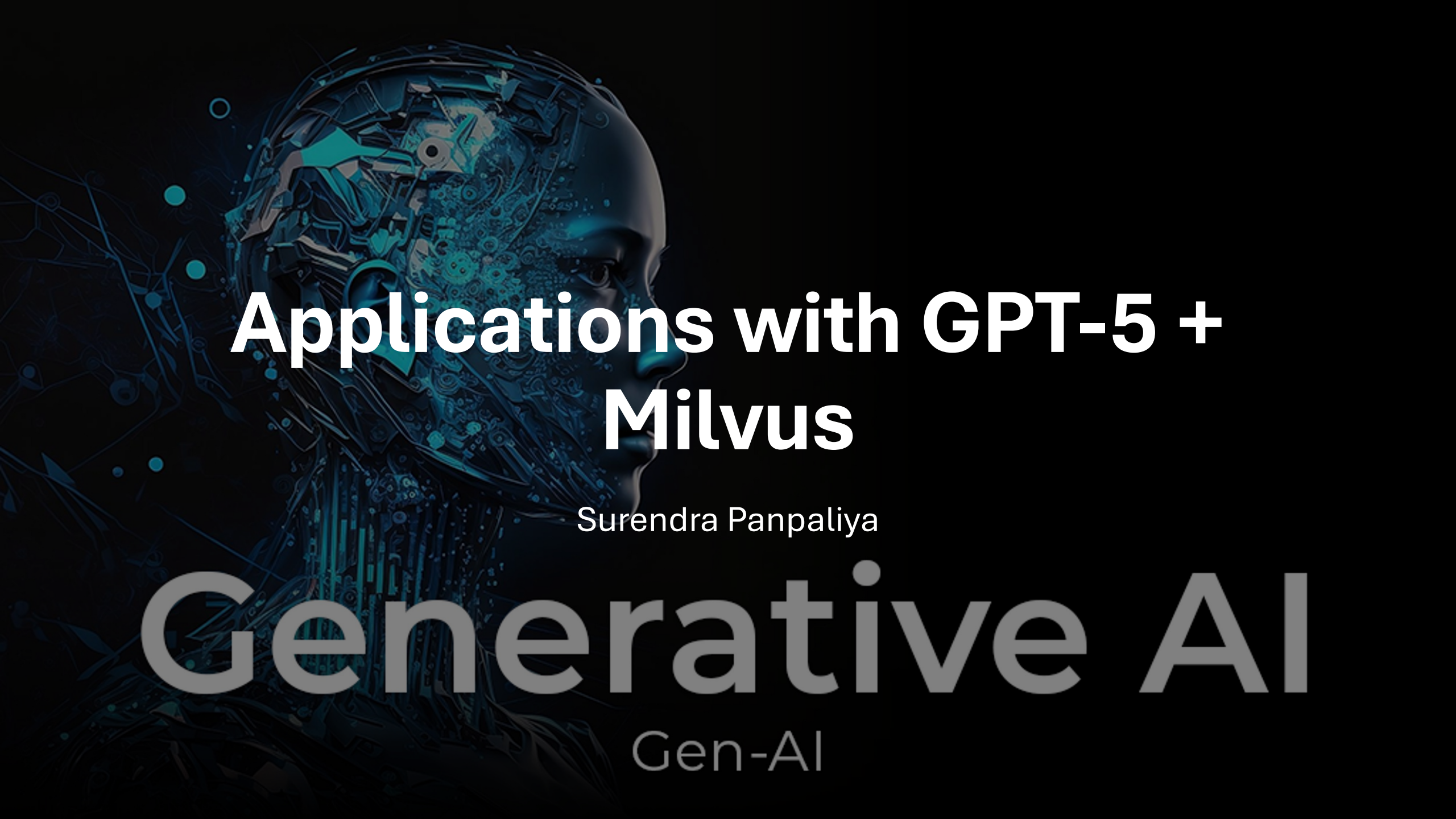


Applications with GPT-5 + Milvus

Surendra Panpaliya

Agenda

Vector databases in enterprise AI

Milvus introduction & **PyMilvus integration**

RAG pipelines for enterprise knowledge bases

LangChain & LangGraph integration with GPT-5

Hands-On

Store & retrieve documents in Milvus

Build semantic Q&A chatbot for project docs

Resume Analyzer with GPT-5 + Milvus

Assignment

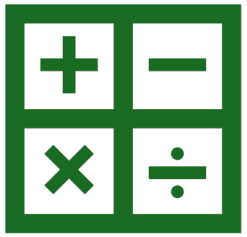
Deploy a mini semantic search app with Streamlit

Vector databases in enterprise AI

- What is a Vector Database?
- Types of Vector Databases in Enterprise AI
- Focus on Milvus & PyMilvus
- Comparison of Popular Open-Source Vector Databases
- Enterprise Considerations

What is a Vector Database?

High-dimensional embeddings



Numerical representations of

text, images, audio, code, etc.

What is a Vector Database?

An **embedding** is just a vector

"ChatGPT" → [0.12, -0.87, 0.44, ...].

What is a Vector Database?

A **vector database** stores

Millions/billions of these vectors and

Supports **Approximate Nearest Neighbor (ANN) search**,

so we can find “the most similar items” fast.

Example

Query: *“Show me documents about cloud cost optimization”*

Convert query → embedding vector

Search in vector DB → return top-k most similar embeddings (docs, FAQs, images).

Types of Vector Databases in Enterprise AI

1. Dedicated Vector Databases

2. Vector Extensions in General Databases

1. Dedicated Vector Databases

Purpose-built for Storing

Searching embeddings.

Examples:

Milvus, Qdrant, Weaviate,

Pinecone, Vespa.

1. Dedicated Vector Databases

Provide APIs, clustering,

Indexing (IVF, HNSW, PQ),

Distributed scaling

1. Dedicated Vector Databases

Designed for **RAG pipelines**

Semantic search

Real-time recommendations

2. Vector Extensions in General Databases

Traditional DBs adding vector support.

Examples:

Postgres + pgvector,

Elasticsearch k-NN,

MongoDB Atlas Vector Search.

2. Vector Extensions in General Databases

Good when you want

SQL + vector search in one place.

Typically **less optimized** than

Dedicated vector DBs for

very large-scale ANN.

Milvus

An **open-source,**

cloud-native vector database.

Designed for **scalability**

(billions of vectors).

Milvus

Supports

Multiple indexing methods

IVF, HNSW, DiskANN.

Milvus

- Integrates with
- enterprise AI pipelines
- RAG, multimodal search
- Backed by **Zilliz** (enterprise offering).
- Strong community adoption

PyMilvus

The **Python SDK for Milvus**.

Lets you connect to a Milvus cluster and

do operations

(insert, query, delete, manage collections).

