



Advanced Gen-AI Development

Hany Saad
hghiett@iti.gov.eg



Course: Developing Gen-AI based software



Course Title: AI-powered Coding Assistant tools

Tracks: Development tracks only (Backend development tracks only)

Duration: 18 hours (3 Lectures + 3 Labs)

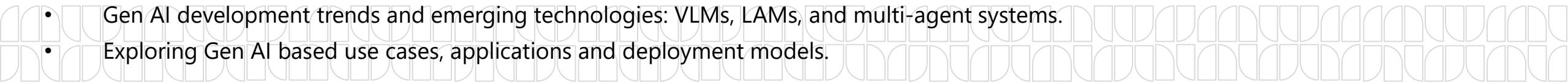
Evaluation: Attendance + Labs.

Prerequisites:

- Intro to Generative AI and prompt engineering.
- REST API (To be after Web API course in each dev track)
- Require paid subscription (OpenAI API Key, initial cost \$5 can be used for multiple tracks), or to use Open-source models

Outlines:

- Gen AI Providers, models and capabilities (OpenAI, Google, Azure OpenAI Service, and Anthropic models): Text, Vision, Image, TTS, STT models.
- Working with OpenAI APIs: Text completions, Image generation and vision APIs (OpenAI APIs or open-source models APIs)
- Types of Prompts: System, assistant, and user prompts, and how to write better prompts for better results.
- Advanced API features: Structured output, Assistant API, Function calling.
- Foundation models and models fine-tuning.
- Open-source models and models catalog platforms: Hugging-face, MS Azure AI, and GitHub models.
- Intro to RAG and AI orchestration platforms: Langchain and MS Semantic Kernel.
- Intro to Agentic AI frameworks and tools for AI Agents development.
- Gen AI development trends and emerging technologies: VLMs, LAMs, and multi-agent systems.
- Exploring Gen AI based use cases, applications and deployment models.



Gen-AI based startups and real products



- **Use cases for real products using Gen AI:**
 - Loop-x (AI Agents for different roles): <https://loop-x.co>
 - Gemelo ai (Twin AI creator): <https://gemelo.ai>
 - Apriora AI (Recruitment and Interviews): <https://www.apriora.ai>
 - Velents AI (Recruitment and Interviews): : <https://www.velents.com>
 - Nancy AI (Recruitment and Interviews): : <https://nancy-ai.com>
 - Mariana AI (Medical): <https://marianaai.com>



Fine Tuning and RAG



Foundation Models and Fine-Tuning

What are Foundation Models?

Foundation models are large-scale AI models pre-trained on vast, diverse datasets. These models are designed to be highly generalizable, providing the base for a wide range of downstream tasks. Examples of foundation models include GPT-4, BERT, and CLIP. These models are versatile, capable of handling various data types such as text, images, and audio.

Fine-Tuning

Fine-tuning refers to the process of customizing a pre-trained foundation model for a specific task by training it on additional, domain-specific data. This approach allows organizations to adapt foundation models to their needs without the need for extensive retraining from scratch. For example, a general language model like GPT can be fine-tuned to better handle finance-related text, resulting in models like FinBERT.

Benefits of Fine-Tuning

- **Customization:** Fine-tuning enhances the model's performance for specific applications, making it more relevant for niche use cases.
- **Efficiency:** It requires less computational power and data compared to training a model from scratch.
- **Scalability:** Fine-tuned models can be deployed across various domains, from healthcare to finance and beyond.

Fine-Tuning – Getting started

Create fine-tuning job

POST `https://api.openai.com/v1/fine_tuning/jobs`

Creates a fine-tuning job which begins the process of creating a new model from a given dataset.

Response includes details of the enqueued job including job status and the name of the fine-tuned models once complete.

[Learn more about fine-tuning](#)

Request body

model string **Required**

The name of the model to fine-tune. You can select one of the [supported models](#).

training_file string **Required**

The ID of an uploaded file that contains training data.

See [upload file](#) for how to upload a file.

Your dataset must be formatted as a JSONL file. Additionally, you must upload your file with the purpose `fine-tune`.

The contents of the file should differ depending on if the model uses the [chat](#) or [completions](#) format.

See the [fine-tuning guide](#) for more details.

Default

Epochs

Validation file

W&B Integration

Example request

curl 

```
1 curl https://api.openai.com/v1/fine_tuning/jobs \
2   -H "Content-Type: application/json" \
3   -H "Authorization: Bearer $OPENAI_API_KEY" \
4   -d '{
5     "training_file": "file-BK7bzQj3FfZFXr7DbL6xJwfo",
6     "model": "gpt-4o-mini"
7   }'
```

Response



```
1 {
2   "object": "fine_tuning.job",
3   "id": "ftjob-abc123",
4   "model": "gpt-4o-mini-2024-07-18",
5   "created_at": 1721764800,
6   "fine_tuned_model": null,
7   "organization_id": "org-123",
8   "result_files": [],
9   "status": "queued",
10  "validation_file": null,
```

<https://platform.openai.com/docs/api-reference/fine-tuning/create>

Retrieval-Augmented Generation (RAG)

What is RAG?

