

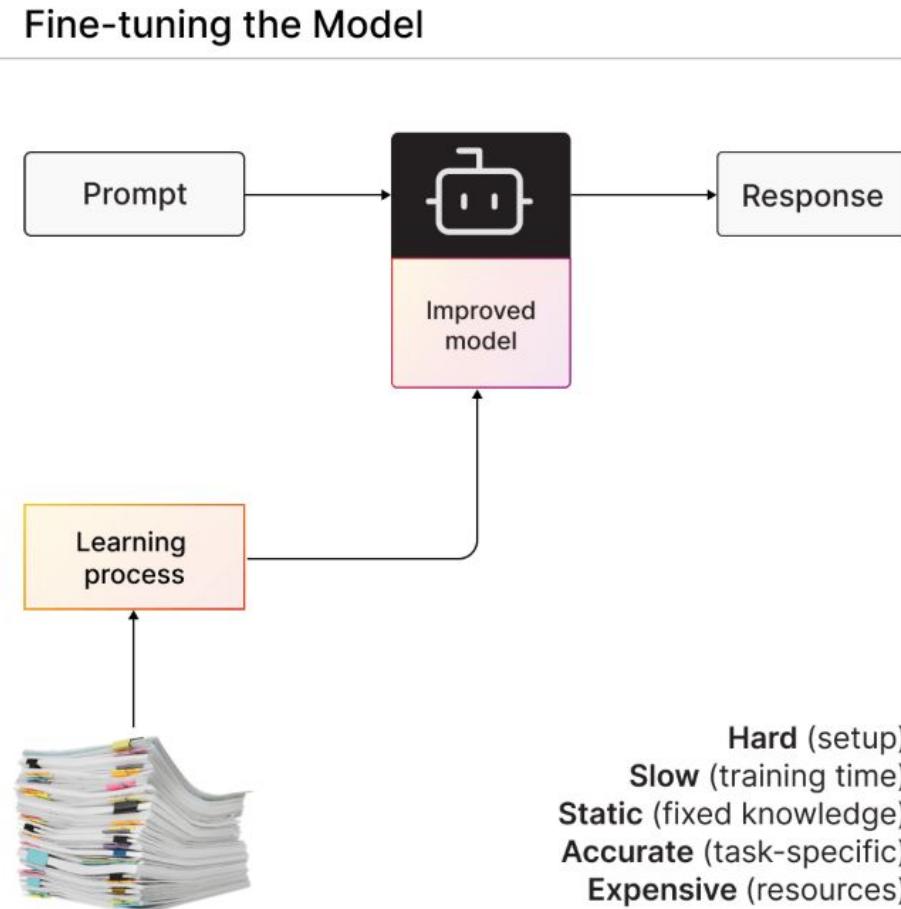
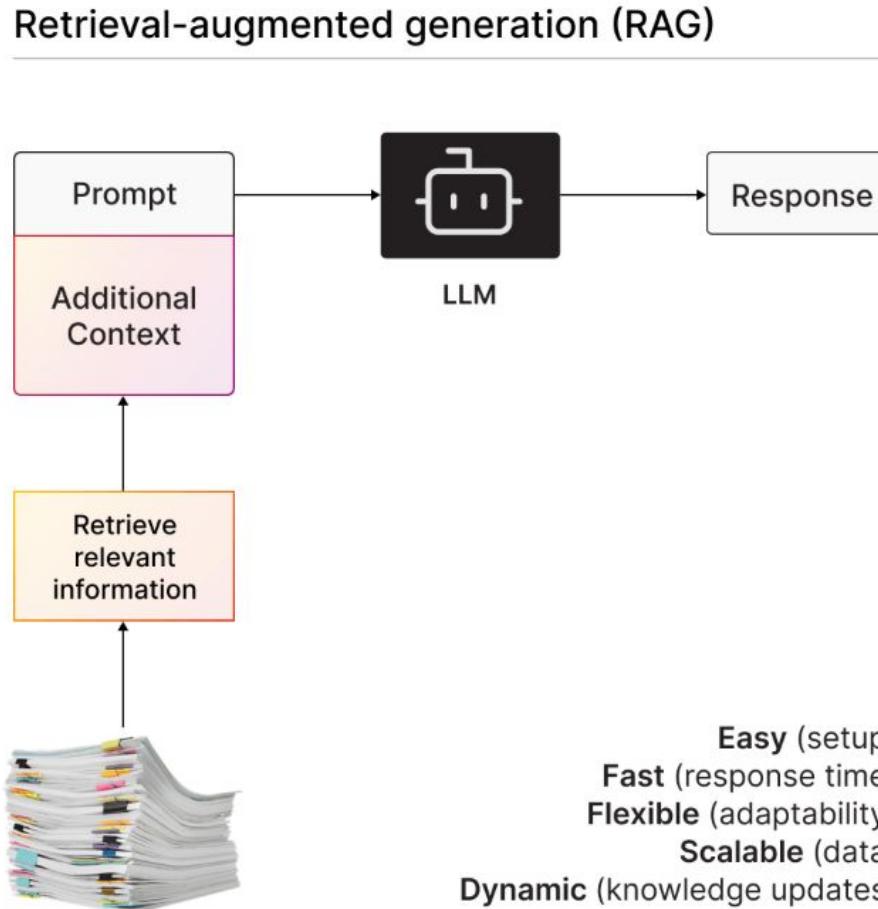


