline

1. Retrieve k-10 documents
2. Generate summary for each document
3. Rank summaries by relevance
4. Select top n-3 summaries

Adaptive RAG

Dynamically adjust retrieval parameters based on query type

// Adaptive Parameter Selection if query_is_factual():

k = 3 + fewer, precise documents

similarity_threshold 0.8

elif query_is_exploratory(): k = 7
more, diverse

documents

similarity_threshold - 0.6

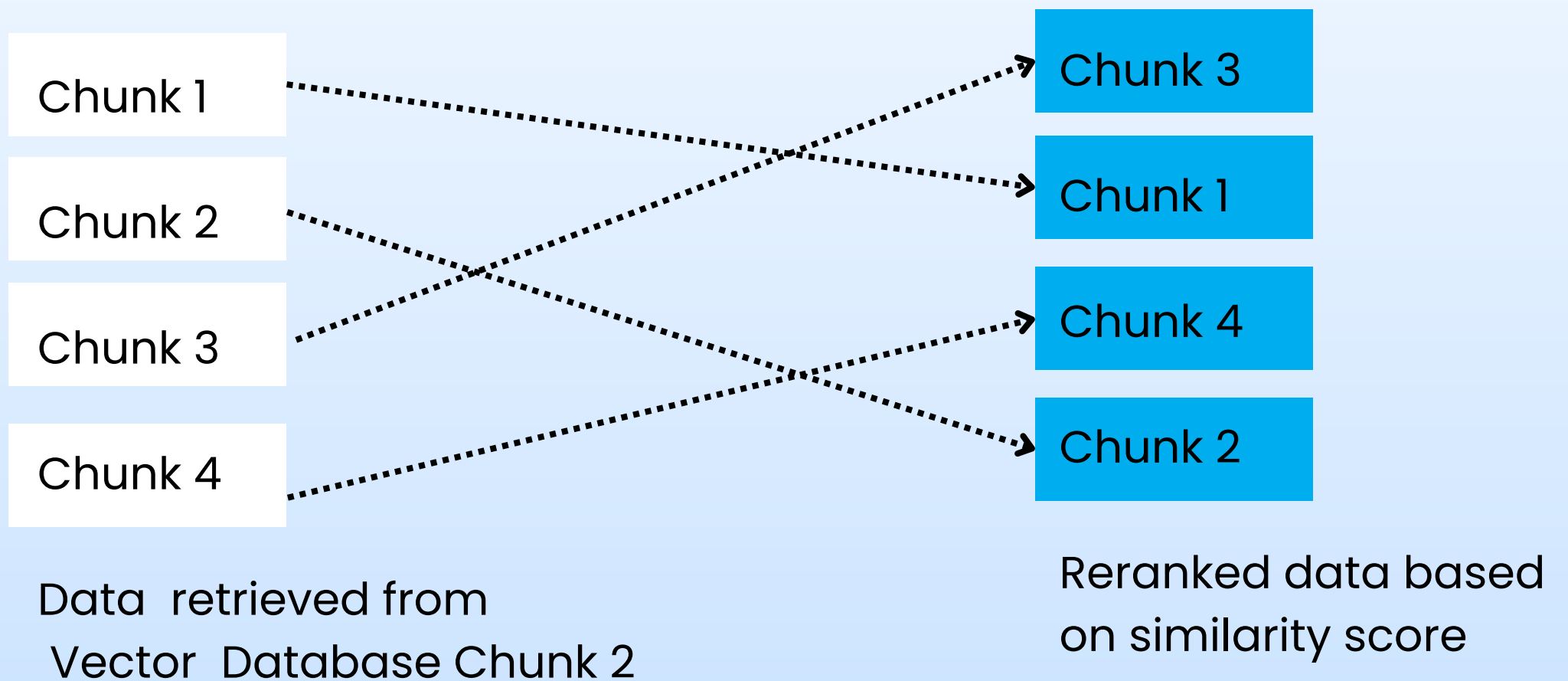
Continuous Learning

Update vector stores and embeddings as new information arrives

// Incremental Updating

1. Monitor for new documents
2. Process and embed new content
3. Merge into existing vector store
4. Periodically re-index for optimization

Reranking



Practical RAG

Implimentation Guide

Setting Up Your Document Processing Pipeline

```
import os
from langchain.document_loaders import DirectoryLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter

# Load documents
loader = DirectoryLoader('./documents/', glob="**/*.pdf")
documents = loader.load()

# Split into chunks
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,
    chunk_overlap=200,
    length_function=len,
)
chunks = text_splitter.split_documents(documents)
```

Creating and Storing Embeddings

```
from langchain.embeddings import OpenAIEmbeddings
from langchain.vectorstores import Chroma

# Initialize embeddings
embeddings = OpenAIEmbeddings()

# Create vector store
vectorstore = Chroma.from_documents(
    documents=chunks,
    embedding=embeddings,
    persist_directory="./chroma_db"
)

# Persist to disk
vectorstore.persist()
```

3. Building the Retrieval System

```
from langchain.retrievers import
ContextualCompressionRetriever
from langchain.retrievers.document_compressors import
LLMChainExtractor

# Create base retriever
retriever = vectorstore.as_retriever(search_type="similarity",
    search_kwargs={"k": 4})

# Add compression for better context
llm = ChatOpenAI(temperature=0)
compressor = LLMChainExtractor.from_llm(llm)
compression_retriever
=ContextualCompressionRetriever(base_compressor=compressor,base_retriever=retriever)
```