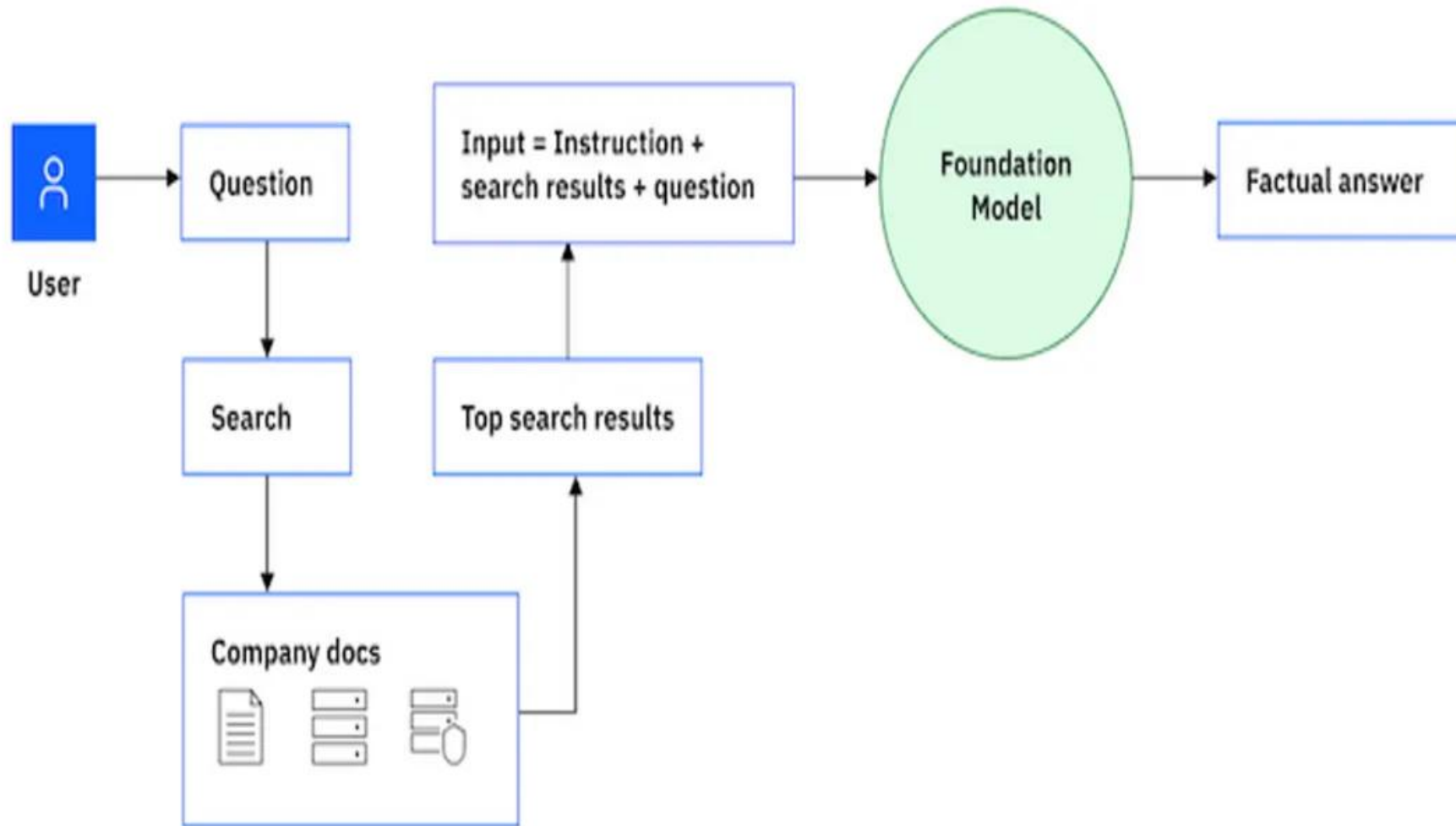
Retrieval-Augmented Generation (RAG) enhances a language model's capabilities by allowing it to access external data sources in real-time. It works by retrieving relevant information from databases or other sources and combining it with the user's query to generate a more contextually accurate and up-to-date response. This is particularly useful when dealing with constantly evolving fields like healthcare or finance, where accessing the latest information is crucial.

RAG VS. Fine tuning

RAG differs from fine-tuning primarily in how each method handles information:

- **RAG** dynamically retrieves external data from databases or APIs during inference, enabling the model to access and incorporate real-time, ensuring the generated responses are up-to-date. It doesn't alter the model's internal knowledge but enhances its answers with current, relevant data.
- In contrast, **fine-tuning** involves retraining the model with a domain-specific dataset to embed specialized knowledge directly into the model itself. This means fine-tuned models provide consistent, specialized outputs but lack the ability to adapt to new or updated information without retraining. This permanently embeds domain-specific knowledge within the model, making it more adept at particular tasks but unable to incorporate new information unless retrained

Retrieval-Augmented Generation (RAG) – Cont.



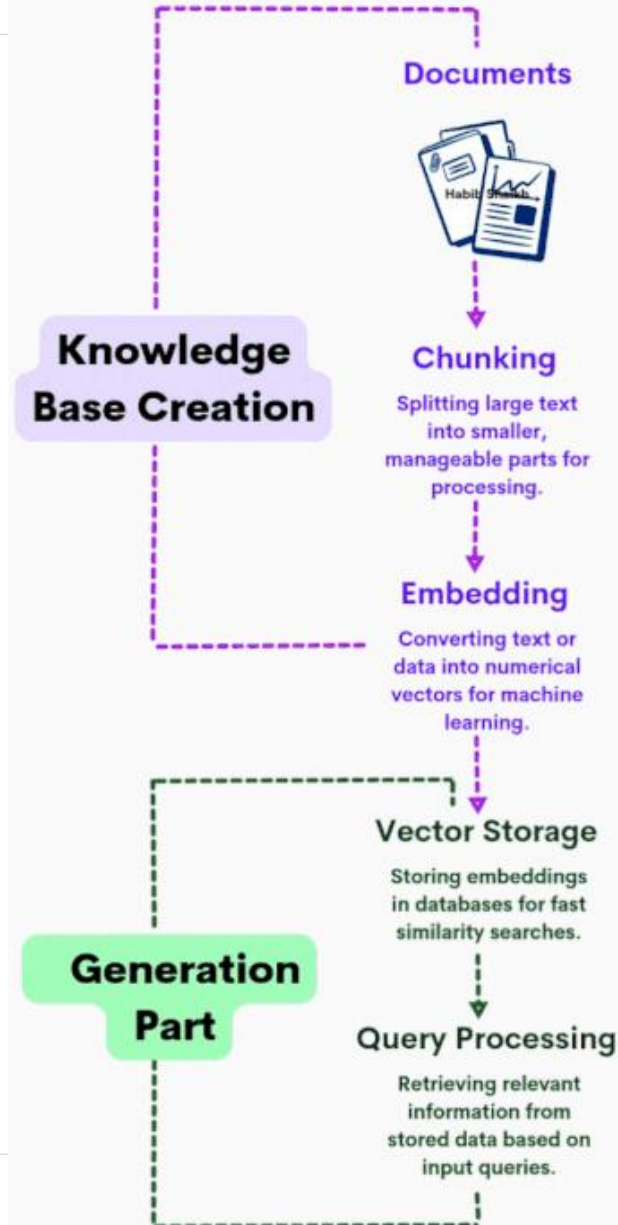
Source: IBM Developer. (n.d.). <https://developer.ibm.com/articles/awb-retrieval-augmented-generation-with-langchain-and-elastic-db/>