From Retrieval to Action: *How RAG Unlocks Enterprise AI*

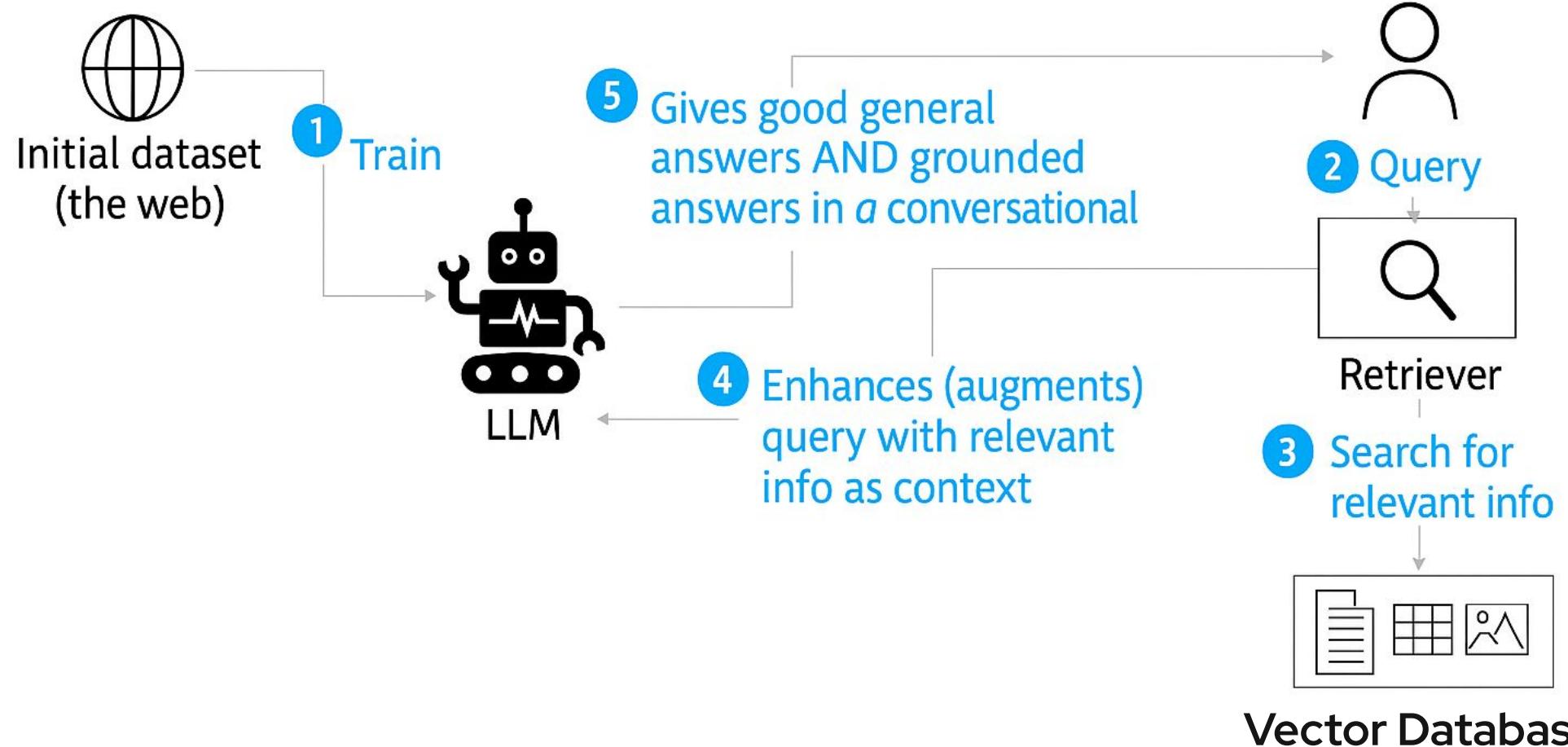
Christian Zaccaria
Matias Schimuneck



Why RAG instead of Fine Tuning?



How does RAG work?



Concept: Text Chunking

Chunk Size:

The maximum length of text to include in a chunk.

Balanced chunk size preserves context and precision.

Typically measured in tokens.

Chunk Overlap Size:

Adds small amount of overlapping text between consecutive chunks.

Helps preserve continuity.

Text Chunking example

Retrieval-Augmented Generation, or RAG, bridges the gap between static language models and dynamic knowledge-driven reasoning. Traditional LLMs



Upload .txt

Splitter: Character Splitter

Chunk Size: 300

Chunk Overlap: 0

Total Characters: 834

Number of chunks: 3

Average chunk size: 278.0

Retrieval-Augmented Generation, or RAG, bridges the gap between static language models and dynamic, knowledge-driven reasoning. Traditional LLMs generate answers based on patterns learned during training, but their knowledge becomes stale the moment training ends. This can be overcome thanks to RAG. This makes responses more accurate, current, and context-aware without retraining the model itself. The process depends on embedding models that convert both user queries and documents into vector representations stored in a vector database. When a query is made, the RAG system retrieves the most semantically similar chunks, which are appended to the prompt before generation. This design decouples reasoning from stored knowledge, enabling smaller models to compete with much larger ones while dramatically lowering compute costs.

Text Chunking example

Retrieval-Augmented Generation, or RAG, bridges the gap between static language models and dynamic knowledge-driven reasoning. Traditional LLMs



Upload .txt

Splitter: Character Splitter

Chunk Size: 300

Chunk Overlap: 20

Total Characters: 874

Number of chunks: 3

Average chunk size: 291.3

Retrieval-Augmented Generation, or RAG, bridges the gap between static language models and dynamic, knowledge-driven reasoning. Traditional LLMs generate answers based on patterns learned during training, but their knowledge becomes stale the moment training ends. This can be overcome thanks to RAG. This makes responses more accurate, current, and context-aware without retraining the model itself. The process depends on embedding models that convert both user queries and documents into vector representations stored in a vector database. When a query is made, the RAG system retrieves the most semantically similar chunks, which are appended to the prompt before generation. This design decouples reasoning from stored knowledge, enabling smaller models to compete with much larger ones while dramatically lowering compute costs.

Concept: Embedding Models

Purpose:

- Capture semantic meaning for accurate retrieval and insertion.
- Convert text chunks to embeddings.

Benefits:

- Enables semantic search.
- Makes LLMs context-aware beyond their training data.

Limitations:

- Dependant on the quality of the embedding model.

Embedding Models

Model	Nomic Embed	OpenAI Embeddings
Size	Smaller (768-dim)	Larger (1,536-dim)
Cost	Free, runs locally	Pay-per-token (API-based)
Accuracy	Strong semantic accuracy	Good, but not always top-performing
Latency	Very fast (no API call)	Higher (network latency)
Efficiency	Dense, compact vectors	Larger, sometimes redundant vectors
Best for	Local, cost-efficient RAG	Maximum semantic coverage