Open-Source Vector Databases

Feature	Milvus	Qdrant	Weaviate	FAISS	Vespa
Type	Dedicated Vector DB	Dedicated Vector DB	Dedicated Vector DB	Library (not full DB)	Search + Serving Engine
License	Open Source (Apache 2.0)	Open Source (Apache 2.0)	Open Source (BSD)	Open Source (MIT)	Open Source (Apache 2.0)
Language	C++/Go backend, Python SDK (PyMilvus)	Rust backend, Python/JS client	Go backend, GraphQL/REST API	C++/Python lib	Java backend
Best Use Case	Enterprise RAG, Large-scale embeddings	Real-time search, Recommendation engines	Semantic search + Knowledge Graphs	Fast local similarity search (small-medium scale)	Large-scale text+vector search, Ads, Personalization
Scalability	Billions of vectors (cloud-native)	Millions to billions (disk-based + HNSW)	Scales horizontally (with Kubernetes)	Not distributed (single node)	Internet-scale search (used by Yahoo, Spotify)

Open-Source Vector Databases

Feature	Milvus	Qdrant	Weaviate	FAISS	Vespa
Index Types	IVF, HNSW, DiskANN, PQ	HNSW	HNSW, Flat, PQ	IVF, HNSW, PQ	Custom ANN indexes
APIs	gRPC + PyMilvus + REST	REST, gRPC, Python SDK	GraphQL/REST	Python/C++ only	REST/Java APIs
Enterprise Support	Zilliz Cloud	Qdrant Cloud	Semi Technologies (Weaviate Cloud)	Community only	Verizon Media / Vespa.ai
Extra Features	Hybrid search (vector + scalar filters), time travel	Payload filtering, multi-tenant	Built-in ML modules, classification	Just library (no persistence)	Strong ranking, hybrid search

Enterprise Considerations

SCALE:

MILVUS & VESPA

PROVEN AT BILLION+ SCALE.

Enterprise Considerations

Integration:

Weaviate has GraphQL &

built-in ML modules

(easy for knowledge graph + AI search).

Enterprise Considerations

Flexibility:

Qdrant is lightweight,

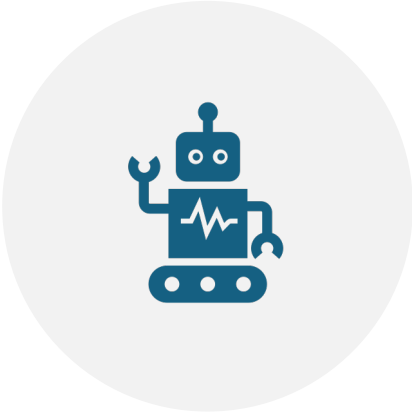
Rust-based,

efficient for real-time use cases.

Enterprise Considerations

DEVELOPER TOOLING:

FAISS IS GREAT FOR RESEARCH,



PROTOTYPING, NOT FOR PRODUCTION.

Enterprise Considerations

Ops/Deployment:

Milvus:

CNCF(Cloud Native Computing Foundation) style,

works with Kubernetes.

Enterprise Considerations

Ops/Deployment:

Qdrant: Cloud service + on-prem.

Weaviate: Cloud-native,

can extend with Python modules.

Summary

A **vector database** is

the **retrieval backbone** of

modern **enterprise AI**

(GenAI + RAG).

Summary

Dedicated DBs

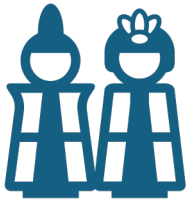
Milvus, Qdrant, Weaviate, Vespa

scale, performance,

specialized indexing.

Summary

Traditional DBs with vector plugins



(pgvector, Mongo, Elastic)

integration convenience,

but less optimized at extreme scale.

Summary

Milvus + PyMilvus is

a strong enterprise choice:

open-source, scalable,

Python SDK,

production-ready.

Summary

Qdrant and **Weaviate** are excellent alternatives

depending on whether you want lightweight

Rust performance (Qdrant) or GraphQL +

knowledge graph integration (Weaviate).

Milvus introduction & PyMilvus integration

What is Milvus?

Key ideas

Ways to run locally

PyMilvus integration

What is Milvus?

An open-source,

cloud-native **vector database**

for storing embeddings

(high-dimensional vectors) and doing

Approximate Nearest-Neighbor (ANN) search

at scale (millions → billions).

What is Milvus?

Powers semantic search,

RAG, Recommendations

Deduplication

Anomaly detection.

Key ideas

Collection: like a table.

Fields: scalar fields

(id, tags, timestamps) +

vector field (the embedding).

Key ideas

Index: HNSW, IVF_FLAT,

IVF_PQ, DiskANN

Metric:

similarity (IP/cosine, L2).

Key ideas

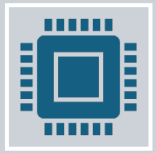
Partitioning: segment data (e.g., by tenant/date).

Consistency: eventual/strong (tune per workload).

Hybrid filters: combine vector search

with scalar predicates.

Ways to run locally



Milvus Standalone (Docker) – realistic dev/prod parity.

Milvus Lite (embedded) –

pip install milvus-lite pymilvus

PyMilvus integration (end-to-end, minimal)

Connects to Milvus

Creates a collection + index

Inserts vectors + metadata

Searches with top-k and an optional filter

PyMilvus integration (end-to-end, minimal)

pip install "pymilvus>=2.4"

pip install milvus-lite to avoid Docker.

If using Docker Milvus,

default gRPC is localhost:19530.

RAG Architecture Overview

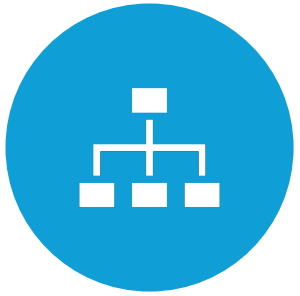
Introduction to Retrieval-Augmented Generation

Why RAG is critical for enterprise GenAI adoption

What is RAG ?

Problem:

LLMs (like GPT-5) are trained on general knowledge



but lack your **enterprise-specific docs**

(policies, Jira tickets, SOPs, PDFs, Confluence pages).

What is RAG?

Solution: RAG combines:

Retriever → fetch relevant knowledge from

enterprise KB (vector DB, search engine).

Generator (LLM) → use the retrieved passages

as additional context to answer user queries.

What is RAG?

Outcome:

LLM answers are **grounded in**

Enterprise content

(accurate, secure, explainable).

What is RAG?

RAG combines traditional **retrieval systems**

(like document or vector search) with **LLMs**

to produce answers grounded in real,

up-to-date data

What is RAG?

Retrieves relevant content

from enterprise data

policies, product manuals,

inventory logs

What is RAG?

Augments

the user query

by combining it

with this retrieved context.