RAG data pipeline flow

High-level flow for a data pipeline that supplies grounding data for a RAG application:

1. Documents are either pushed or pulled into a data pipeline.
2. The data pipeline processes each document individually by completing the following steps:
 - a. **Chunk document:** Breaks down the document into semantically relevant parts that ideally have a single idea or concept.
 - b. **Enrich chunks:** Adds metadata fields that the pipeline creates based on the content in the chunks. The data pipeline categorizes the metadata into discrete fields, such as title, summary, and keywords.
 - c. **Embed chunks:** Uses an embedding model to vectorize the chunk and any other metadata fields that are used for vector searches.
 - d. **Persist chunks:** Stores the chunks in the search index.

RAG PIPELINE



Scaling Techniques

Chunking Optimization

- Adjust chunk size for efficiency
- Use overlapping chunks for context

Embedding Optimizations

- Use optimized embedding models
- Reduce dimensionality for speed



Vector Storage Optimization

- Use scalable vector databases.
- Implement indexing for fast retrieval.

Query Optimizations

- Use batch processing for multiple queries
- Distribute queries across machines

Performance Considerations

Retrieval Latency

Time taken to fetch documents.

Embedding Quality

Accuracy of vector representations.

Scalability

Handling increased data and queries.



Memory Usage

Amount of memory required.

Load Balancing

Distributing load to avoid bottlenecks.

RAG – Embeddings

Embeddings are a way to represent words, sentences, or even entire documents as dense vectors in a high-dimensional space.

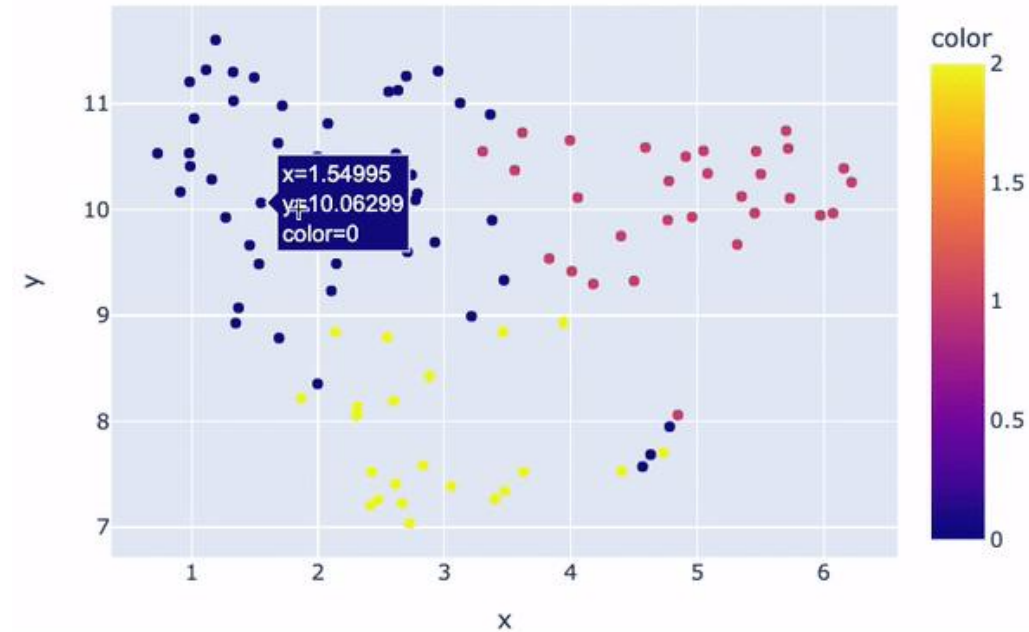
An embedding is a mathematical representation of an object, such as text. When a neural network is being trained, many representations of an object are created. Each representation has connections to other objects in the network. *The purpose of embeddings is to capture the semantic meaning of text, such that words or phrases with similar meanings are located closer to each other in this vector space.*

For example:

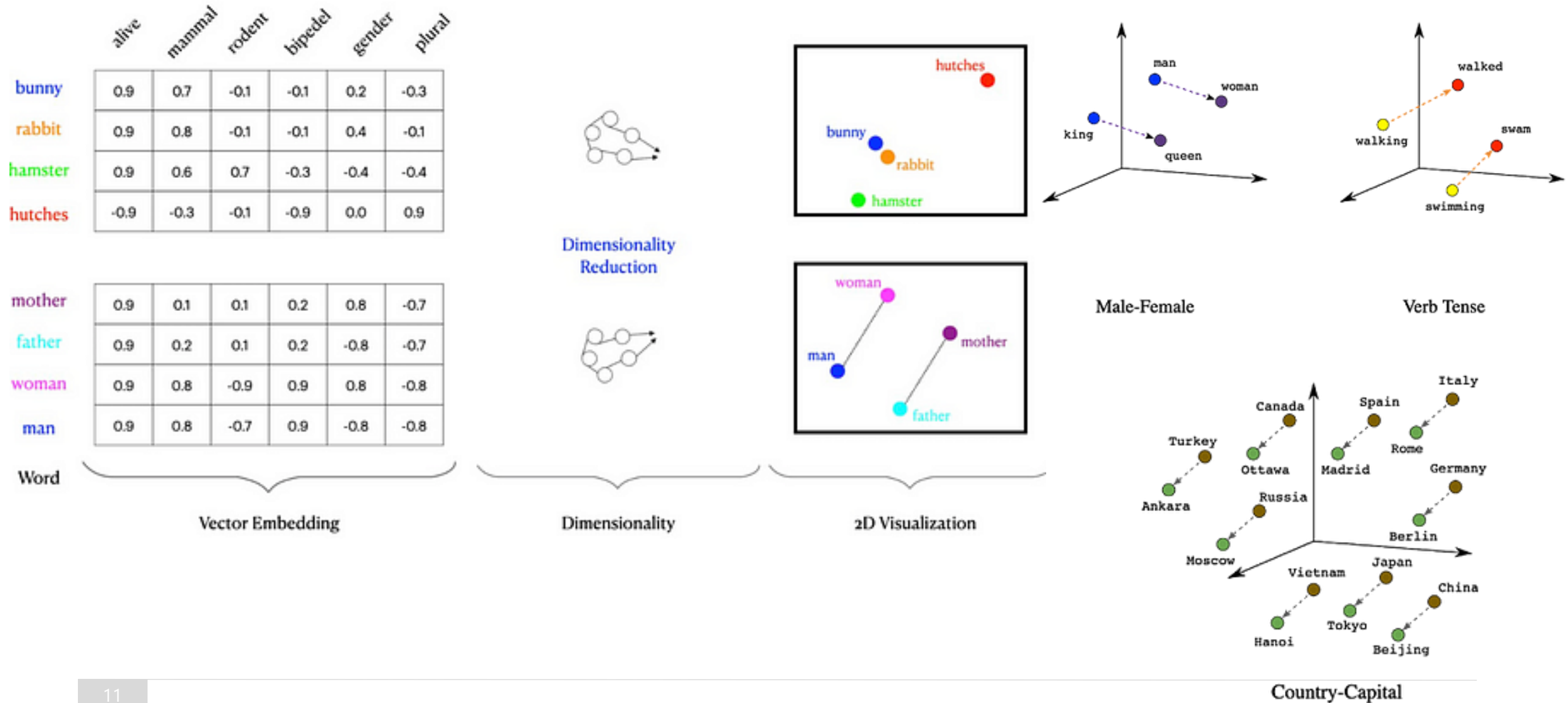
- The words “king” and “queen” might be close to each other in an embedding space because they share similar semantics (both are royalty).

Embedding similarity: the distance between any two items can be calculated mathematically and can be interpreted as a measure of relative similarity between those two items.

The AI model is trained in such a way that these vectors capture the essential features and characteristics of the underlying data.



RAG – Embeddings (Cont.)



RAG – Creating Embeddings

Create embeddings

POST `https://api.openai.com/v1/embeddings`

Creates an embedding vector representing the input text.

Request body

input string or array **Required**

Input text to embed, encoded as a string or array of tokens. To embed multiple inputs in a single request, pass an array of strings or array of token arrays. The input must not exceed the max input tokens for the model (8192 tokens for `text-embedding-ada-002`), cannot be an empty string, and any array must be 2048 dimensions or less. [Example Python code](#) for counting tokens. Some models may also impose a limit on total number of tokens summed across inputs.

✓ Show possible types

model string **Required**

ID of the model to use. You can use the [List models](#) API to see all of your available models, or see our [Model overview](#) for descriptions of them.

dimensions integer **Optional**

The number of dimensions the resulting output embeddings should have. Only supported in `text-embedding-3` and later models.

encoding_format string **Optional** Defaults to float

The format to return the embeddings in. Can be either `float` or `base64`.

Example request

curl ↕

```
1 curl https://api.openai.com/v1/embeddings \
2   -H "Authorization: Bearer $OPENAI_API_KEY" \
3   -H "Content-Type: application/json" \
4   -d '{
5     "input": "The food was delicious and the waiter...",
6     "model": "text-embedding-ada-002",
7     "encoding_format": "float"
8   }'
```

Response

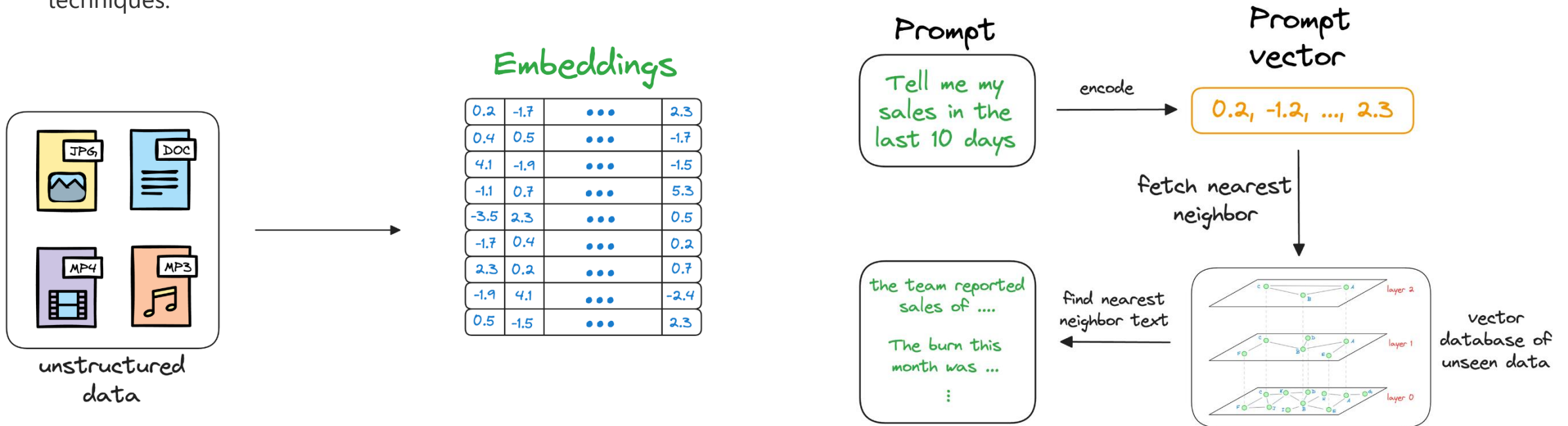
↗

```
1 {
2   "object": "list",
3   "data": [
4     {
5       "object": "embedding",
6       "embedding": [
7         0.0023064255,
8         -0.009327292,
9         .... (1536 floats total for ada-002)
10        -0.0028842222,
11      ],
12       "index": 0
13     }
14   ],
```