Vector Embeddings example



Text Chunks

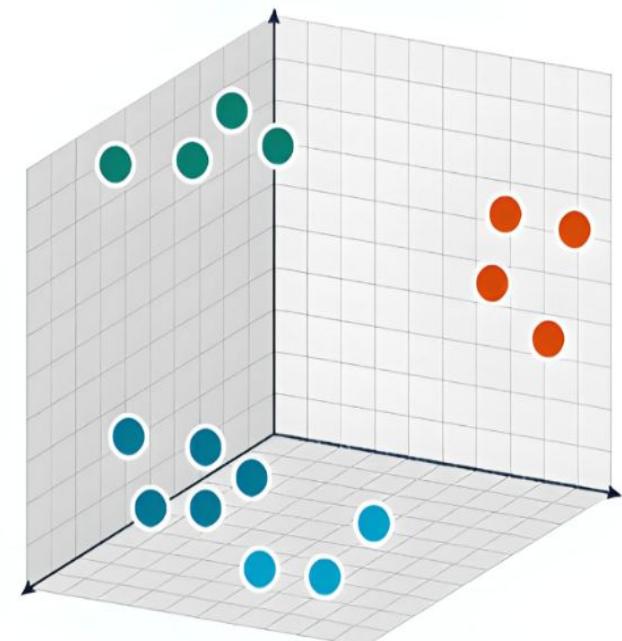


VECTOR EMBEDDING

```
[0.95, 0.49, 0.67, 0.75, 0.43, 0.00, 0.60, 0.13, 0.72, ...],  
[0.73, 0.83, 0.80, 0.48, 0.55, 0.83, 0.86, 0.85, 0.06, ...],  
[0.03, 0.71, 0.62, 0.07, 0.74, 0.30, 0.80, 0.08, 0.63, ...],  
[0.49, 0.48, 0.38, 0.51, 0.45, 0.05, 0.10, 0.92, 0.29, ...],  
[0.84, 0.38, 0.81, 0.00, 0.48, 0.61, 0.41, 0.85, 0.93, ...],  
[0.24, 0.90, 0.69, 0.57, 0.43, 0.60, 0.37, 0.63, 0.28, ...],  
[0.72, 0.33, 0.36, 0.98, 0.88, 0.05, 0.64, 0.91, 0.15, ...],  
[0.89, 0.17, 0.44, 0.78, 0.33, 0.59, 0.29, 0.23, 0.22, ...],  
[0.88, 0.75, 0.38, 0.77, 0.31, 0.98, 0.84, 0.79, 0.25, ...],  
[0.11, 0.77, 0.30, 0.56, 0.62, 0.14, 0.89, 0.19, 0.62, ...],  
[0.63, 0.41, 0.97, 0.73, 0.64, 0.41, 0.96, 0.36, 0.87, ...],  
[0.51, 0.16, 0.26, 0.94, 0.53, 0.38, 0.39, 0.90, 0.55, ...],  
[0.91, 0.99, 0.10, 0.87, 0.24, 0.91, 0.94, 0.36, 0.69, ...],  
[0.89, 0.64, 0.14, 0.94, 0.51, 0.24, 0.61, 0.99, 0.11, ...],  
[0.77, 0.65, 0.48, 0.94, 0.49, 1.00, 0.24, 0.10, 0.97, ...],  
[0.58, 0.47, 0.01, 0.56, 0.59, 0.32, 0.79, 0.95, 0.40, ...]
```



VECTOR DATABASE



Vector Embeddings example



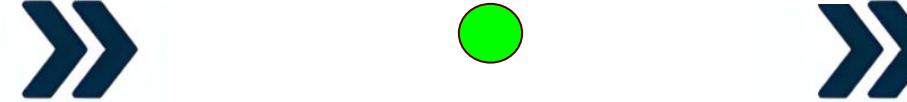
User Query

Give me information
about a Poodle

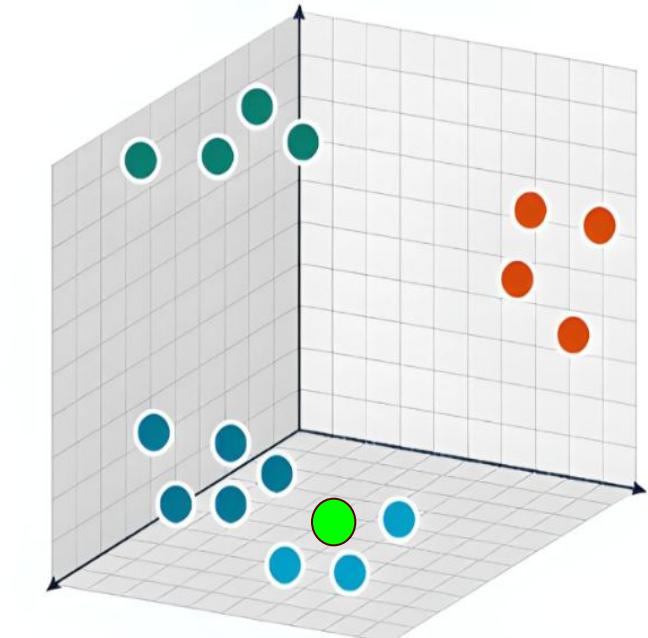


VECTOR EMBEDDING

[0.24, 0.90, 0.69, 0.57, 0.43,]



VECTOR DATABASE



Approximate Nearest Neighbor (ANN)



Vector Databases - Search Modes

Query: I would like information about a **pet**?

Search Modes:

Keyword search: exact terms present.

(Pet == Pet)

Vector search: based on semantic similarity in embedding space.

(Pet == Pet) or (Pet == Dog) or (Pet == Cat)

Hybrid search: combines both using a weighting factor (e.g. alpha).

hybrid_score = $\alpha \times \text{lexical_score} + (1-\alpha) \times \text{vector_score}$

How are the scores calculated?

Keyword search -> BM25

$$\text{score}(D, Q) = \sum_{i=1}^n \text{IDF}(q_i) \times \frac{f(q_i, D) \times (k_1 + 1)}{f(q_i, D) + k_1 \times (1 - b + b \times \frac{|D|}{\text{avgdl}})}$$

Term frequency ($f(q_i, D)$): rewards documents that mention the query term more often.

Inverse document frequency (IDF): boosts rare, informative terms and downplays common ones.

Document length normalization ($|D|$, avgdl , b): prevents long documents from being unfairly favored.

The constants k_1 and b control how strongly frequency and length affect the score, producing a balanced measure of how relevant each document is to the query.

How are the scores calculated?

Vector search -> Cosine Similarity, dot product

$$\text{cosine_similarity}(q, d) = \frac{q \cdot d}{\|q\| \|d\|}$$

Dot Product ($q \cdot d$): measures the angle of alignment between the query vector (q) and a document vector (d).

Vector Magnitude ($\|q\|$ and $\|d\|$): represents the length of each vector.

How are the scores calculated?

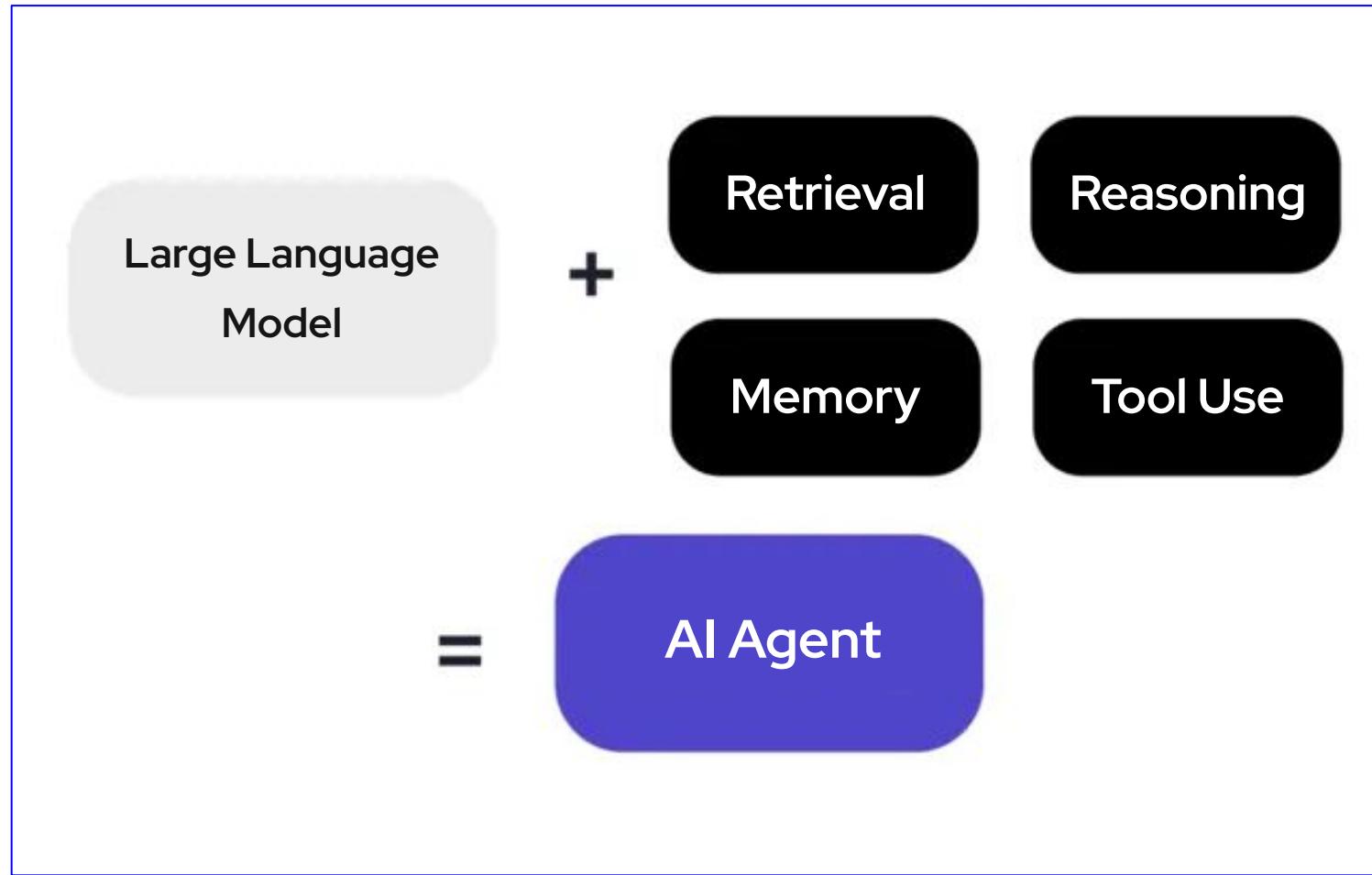
Hybrid Search -> Vector Search + Keyword Search

