Analytics Vidhya

Mastering RAG

RAG Developer Stack

OpenAI

MISTRAL AI_

Claude

Gemini

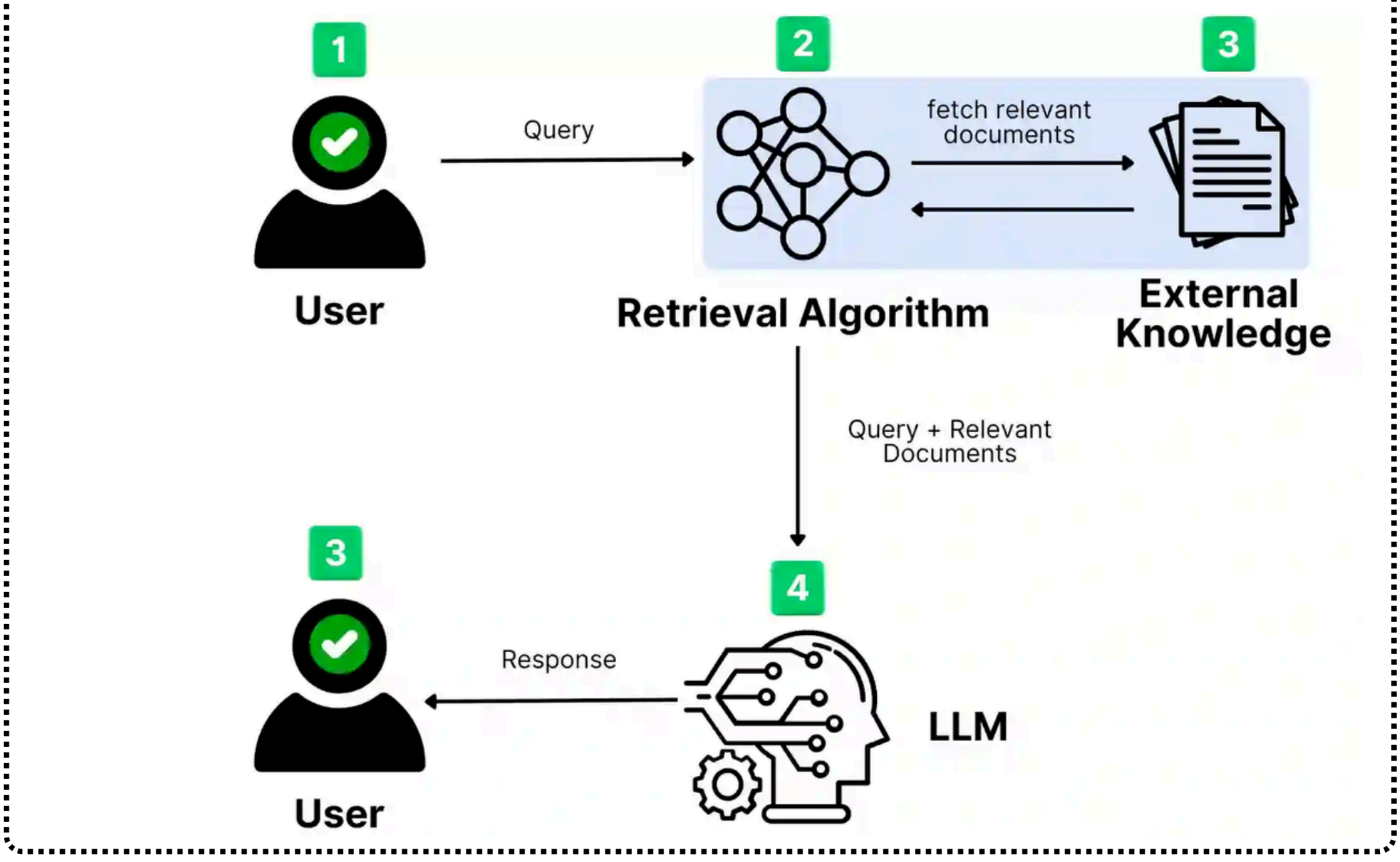
LLaMA by Meta

Chroma

Cohere

Weaviate

Pinecone



1. Large Language Models (LLMs)

RAG uses pre-trained LLMs for text generation. Selecting the right model depends on latency, cost, and accuracy requirements.