What is RAG?

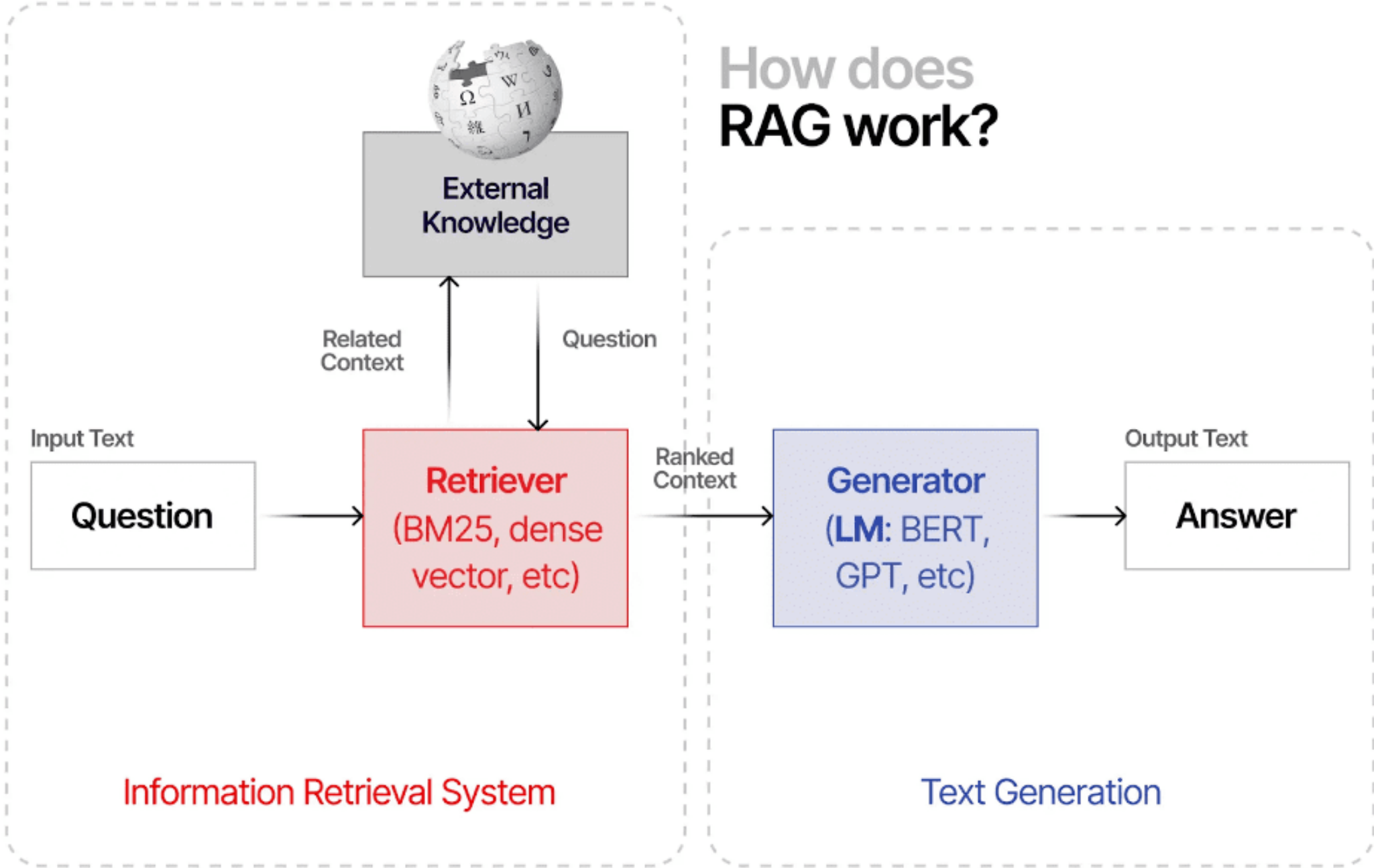
Generates

a precise,

accurate response

using the LLM.

RAG Pipeline Architecture (Enterprise Knowledge Base)


How does
RAG work?
External Knowledge
Related Context
Question
Input Text
Question
Retriever (BM25, dense vector, etc)
Ranked Context
Generator (LM: BERT, GPT, etc)
Output Text
Answer
Information Retrieval System
Text Generation

1. Data Sources

- Docs: PDFs, Word, PPT, CSVs
- Systems: Confluence, Jira, SharePoint, GitHub, ServiceNow
- Databases: SQL, NoSQL, Data Warehouses

2. Ingestion & Preprocessing

- Chunking (split docs into smaller passages, e.g., 500–1000 tokens)
- Cleaning (remove boilerplate, headers, formatting noise)
- Metadata tagging (source, author, department, timestamp)

3. Embedding & Indexing

- Convert chunks → embeddings via LLM models (e.g., OpenAI text-embedding-3-large, HuggingFace models).
- Store embeddings in a **vector database** (Milvus, Qdrant, Weaviate, Pinecone, or pgvector).
- Index with HNSW / IVF for fast similarity search.

4. Retriever

- Given a query → embed it → vector search in the DB → fetch top-k relevant passages.
- Optional: Hybrid retrieval (vector + keyword filters, e.g., “department=HR”).

5. Generator (LLM)

- Construct prompt with:
  - User query
  - Retrieved passages (with metadata)
- LLM (GPT-5, Llama, etc.) generates grounded response.

6. Post-processing

- Add citations to source docs.
- Apply redaction (mask PII, confidential terms).
- Enforce guardrails (no hallucination beyond retrieved context).

Open Source Vector DBs for Enterprise RAG

Feature	Milvus	Qdrant	Weaviate	pgvector
Scale	Billion+ vectors	Million–Billion	Billion+	Smaller scale
Query	ANN + scalar filters	ANN + payload filters	ANN + hybrid KG search	SQL + ANN
Strength	Cloud-native, CNCF, PyMilvus SDK	Rust speed, lightweight	Built-in ML modules	Easy SQL integration
Best Fit	Large RAG KBs (cross-department)	Recs, fast metadata filters	Knowledge Graph + AI search	Adding vectors into existing Postgres infra

Enterprise Considerations for RAG

- **Security**: Control access (row/field-level filters by department/tenant).
- **Compliance**: PII redaction, GDPR “right to forget” = delete vectors by doc_id.
- **Freshness**: Continuous ingestion pipelines (watch Confluence/Jira/Git).
- **Governance**: Track embeddings version & reindex if model updates.
- **Observability**: Monitor recall, latency, coverage (did retriever find useful docs?).
- **Guardrails**: Never let LLM hallucinate outside retrieved context → enforce “answer only from context.”