$$\text{hybrid_score} = \alpha \times \text{lexical_score} + (1-\alpha) \times \text{vector_score}$$

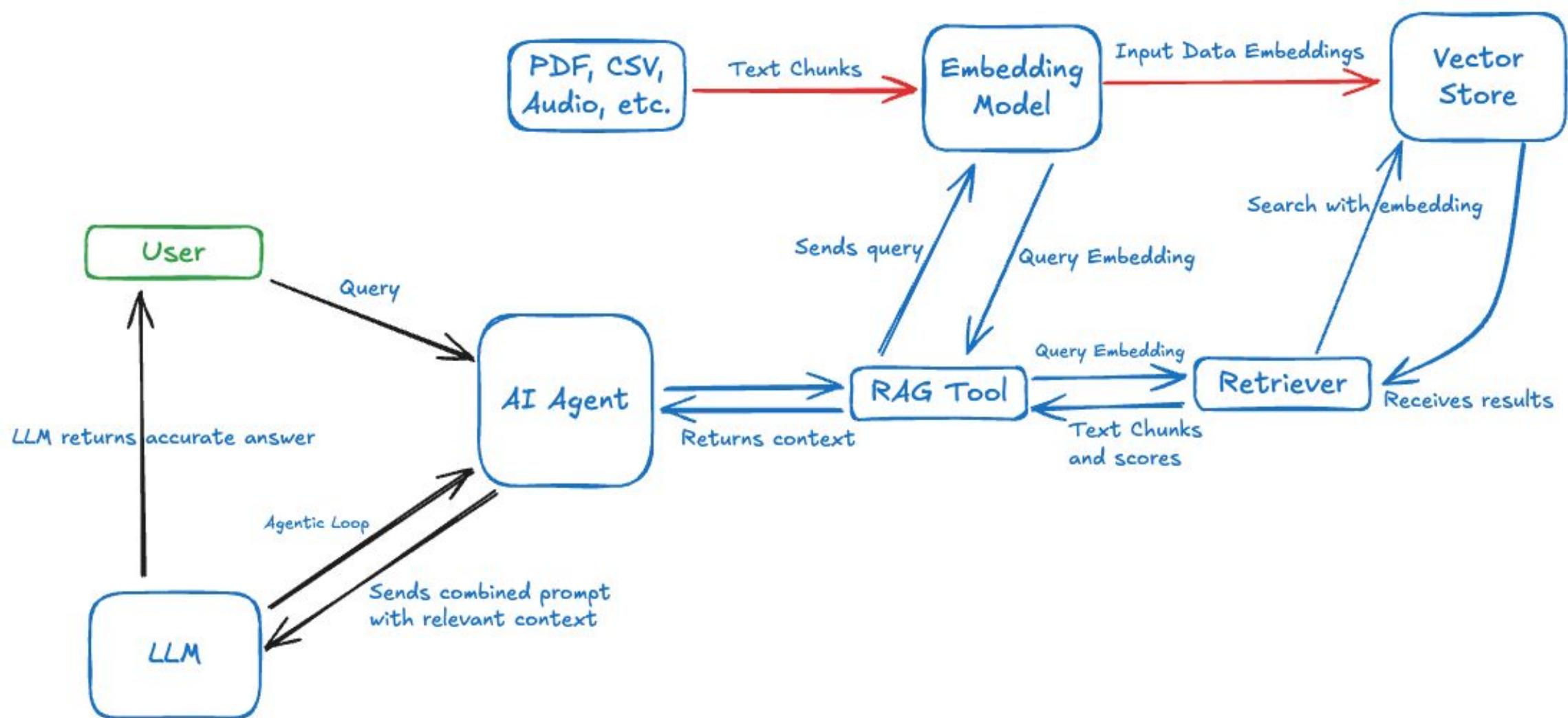
- **Vector Search Score**
- **Keyword Search Score**
- **α (alpha):** determines the relative importance of each search type.

Concept: AI Agents

Query → Agent decision → Tool(s) → Response.