Popular LLMs for RAG:

- OpenAI GPT-4.5 / GPT-4o (via API)
- Mistral / Mixtral
- Meta LLaMA 3.3 / 3.2
- Anthropic Claude 3.7
- Google Gemini 2.0
- Falcon / Bloom / Pythia
- Command R+ (Cohere)

💡 Tip: Choose open-source LLMs for privacy & on-premise deployment.

2. Retrieval Mechanisms

Retrieval is a crucial step in RAG, responsible for fetching relevant information before passing it to the LLM.

Types of Retrieval:

Dense Retrieval

- Uses neural embeddings to find semantically relevant documents.
- Example: Dense Passage Retrieval (DPR), ColBERT, Contriever

Sparse Retrieval (BM25 / TF-IDF)

- Traditional search method based on term frequency & relevance scoring.

Hybrid Retrieval (Dense + Sparse)

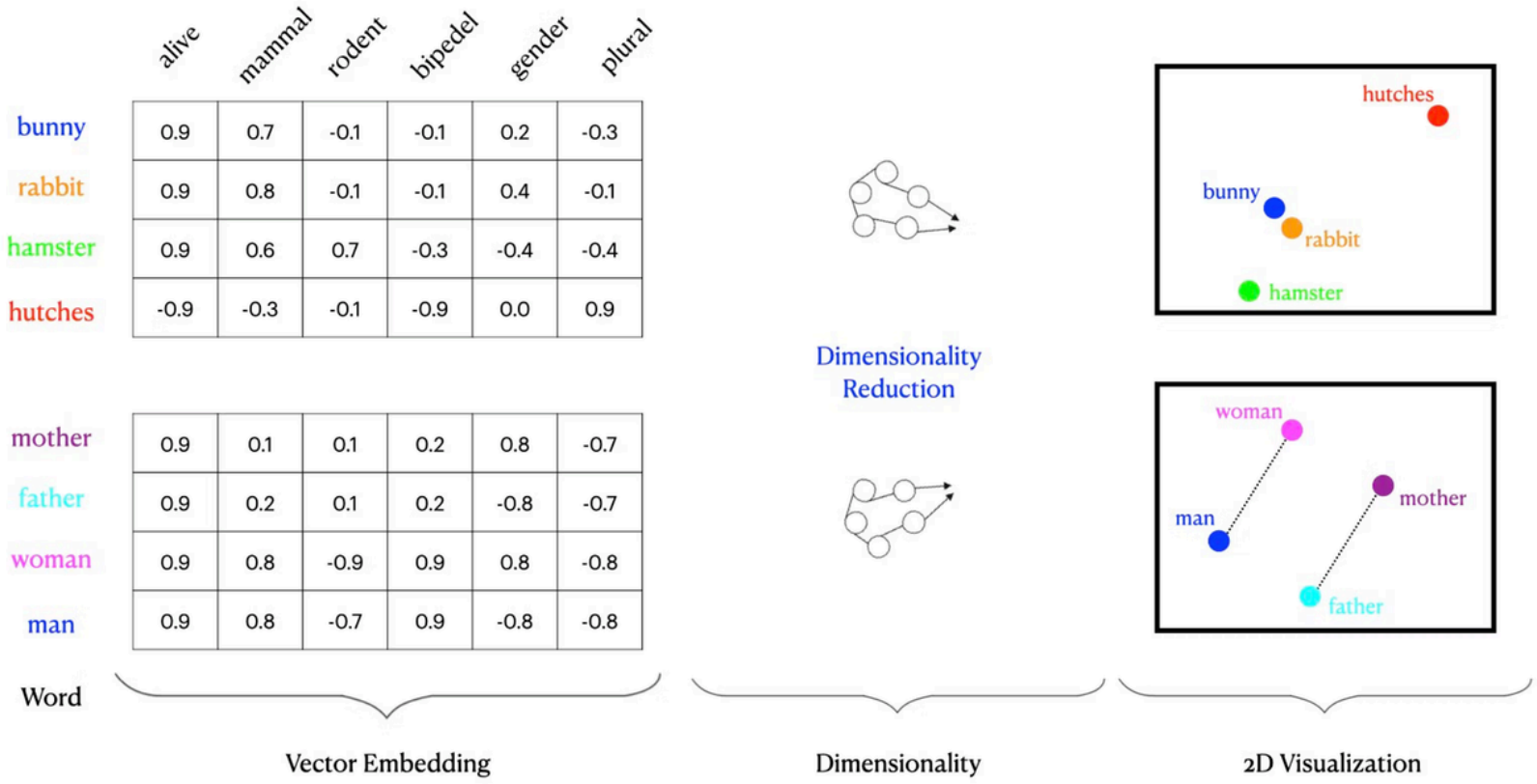
- Combines BM25 & Vector Search for better recall & precision.