Summary

- RAG = **Retriever (vector DB) + Generator (LLM)**.
- Use it to ground GenAI in **enterprise knowledge bases**.
- **Milvus + PyMilvus** is enterprise-grade (scale, hybrid search, filters).
- SQLite/Postgres hold metadata; Milvus holds embeddings.
- Add **guardrails + governance** for compliance and trust.

Step-by-Step RAG Workflow

Why RAG is Critical for Enterprise GenAI ?

Reduces hallucinations,

ensuring accuracy and trust

Why RAG is Critical for Enterprise GenAI ?

Keeps responses current

by accessing live

enterprise data

no retraining needed

Why RAG is Critical for Enterprise GenAI ?

ENABLES DOMAIN-SPECIFIC ANSWERS,

IMPORTANT FOR POLICY,

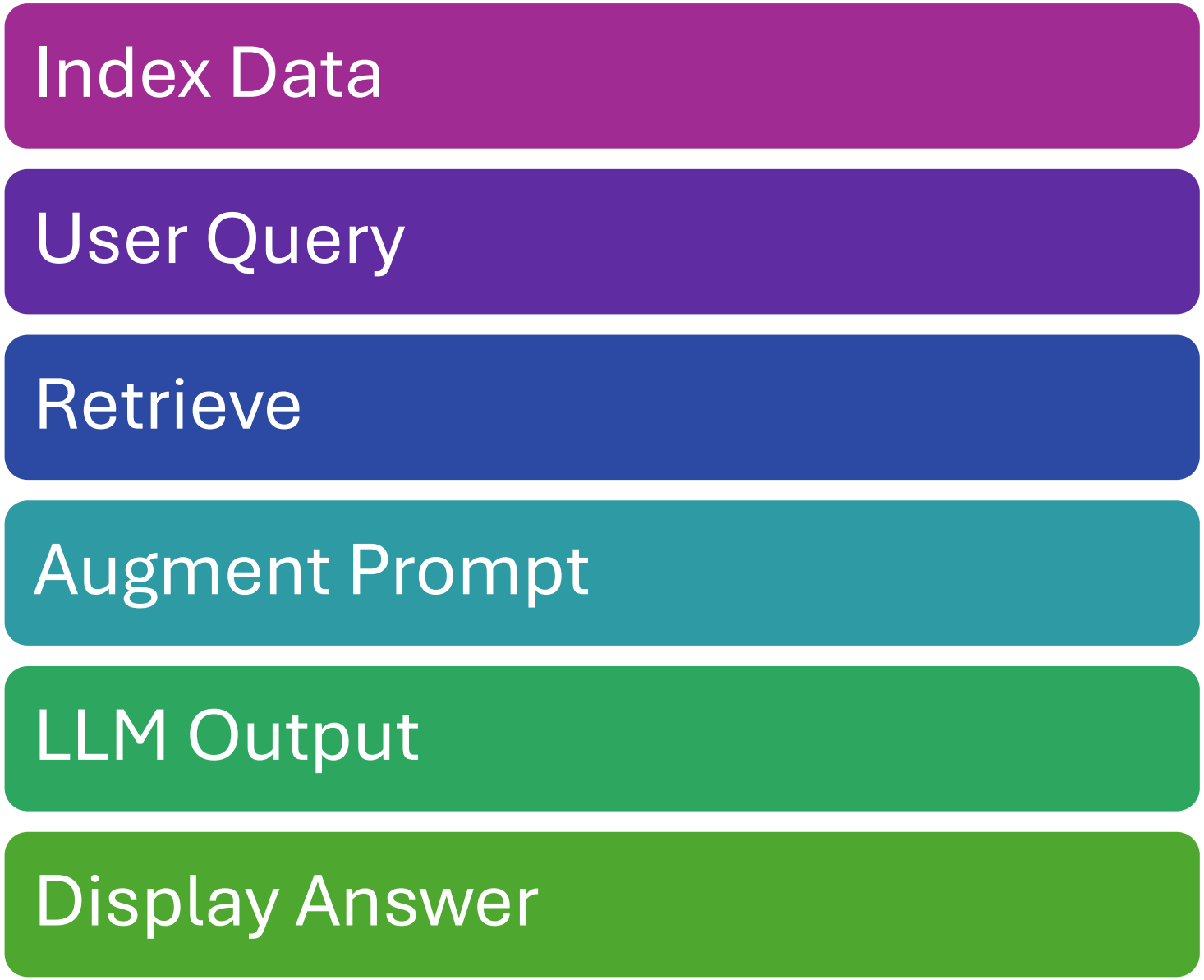
INVENTORY, LEGAL, OR

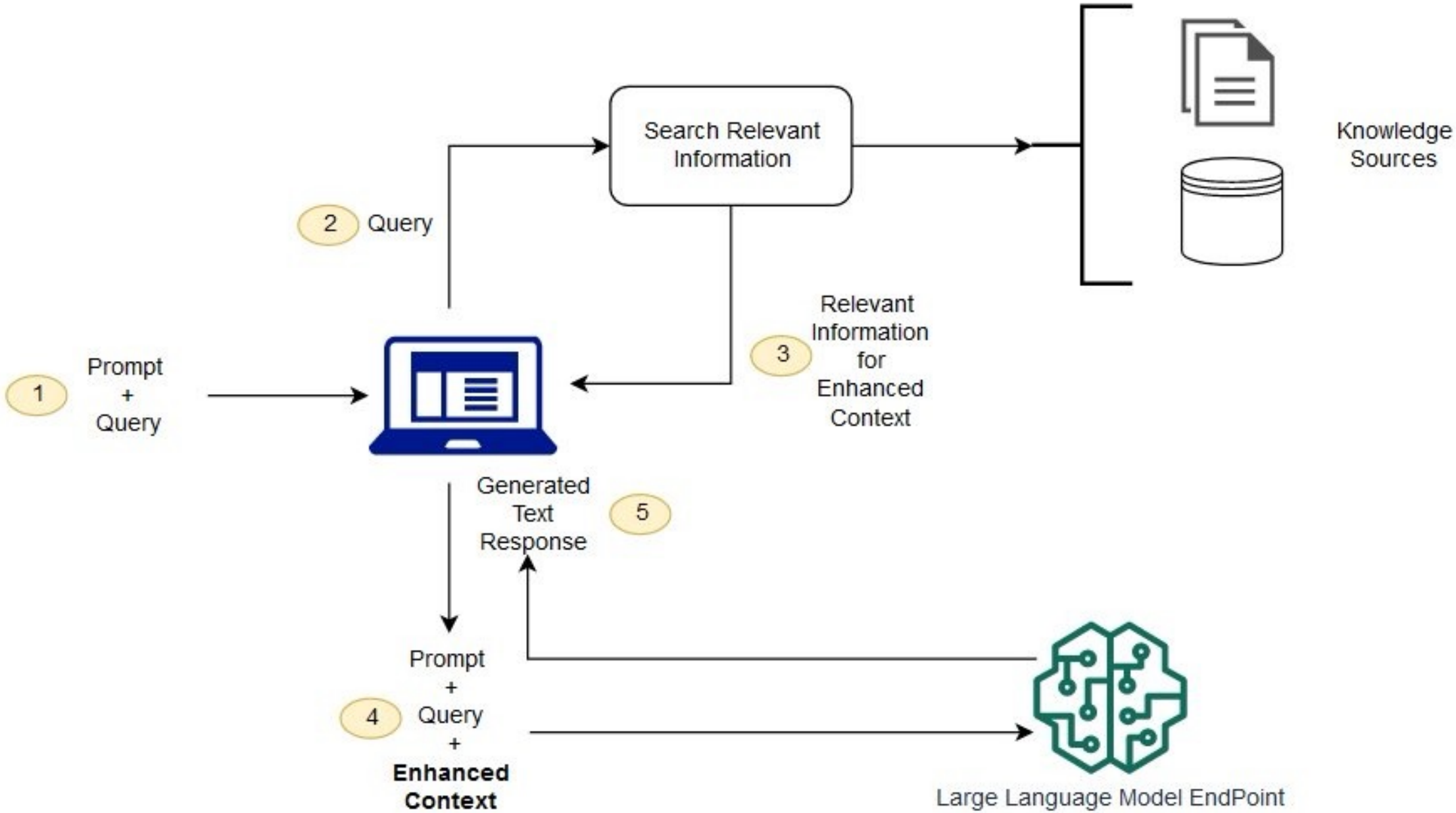
OPERATIONAL QUERIES

Why RAG is Critical for Enterprise GenAI ?

Faster and cheaper to implement than full model retraining

RAG Workflow




Search Relevant Information