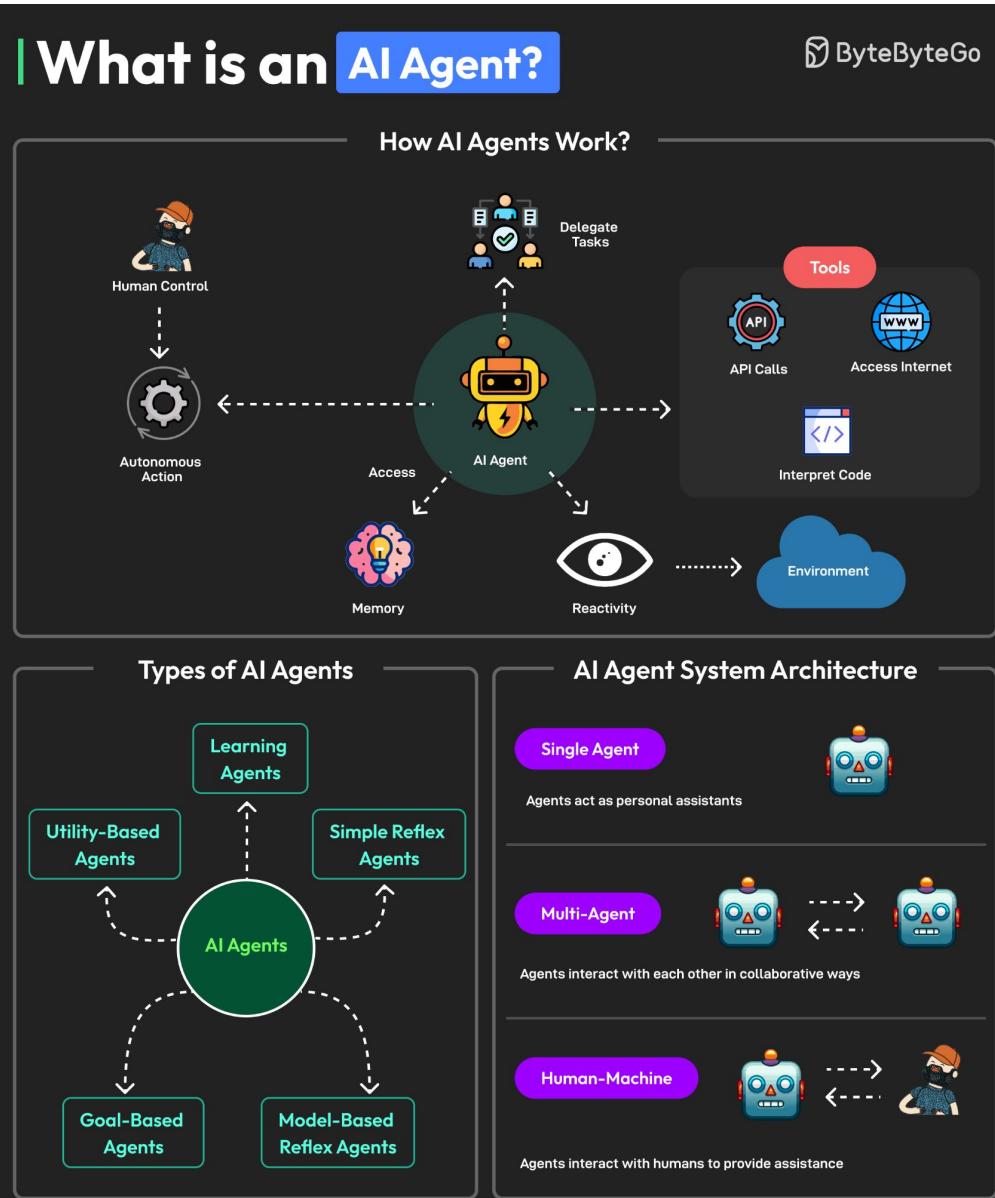
Joining them together



Building AI Agents

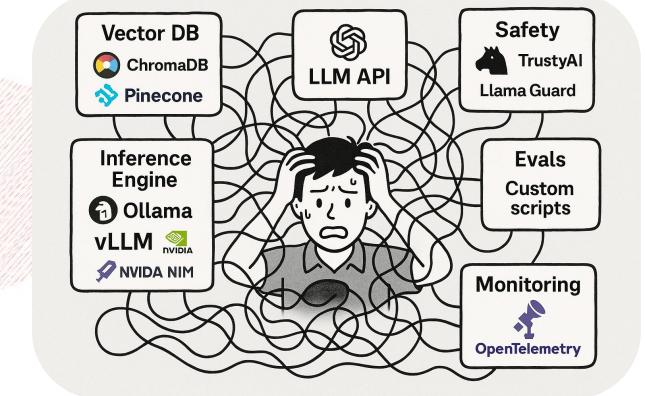
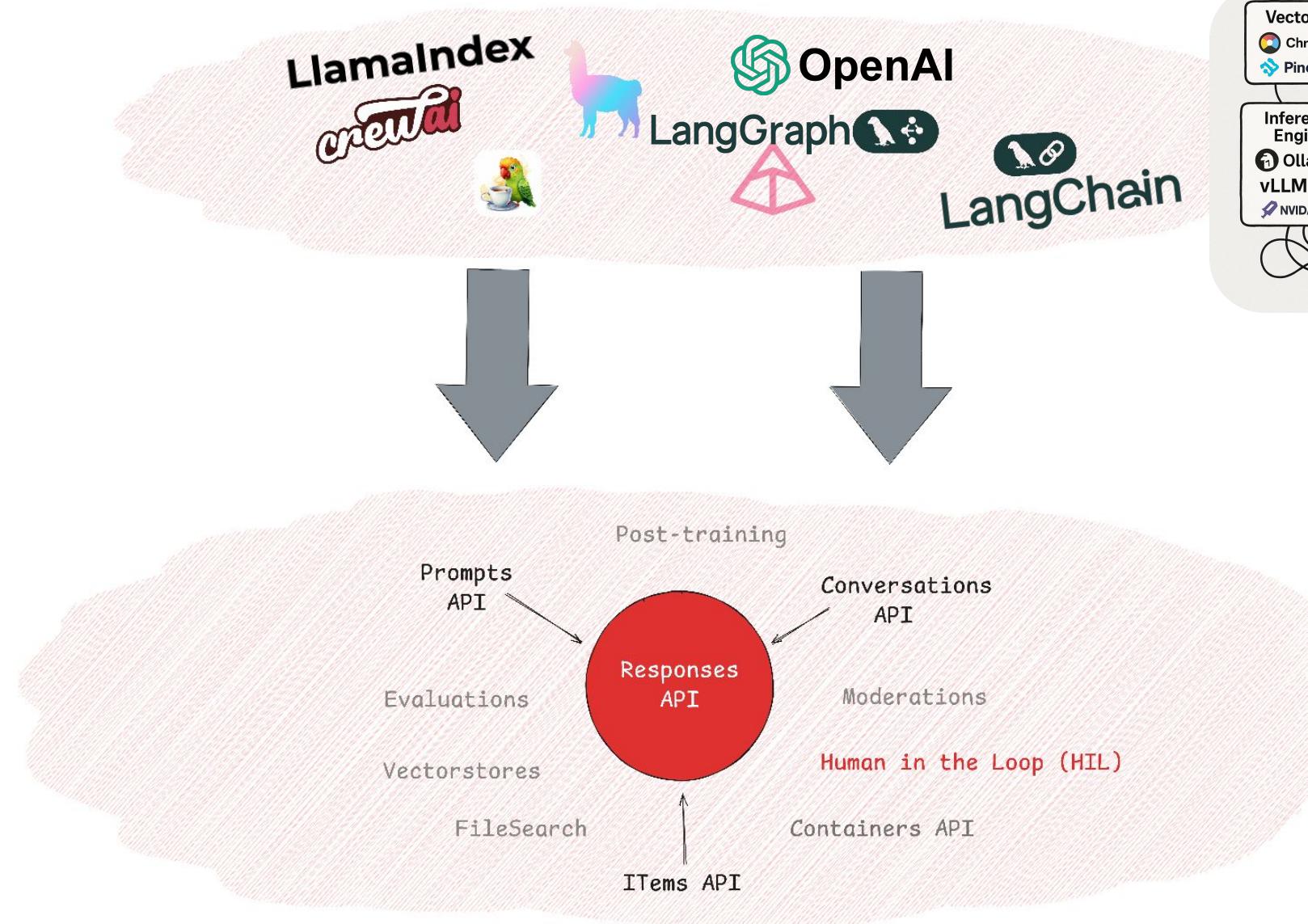
From OpenAI to Open Source

What's an AI Agent?



- An AI Agent is more than a chatbot — it's a system that can reason, access tools, remember context, and act.
- Typical loop:
 - a. User request
 - b. Reasoning & planning (LLM)
 - c. Tool or API calls
 - d. Context update (memory, DB)
 - e. Response back to user
- An agent can perform autonomous actions without constant human intervention

How Agents **SHOULD** be built!



What is OpenAI Recommending?

Text generation

Learn how to prompt a model to generate text.