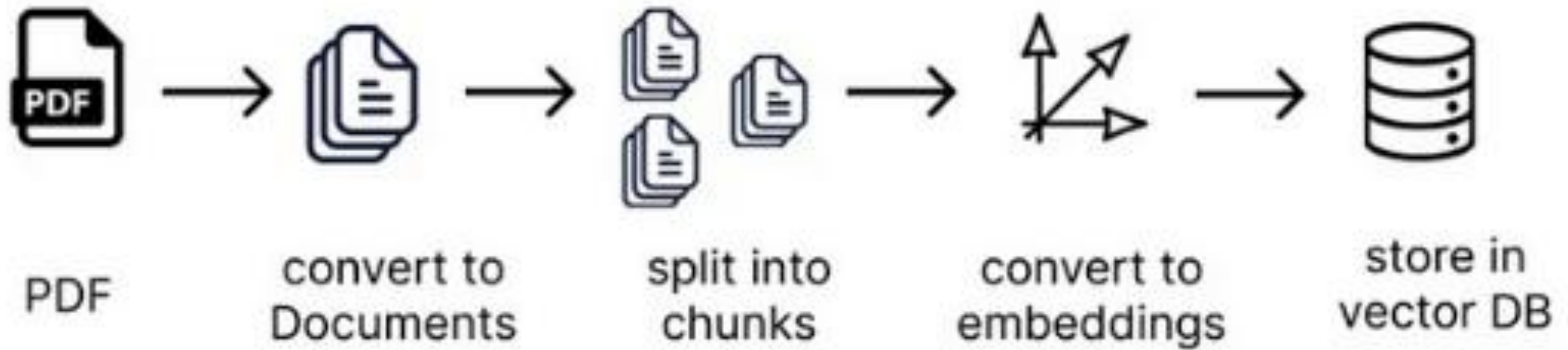
RAG – Vector databases

Vector database stores unstructured data (text, images, audio, video, etc.) in the form of vector embeddings.

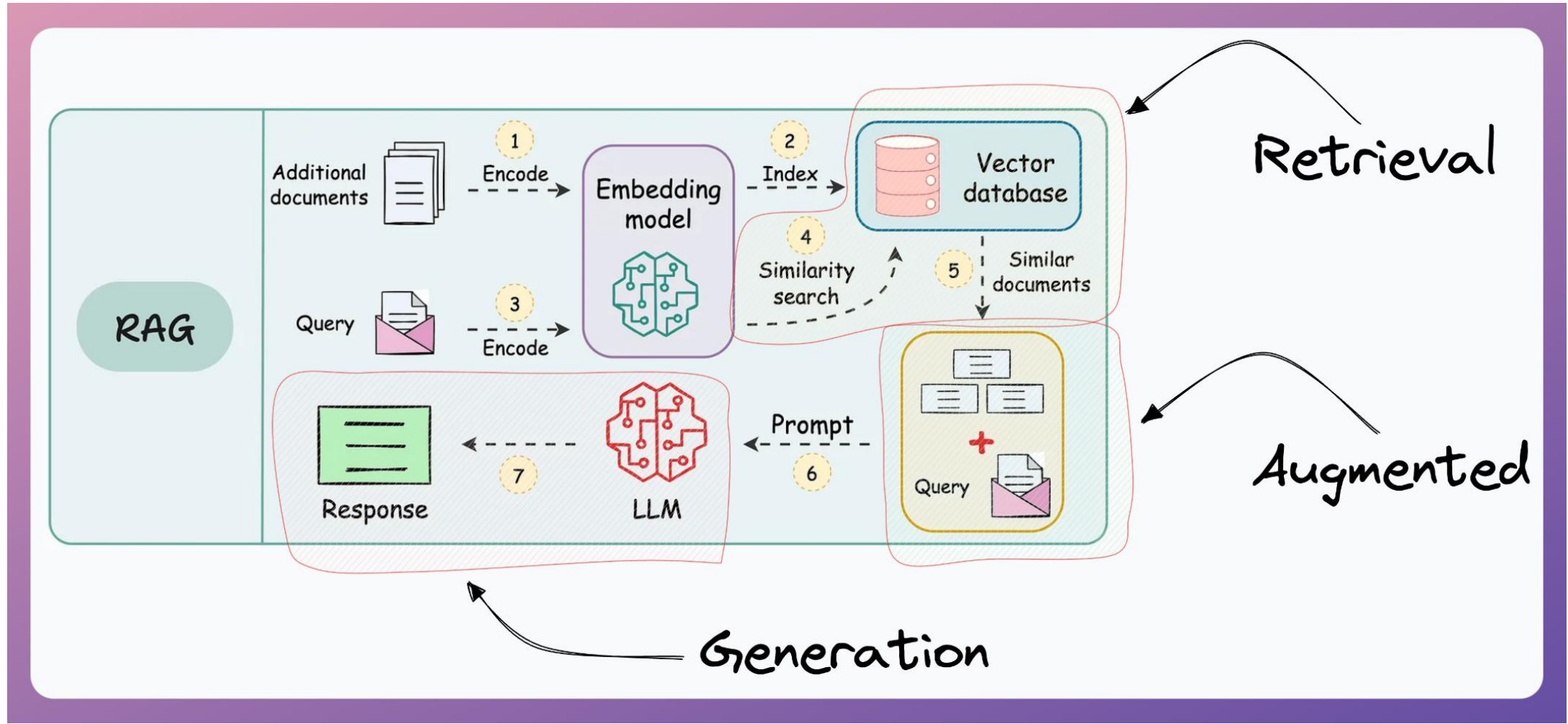
Each data point, whether a word, a document, an image, or any other entity, is transformed into a numerical vector using ML techniques.