Knowledge Sources
2 Query
1 Prompt + Query
3 Relevant Information for Enhanced Context
Generated Text Response
5
4 Prompt + Query + Enhanced Context
Large Language Model EndPoint

Summary of Each Step

Step	Action
1	User inputs a prompt + query
2	System sends the query to the retriever
3	Retriever fetches relevant knowledge
4	System builds enhanced context (query + prompt + facts)
5	LLM generates a grounded and accurate text response

Why Walmart Needs RAG?

Without RAG:

LLMs might **hallucinate** or provide outdated info.

Sensitive enterprise data stays siloed.

LangChain Integration

Surendra Panpaliya

GKTCS Innovations

https://www.gktcs.com

Agenda

LangChain with OpenAI / Gemini

Document loading,

Retriever setup,

Chain creation

What is LangChain?

A powerful **framework**

for building applications

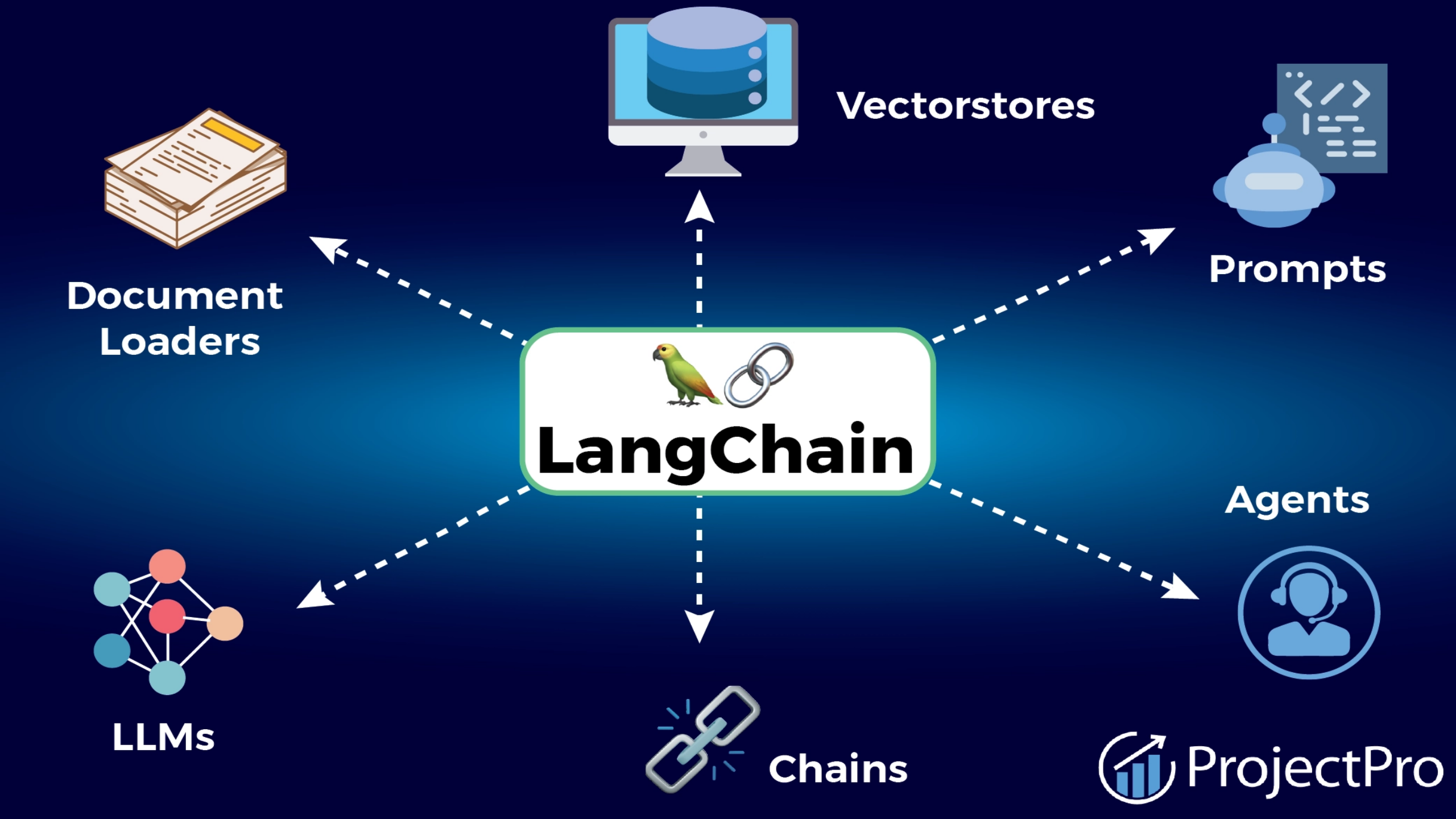
using Large Language Models (LLMs).

What is LangChain?