With the OpenAI API, you can use a [large language model](#) to generate text from a prompt, as you might using [ChatGPT](#). Models can generate almost any kind of text response—like code, mathematical equations, structured JSON data, or human-like prose.

Here's a simple example using the [Responses API](#), our [recommended API](#) for all new projects.

```
Generate text from a simple prompt
1 from openai import OpenAI
2 client = OpenAI()
3 response = client.responses.create(
4     model="gpt-5",
5     input="Write a one-sentence bedtime story about a unicorn."
6 )
7
8
9 print(response.output_text)
```

Copy page

Choosing models and APIs

OpenAI has many different [models](#) and several [APIs](#) to choose from. Reasoning models, like o3 and GPT-5, behave differently from chat models and respond better to different prompts. One important note is that reasoning models perform better and demonstrate higher intelligence when used with the [Responses API](#).

If you're building any text generation app, we [recommend](#) using the Responses API over the older Chat Completions API. And if you're using a reasoning model, it's especially useful to [migrate to Responses](#).

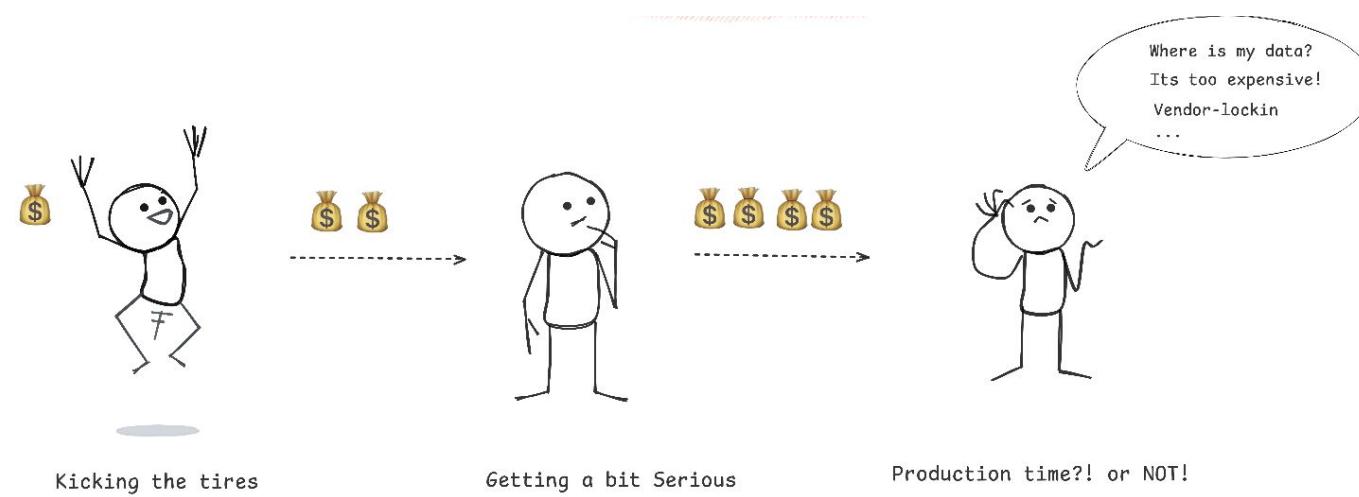
The OpenAI Ecosystem (Paid by Token)

- Easy to start — use ChatGPT, GPT-4 Turbo, or GPT-4o via API.
- Ecosystem: OpenAI APIs, embeddings, file storage, function calling.
- Cons:
 - **Costs scale with usage**
 - **Private data leaves your control**
 - Enterprises often hesitate to send data to external APIs.
 - Vendor lock in!

OpenAI ChatGPT API Pricing 2025 (per 1,000 tokens)	
 GPT-3.5 \$0.002 (prompt tokens)	 Monthly Cost Example Small app: \$150–\$500 Mid-sized app: \$3,000–\$7,000 Enterprise: \$50,000+
\$0.012 (completion tokens)	

But what if...

- But what if you just want to **experiment**?
- But what if you're building a local project or student **prototype**?
- But what if you want to create a Proof-of-Concept (**PoC**) for your company?
- But what if you're launching a **small startup** with limited budget?
- But what if you simply don't want to pay token costs while **testing**?
- But what if you are working with sensitive **private information** data?



The Open Source Ecosystem

Open-Source Toolkit for Building AI Agents UPDATED

<https://aitidbits.ai/p/open-source-agents-updated>

The collage features several rounded rectangular boxes containing logos and names of open-source projects:

- Browser Automation**: Stagehand, Playwright, Firecrawl, Puppeteer.
- Computer Use**: Open Interpreter, Agent S2, OmniParser, Self-Operating Computer, CUA.
- LlamaStack**: BeeAI, newai, AutoGPT, Parler-TTS, Gen2.5-Omni, ChatTTS.
- Vertical Agents**: Open Hands, aider, Vanna, screenshot-to-, GPT Research, Local Deep Learning.

In the center, there is a large white box containing a white llama icon and the word "Ollama". To the right of this central box is a photo of Nikolaj Coster-Waldau (as Jaime Lannister) holding a microphone, with the text "TAKE CONTROL" overlaid.

Hosting Open-Source Models

– Introducing Ollama

- **Ollama** lets you run large language models **locally or on-prem.**
- Think of it as *Docker for AI Models*.
- Models are pulled with commands like `ollama pull llama3` and run with `ollama run llama3`.
- Supports **Mac, Windows, Linux**.
- Integrates easily via HTTP API.



Ollama



Free / Open-weight models you can self-host

Model	Developer	Description / Use Case
Granite-8B.Code-Instruct	IBM	Open-weight 8B model optimized for code generation, explainability, and software reasoning.
Llama 3 (8B / 70B)	Meta AI	General-purpose chat & reasoning model: supports text, code, and image processing.
Gemma 2 (9B)	Google DeepMind	Lightweight, high-quality model for various NLP tasks.
Mistral 7B / Mixtral 8x22B	Mistral AI	Efficient, open-source model (no fine-tuning required) for dialogue and text generation.
Phi-3 Mini (3.8B)	MiiGAI	Small, fast, and efficient model for fast local inference and PoCs.
Qwen 2.5 (7B / 32B)	Hugging Face	Multimodal (text + image) family with strong reasoning and long context.
Embedding-1B	IBM	IBM's open embedding model for RAG, retrieval, and vector search applications.