Implementing the RAG Chain

```
from langchain.chat_models import ChatOpenAI
from langchain.chains import RetrievalQA

# Initialize LLM
llm = ChatOpenAI(temperature=0, model="gpt-4")

# Create RAG chain
qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=compression_retriever,
    return_source_documents=True,
    chain_type_kwargs={
        "prompt": CUSTOM_PROMPT_TEMPLATE
    }
)

# Query the system
result = qa_chain({"query": "How do rockets work?"})
print(result["result"])
```


Popular Vector Database Comparison

Pinecone



- Fully managed service
- Low latency queries
- Scales to billions of vectors
- High availability

Weaviate



- Open-source
- GraphQL API
- Classification support
- Multi-tenancy support

Chroma



- Open-source
- Easy Python integration
- Simple deployment
- Good for prototyping

Milvus



- Open-source
- Distributed architecture
- Hybrid search
- High scalability

Qdrant



- Open-source
- Filtering capabilities
- On-prem deployment
- REST and gRPC APIs

"Vector Databases"

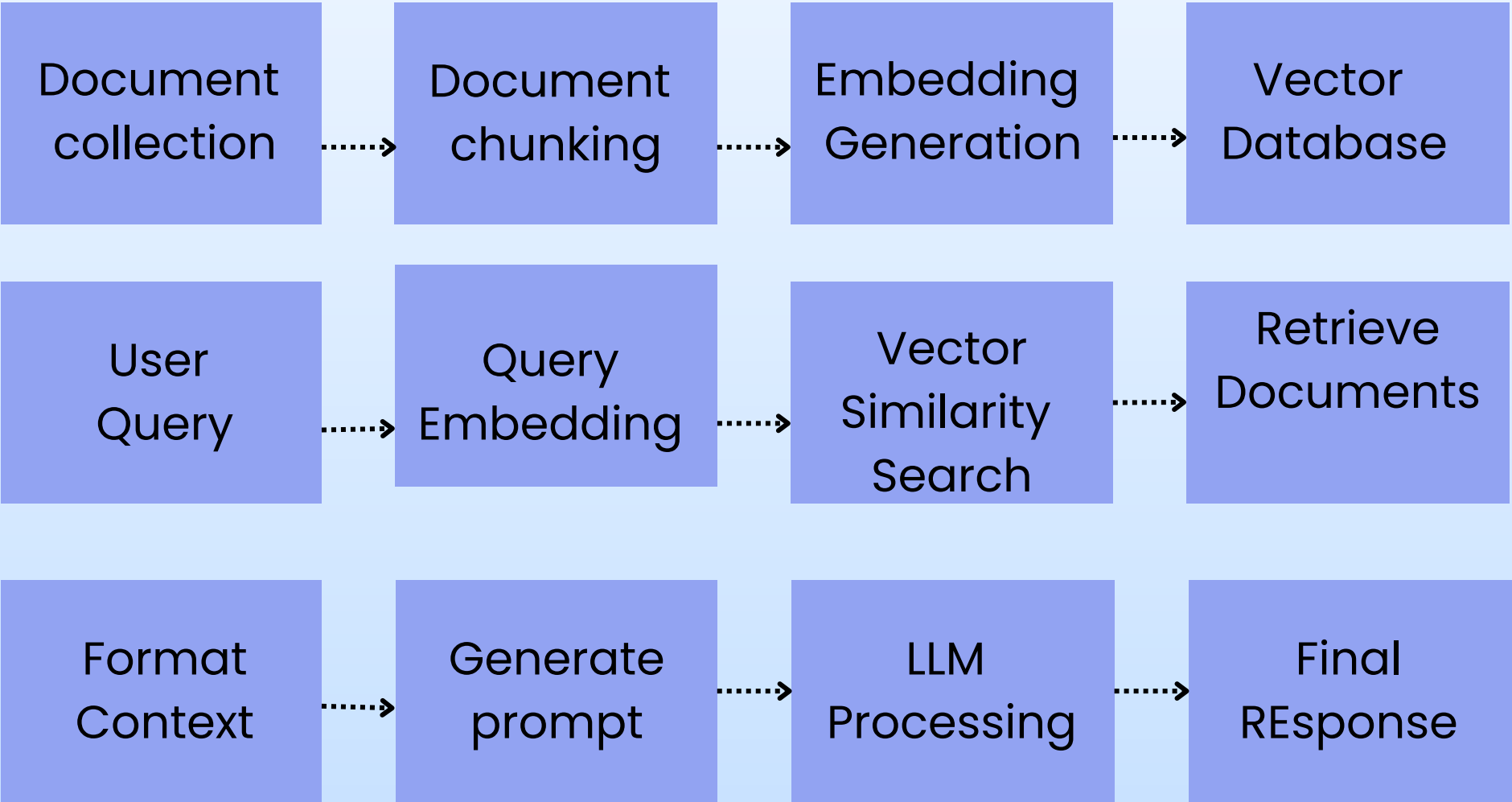


Pinecone
Weaviate
Chroma
Milvus
Qdrant
Redis
Vespa

Comparing RAG performance Metrics

Architecture	Response Time	Accuracy	Memory Usage	Implementation Complexity
Corrective RAG	★★	★★★★★	★★★★	★★★★
Speculative RAG	★★★ ★★	★★★★★	★★	★★★★
Fusion RAG	★★	★★★★	★★★	★★
Agentic RAG	★★	★★★★★	★	★★★★ ★
Standard RAG	★★★ ★★	★★★★	★★★★	★

Standard RAG Implementation Flowchart



popular RAG Tools and Frameworks

LangChain

Framework for connecting LLMs with external data sources



LlamaIndex

Data framework for augmenting LLMs with private data



Haystack

End-to-end framework for building NLP pipelines



txtai

All-in-one embeddings database with search capabilities



Semantic Kernel

Microsoft's SDK for integrating LLMs with code





WAS THIS POST USEFUL?

**FOLLOW
FOR MORE**

 **Repost**