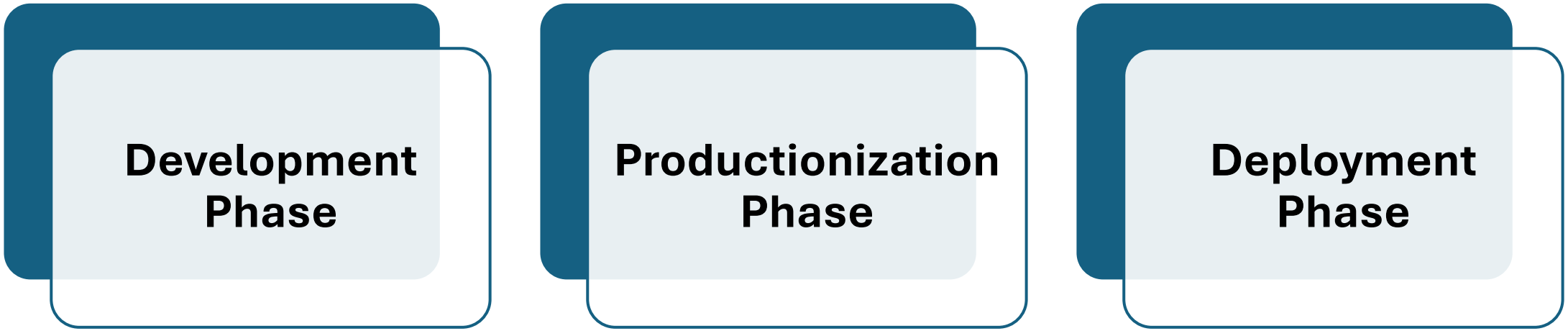
SIMPLIFIES
EVERYTHING

FROM DEVELOPMENT
TO DEPLOYMENT

OF LLM-BASED
SOFTWARE SYSTEMS.


Vectorstores
Prompts
Document Loaders
LangChain
Agents
LLMs
Chains
ProjectPro

Key Capabilities at a Glance



1. Development Phase

USE **OPEN-SOURCE COMPONENTS**

THIRD-PARTY TOOLS

TO BUILD **LLM APPLICATIONS.**

1. Development Phase

LangGraph

enables

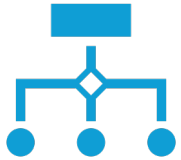
to create

stateful agents

1. Development Phase

Support

Streaming outputs



Human-in-the-loop feedback

Tool usage coordination

2. Productionization Phase

LangSmith

Inspect and debug LLM applications

Monitor performance

Evaluate accuracy & quality

Continuously optimize models and prompts

3. Deployment Phase

Turn LangGraph apps

Into APIs

Production-ready AI Assistants

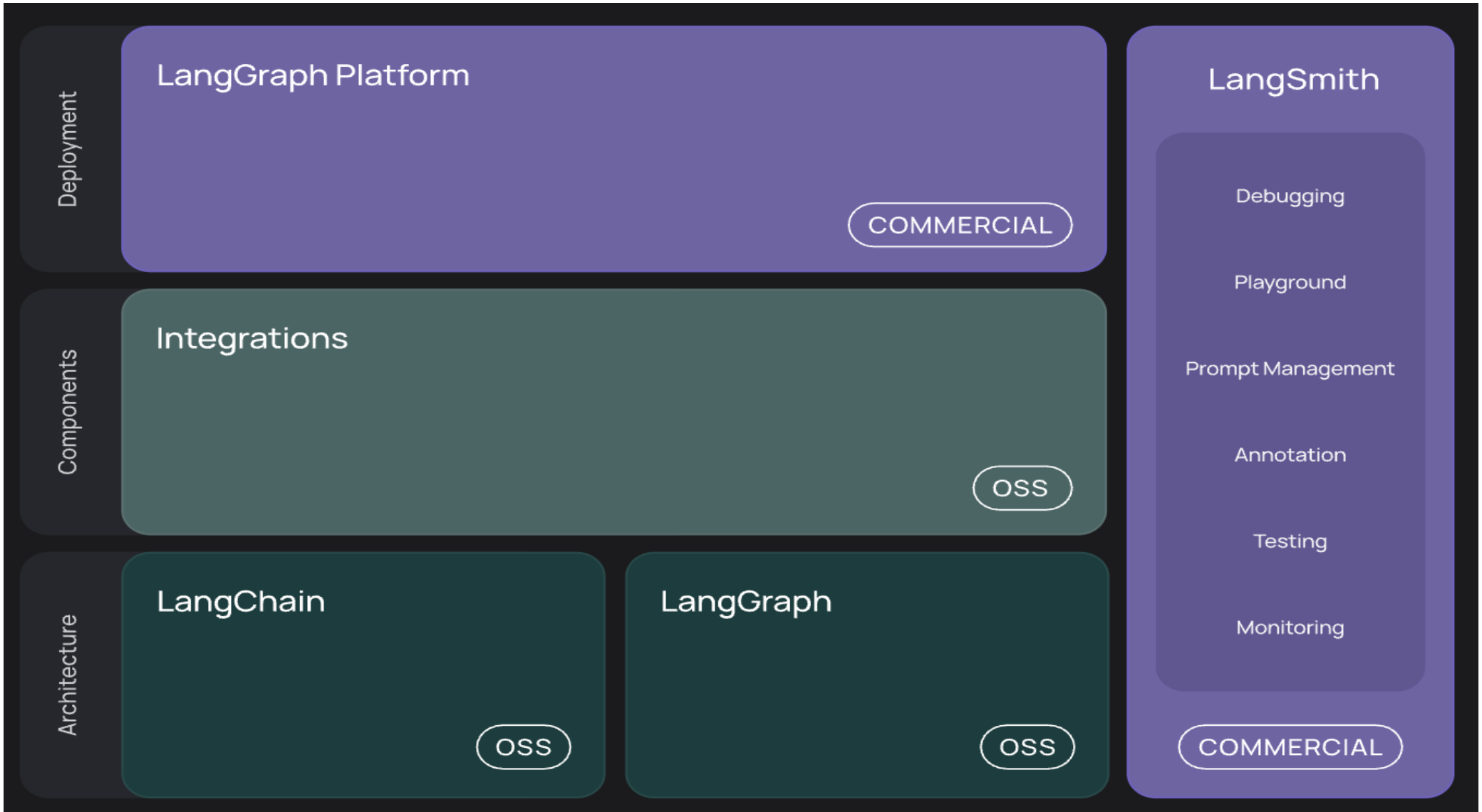
3. Deployment Phase

USE **LANGGRAPH PLATFORM**

FOR CLOUD-BASED HOSTING

AND ORCHESTRATION

LangChain + LangGraph Architecture



Architecture Overview

Module	Purpose
langchain-core	Base interfaces & abstractions for LLMs, tools, and prompts
langchain	Contains higher-level components like agents, chains, and retrieval strategies