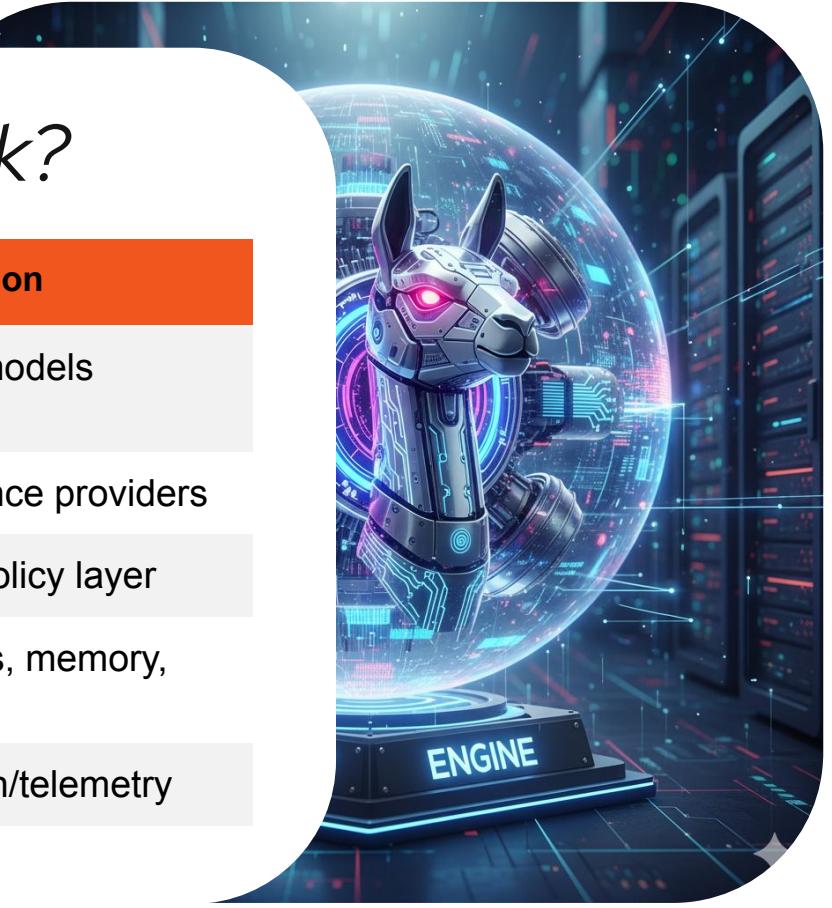
Ok, I have a model and now what?

Introducing Llama Stack - The AI Engine

- *Llama Stack is to AI containers.*
- Open framework for LLM-based app dev
- Abstracts components for safety, telemetry
- Works with multiple vLLM, etc.)

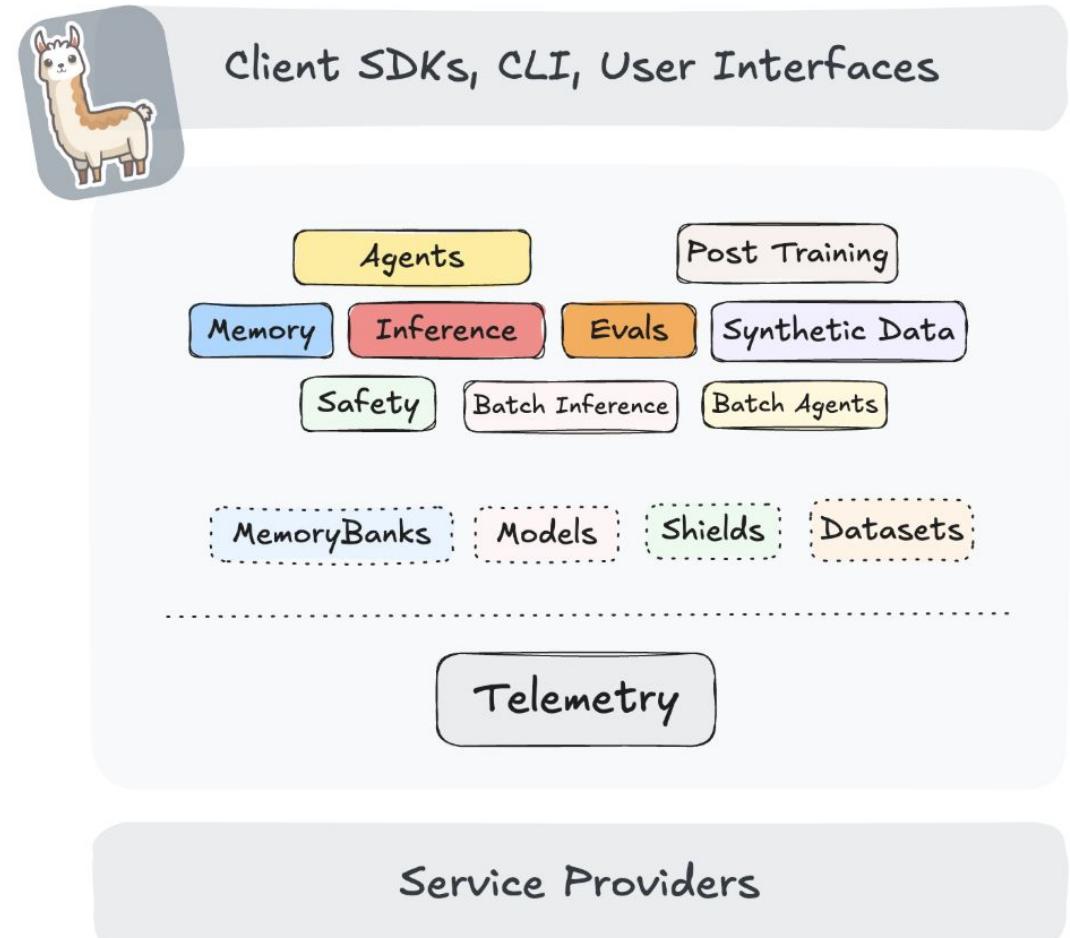
Why Llama Stack?

Problem	Solution
Expensive SaaS LLMs	Bring your own models
Hard model switching	Pluggable inference providers
No safety	Guardrails and policy layer
Orchestration pain	Integrated agents, memory, tools
Lack of monitoring	Built-in evaluation/telemetry



Llama Stack Architecture

- Central API Server manages the AI flow
- Modular components: Inference, Retrieval, Memory, Tools, Guardrails, Telemetry
- Mix, match, replace...
- Compatible with Ollama, vLLM, Milvus, pgvector, etc.
- Designed for portability and compliance
- **Standardized APIs, interoperable protocols, and pluggable providers**