Retrieval Frameworks:

- **FAISS (Facebook AI Similarity Search)**
- **ChromaDB** (lightweight & fast)
- **Weaviate** (open-source & scalable)
- **Pinecone** (fully managed vector DB)
- Qdrant (AI-native vector database and a semantic search engine)
- **Milvus** (high-speed retrieval)

3.Vector Embeddings

Documents & queries are converted into high-dimensional vectors before retrieval.



Popular Embedding Models:

- OpenAI’s text-embedding-3-large
- Hugging Face Sentence Transformers (e.g., BERT, MiniLM) Cohere Embed Models
- BAAI’s BGE Embeddings

💡 Tip: Choose open-source LLMs for privacy & on-premise deployment.

4.Chunking & Indexing

To improve retrieval efficiency, documents must be chunked & indexed effectively.

Chunking Strategies:

- **Fixed-Length Chunks** (e.g., 512 or 1024 tokens)
- **Recursive Character Splitting** (based on paragraph boundaries)
- **Sliding Window** (overlapping chunks for better context)

Indexing Frameworks:

- **LlamaIndex (Formerly GPT Index)**
- **Haystack (deepset AI)**
- **LangChain Document Loaders & Splitters**

5. Re-Ranking

Re-ranking **improves retrieval results** by scoring and ordering retrieved documents before feeding them to the LLM.

Re-Ranking Models:

- **ncoders (e.g., MS-MARCO, Cohere Reranker)**
- **ColBERT (Late Interaction Ranking)**
- **bge-m3**
- **mxbai-embed-large-v1**
- **Hybrid Rankers (BM25 + Neural Re-rankers)**

6. Orchestration & Frameworks

To simplify RAG workflows, **frameworks** help in **retrieval, embedding, and response generation.**

Best RAG Frameworks:

- **LangChain** (Modular, widely used)
- **LlamaIndex** (Efficient document indexing & retrieval)
- **Haystack** (Scalable, for production RAG apps)
- **FastRAG** (Lightweight & optimized)

7. Query Processing & Prompt Engineering

The quality of the retrieval query directly affects RAG output.

Techniques for Query Optimization:

- **Query Expansion** (Add synonyms & related terms)
- **Rewriting Queries** (Using LLMs to generate better search queries)
- **Contextualization** (Retain user history for relevance)

Prompt Engineering Methods:

- **Chain-of-Thought (CoT)** (For reasoning-heavy tasks)
- **Retrieval-Augmented Prompts** (Dynamically inserting context)
- **Few-Shot Learning** (Providing examples for better outputs)

8. Caching for Speed Optimization

Since retrieval & generation can be computationally expensive, caching is used to speed up responses.

Caching Strategies:

- **Semantic Caching** (Store past queries & responses)
- **Vector Index Caching** (Avoid redundant retrieval)
- **LLM API Response Caching** (Reduce token cost)

Tools for Caching:

- **Redis** (for fast in-memory caching)
- **LlamaIndex Hybrid Cache**
- **Local Disk-Based Caching** (via SQLite, Pickle)

9. Evaluation & Metrics

Measuring RAG system performance ensures **accuracy & efficiency.**

Key Evaluation Metrics:

- **Retrieval Precision & Recall** (Relevance of retrieved documents)
- **Hallucination Rate** (False information in generated responses)
- **Latency** (Time taken for retrieval + generation)
- **Token Efficiency** (Cost-effective context usage)

Evaluation Frameworks:

- **EVALRAG** (by Hugging Face)
- **DeepEval**
- **Arize AI Phoenix**
- **LlamaIndex Evaluator**
- **OpenAI's EvalGPT**
- **Retrieval-Augmented Benchmarking Tools (RAGAS)**

10. Deployment & Scalability

RAG applications need to be **scalable & optimized** for production use.

Deployment Options:

- **Cloud-Based** (AWS, GCP, Azure)
- **On-Premises** (Using Hugging Face Models + FAISS)
- **Hybrid** (Edge + Cloud for latency optimization)

Scaling Strategies:

- **Batch Processing** (Pre-compute embeddings)
- **Asynchronous Retrieval** (Parallel requests for speed-up)
- **Model Distillation** (Use smaller LLMs for cost-efficiency)