Architecture Overview

Module	Purpose
langchain-openai, langchain-anthropic,	Integration packages maintained by LangChain + provider teams
langchain-community	Third-party and community-contributed integrations

Architecture Overview

Module	Purpose
langgraph	Agent orchestration framework with support for memory, streaming, and persistence
LangSmith	Observability and evaluation platform for LangChain apps
LangGraph Platform	Cloud deployment & lifecycle management of LangChain applications

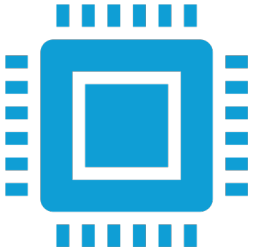
Reference

- https://python.langchain.com/docs/introduction/

Ecosystem Integration

LangChain integrates

with **hundreds of tools**,



models, and providers

GenAI Tools Providers

OpenAI,

Anthropic,

Hugging Face,

Cohere,

Google Vertex

Vector stores

Pinecone,

Chroma,

FAISS

Tools

Search APIs,

SQL databases,

file readers,

Python REPL

Cognitive Architecture Features

LangChain’s design

encourages building apps

that simulate thinking agents with:

Memory, Tool usage

Multi-step planning,

Retrieval-Augmented Generation (RAG)

Cognitive Architecture Features

Memory: Store past interactions

Tool usage: Perform calculations, search, look up info

Multi-step planning: Use planners, routers, chains

Retrieval-Augmented Generation (RAG)

Fetch relevant knowledge before answering

Summary of Key Points

Category	Summary
Framework Type	Agentic LLM application development
Core Components	Chains, Agents, Tools, LangGraph
Lifecycle Coverage	Build → Monitor (LangSmith) → Deploy (LangGraph Platform)

Summary of Key Points

Category	Summary
Extensibility	Integrates with major LLM providers and tools
Production Readiness	Enables streaming, persistence, tracing, and evaluation

Why LangChain?

LangChain makes it easy

to build complex GenAI pipelines using:

Document Loaders

Retrievers

Chains/Agents

Why LangChain?

Document Loaders:

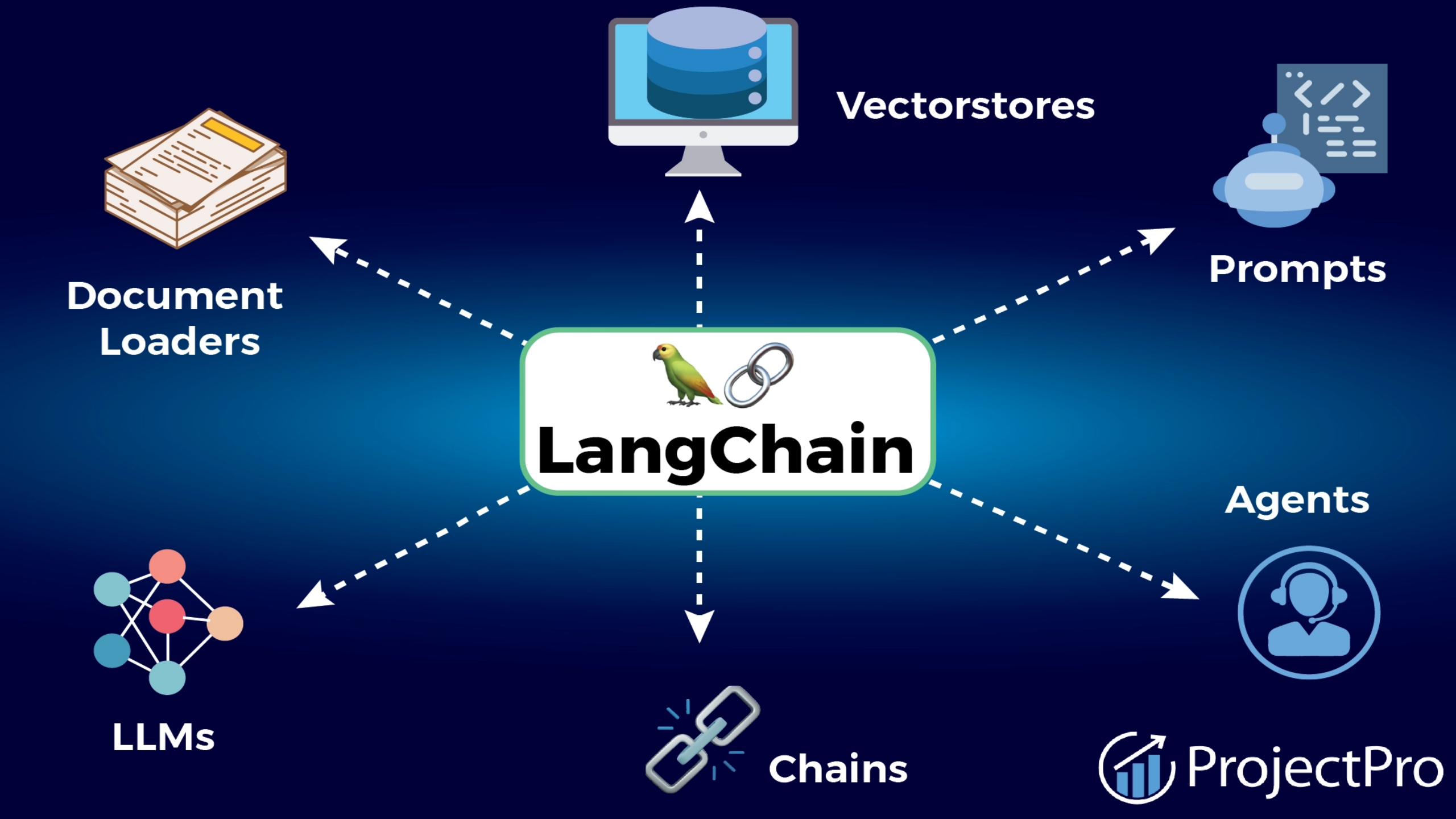
bring in PDFs, Excel, web pages

Retrievers:

fetch relevant info via embeddings

Chains/Agents: orchestrate workflows

e.g., search → generate → respond


Vectorstores
Prompts
Document Loaders
LangChain
Agents
LLMs
Chains
ProjectPro

Semantic vs Native Functions

Aspect	Native Function	Semantic Function
Definition	A function implemented in code (e.g., Python/ C# method)	A function defined via a prompt template calling an LLM
Execution Mode	Executes code directly on input data	Sends input to LLM via prompt, returns result
Use Case	Simple logic, deterministic operations (e.g., formatting)	Complex language tasks: summarization, rewriting, creative
Predictability	High – result based on known code	Less deterministic – depends on LLM output