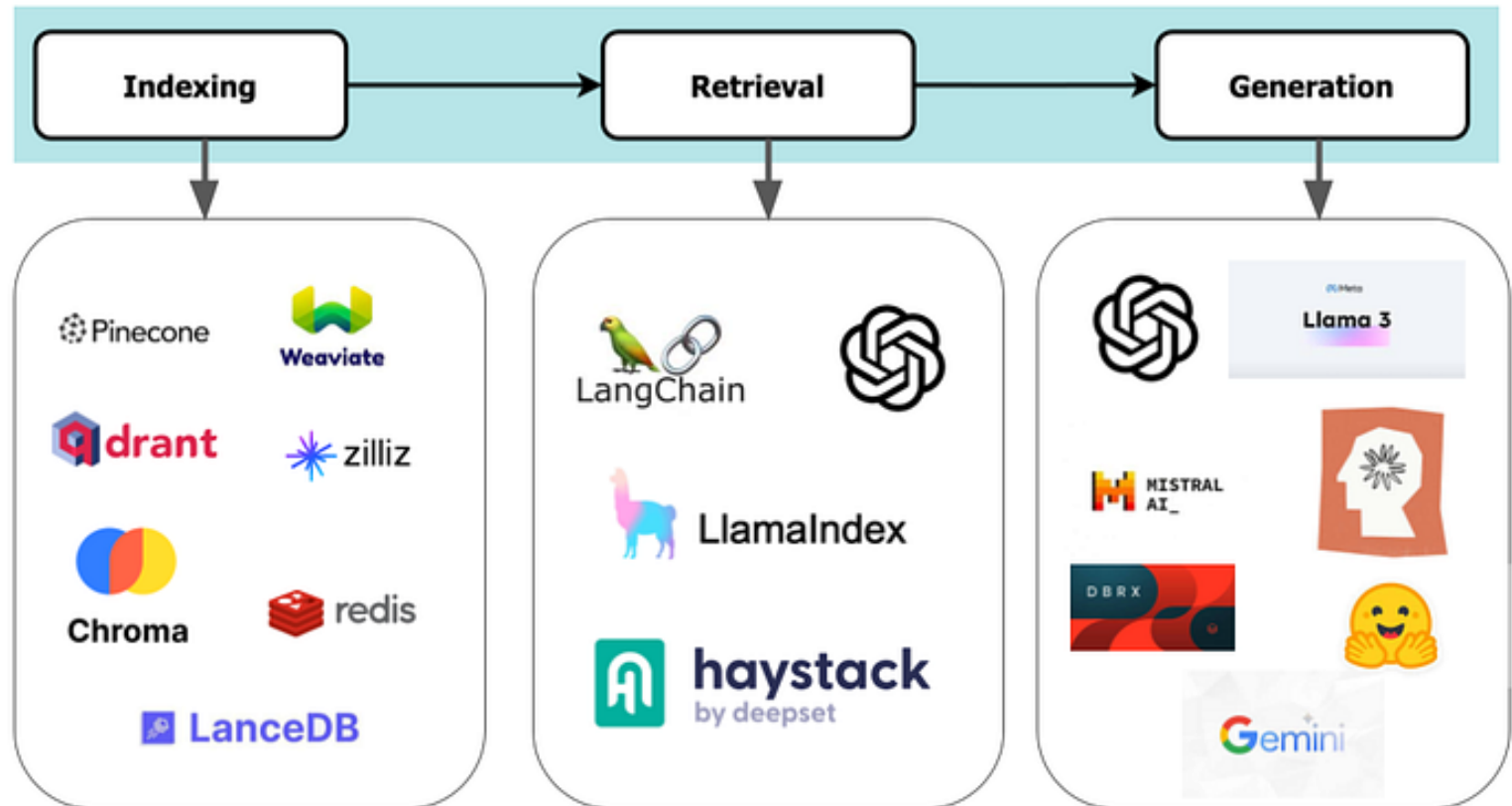
RAG – Putting all together



RAG re-represented



RAG developer's stack



RAG developer' stack

RAG Developer's Stack

LLMs



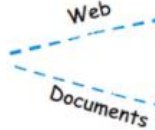
Frameworks



Vector Databases



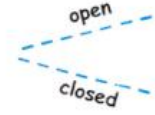
Data Extraction



Open LLMs Access



Text Embeddings

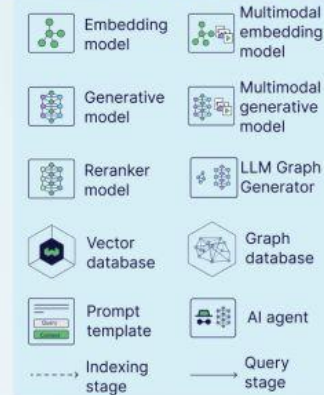


Evaluation

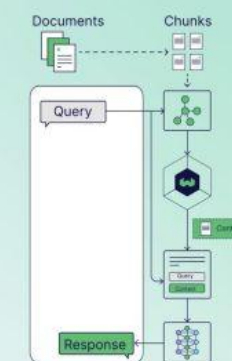


RAG Architectures

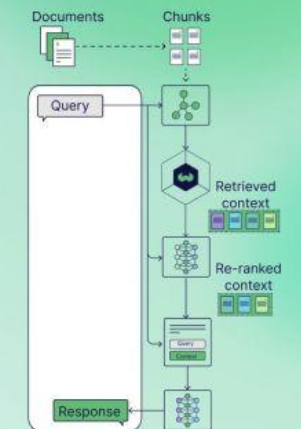
Retrieval-Augmented Generation Architectures