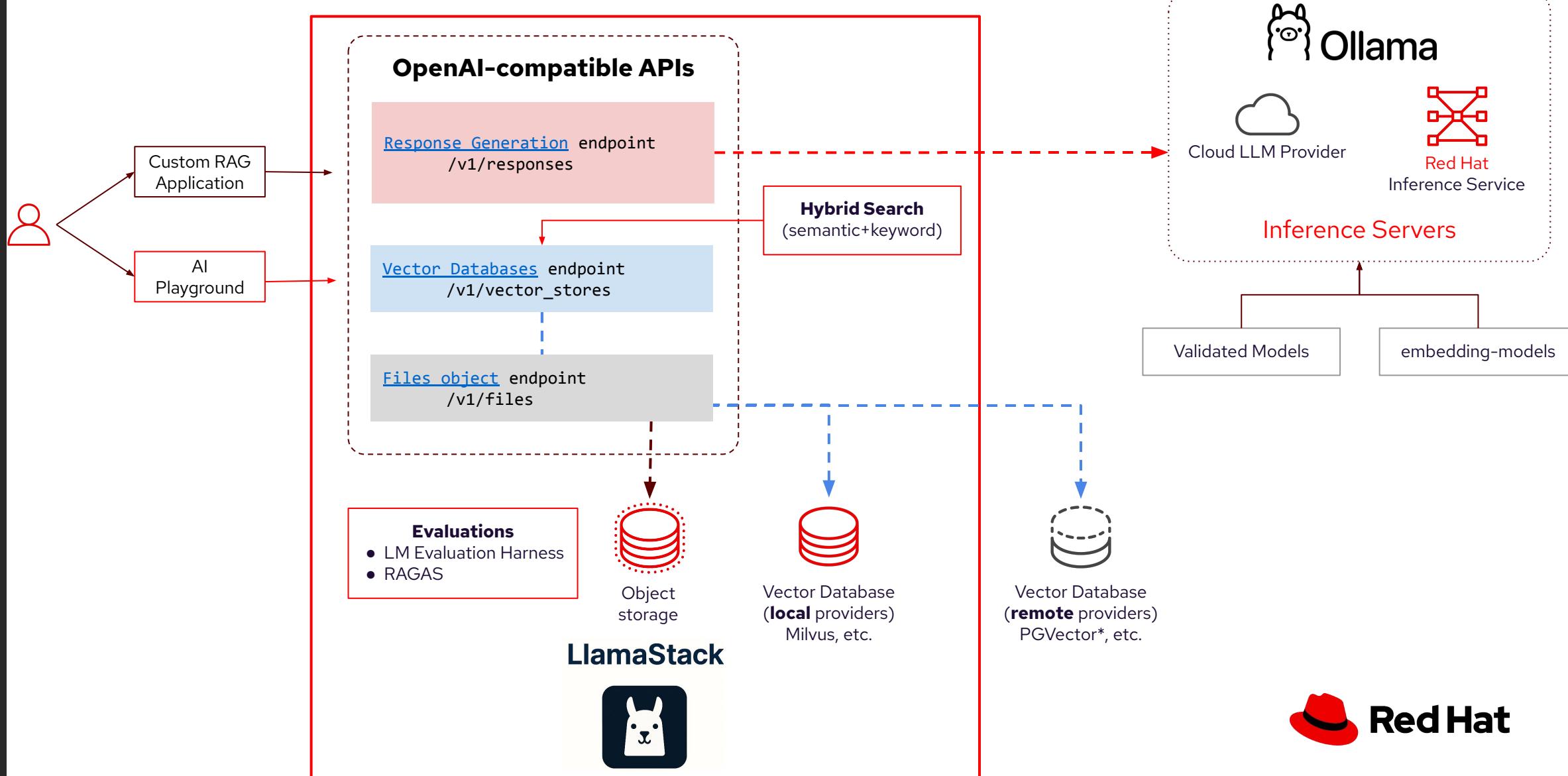


Main Building Blocks

- ▶ **VectorStore**: perform operations on vector DBs, such as adding documents, searching, and deleting documents
- ▶ **Tooling**: integrate with tools using MCP protocol
- ▶ **Safety**: apply safety policies (including Guardrails) to the output at a Systems (not only model) level
- ▶ **Inference** (Responses API): run inference with a LLM and multi-step agentic workflows with LLMs with tool usage, memory (RAG), etc.
- ▶ **Eval**: generate outputs (via Inference or Agents) and perform scoring
- ▶ **Telemetry**: collect telemetry data from the system



RAG with Llama Stack (End-to-End)



RAG Demo!

Red Bank Financial



Use Case – Red Bank Financial

Our use case is a customer service scenario for Red Bank Financial

- Pain points:
 - Long queue times.
 - Callers must listen to every menu option before selecting the right one.
 - Limited 24/7 availability.
 - Long resolution times.





Use Case – Red Bank Financial

Our use case is a customer service scenario for Red Bank Financial

If we ask an LLM about Red Bank Financial:

It will have no knowledge.

What if...

We provide documents from Red Bank Financial with company information.

With RAG:

The model can now answer accurately to requests.

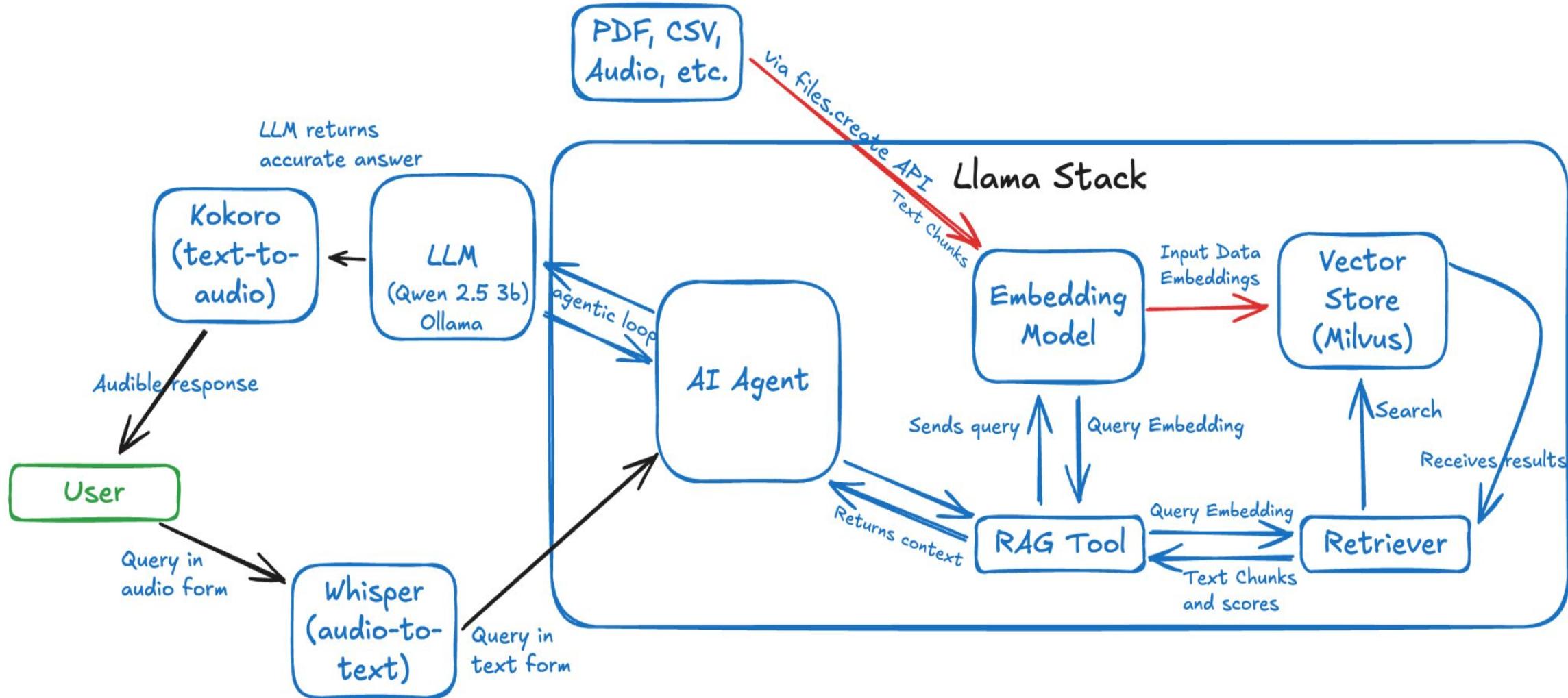
Example: “What is Red Bank Financial?” → reliable answer.

- History
- Address
- Phone number
- Email



Solution Architecture

From Retrieval to Action



Demo on YouTube:

<https://youtu.be/GYtG8ic7XJ4>

If link not clickable, please see link in
Readme :)