Semantic vs Native Functions

Aspect	Native Function	Semantic Function
Integration point	Low latency, small compute footprint	Higher latency, uses LLM service
When to use	When you need reliable, rule-based operations	When you need language understanding / generation
Example	def count_words(text): return len(text.split())	Prompt: “Summarize the following text in 5 sentences…”

SK vs LangChain for Similar Task

Feature	LangChain	Semantic Kernel (SK)
Primary Focus	Chains of LLM calls + tools; workflow orchestration	Plugin/skill-based architecture; code + prompt functions
Prompt Abstraction	PromptTemplate + Chains	Semantic functions + native functions in plugin form
Tool / Service Integration	Tools as objects (e.g., GoogleSearchTool)	Plugins registered in Kernel; AI services swapped easily
Modularity	Chains can be composed	Plugins + planners + memory offer high modularity

SK vs LangChain for Similar Task

Feature	LangChain	Semantic Kernel (SK)
Enterprise Governance	Requires building wrappers	Kernel supports logging, middleware, plugin/ service monitoring
Complexity For Simple Tasks	Straightforward for simple flows	Slightly more setup but scalable for multi-agent & plugin scenarios
Best Use Case	Build applications centered around LLM + retrieval + tool use	Build enterprise-colonial agentic systems, reuse code + prompt, integrate deeply with services
Example Implementation (Summarization)	Create a PromptTemplate -> LLMChain -> Run	Register service, define semantic function plugin, run via kernel

Embedding & Vector Store

Managed Milvus:

Vector DB-as-a-Service

Embedding generation pipeline:

Sentence Transformers, HuggingFace

Document ingestion and search

What is Managed Milvus?

A vector database-as-a-service

optimized for similarity search

large-scale embeddings .

What is Managed Milvus?

Built for storage,

query,

indexing,

scaling across billions of vectors

What is Managed Milvus?

Offers both

open-source standalone and

fully **managed cloud service**

options via Zilliz Cloud .

Embedding Generation Pipeline

Use Sentence Transformers or

HuggingFace models to convert text

(product titles, reviews, policies)

into dense vectors

Document Ingestion & Vector DB Setup

Load documents: CSVs of product data,

PDFs of policy docs

Generate embeddings

Insert into Milvus

Document Ingestion & Vector DB Setup

Supports **auto-scaling**, **sharding**, and **fault tolerance** .

Zilliz Cloud offers fully managed infrastructure for ease and performance .

Summary

Stage	What Happens	Impact
Embedding Generation	Convert text/images → numeric vectors	Meaningful search, similarity matching
Vector Ingestion	Store embeddings in Milvus database	Scalable, low-latency retrieval
Semantic Search	Find similar vectors using nearest neighbor	Better product recommendations, search UX
RAG + LLM Integration	Use retrieved docs as context for LLM responses	Accurate chatbot answers, grounded responses

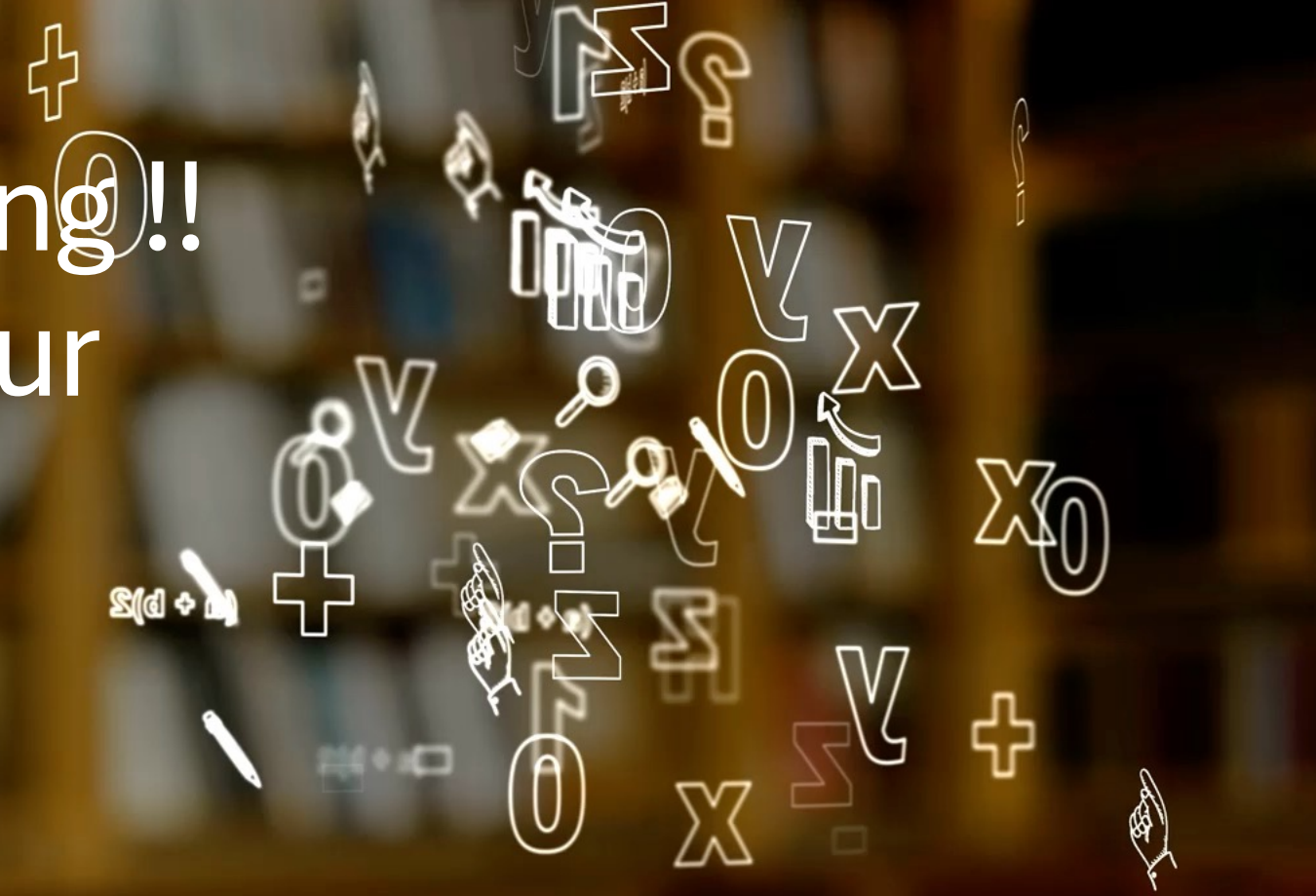
Hands-On Session

Generate embeddings from text documents

Store and query documents using Milvus vector DB

Perform similarity-based document retrieval

Happy Learning!! Thanks for Your Patience ☺



Surendra Panpaliya

GKTCS Innovations