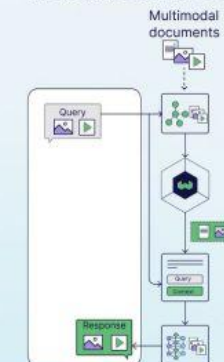
Naive RAG



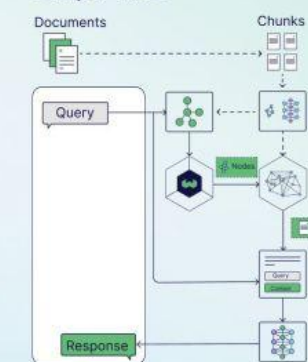
Retrieve-and-rerank



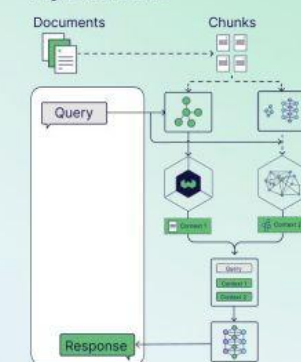
Multimodal RAG



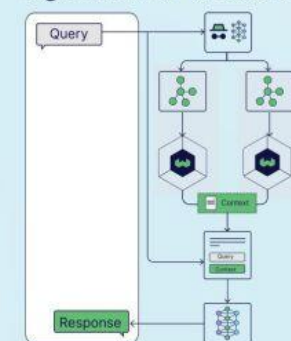
Graph RAG



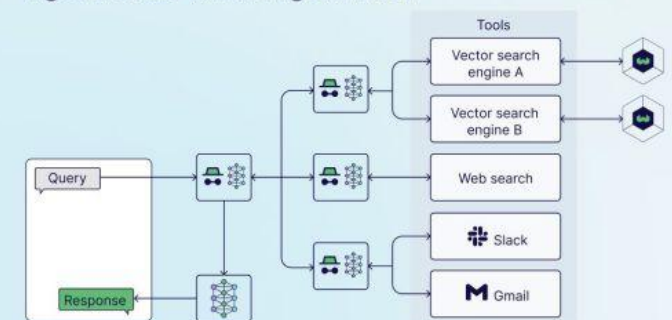
Hybrid RAG



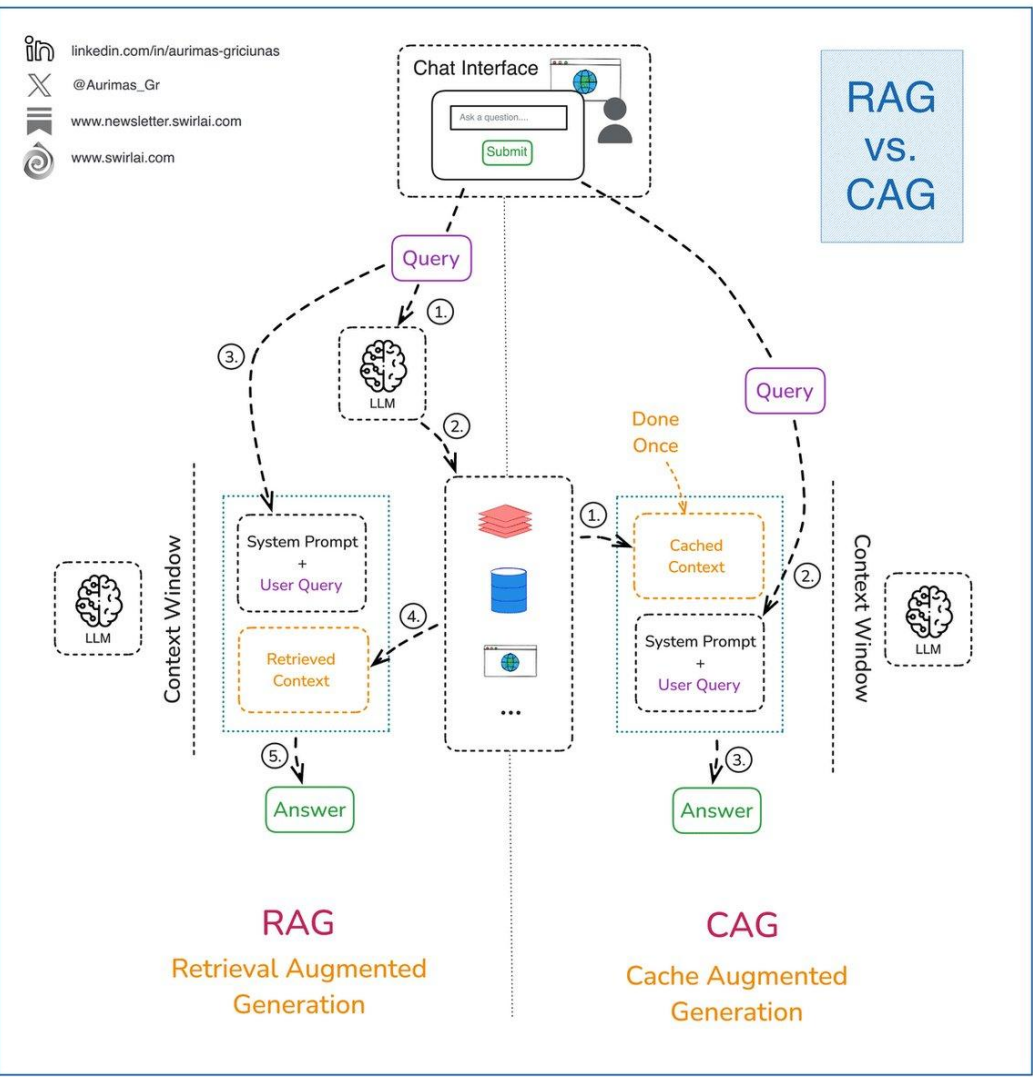
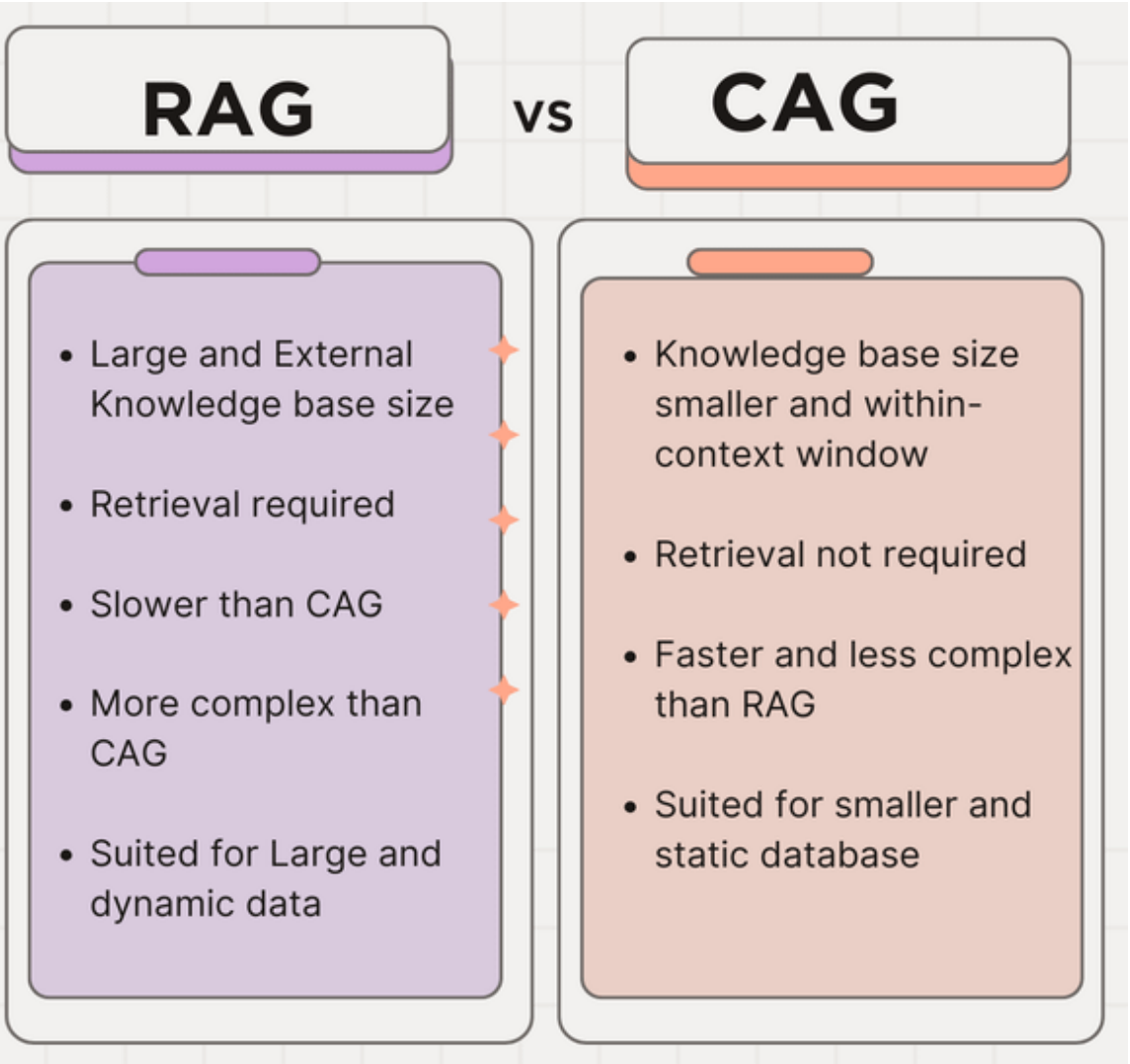
Agentic RAG (Router)



Agentic RAG (Multi-Agent RAG)



Cache-Augmented Generation (CAG)



AI Agents and Agentic framework



Agentic Framework and Multi-Agents

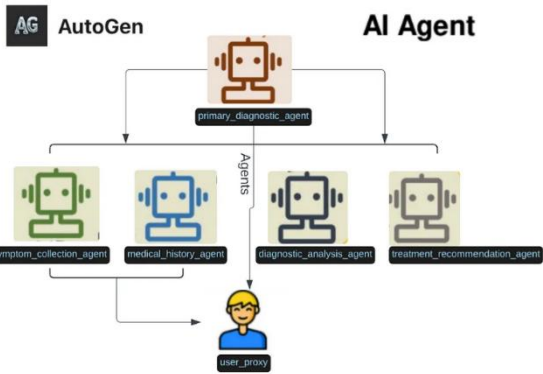
AI agent refers to a system or program that is capable of autonomously performing tasks on behalf of a user or another system by designing its workflow and utilizing available tools.

Non-agentic AI chatbots are ones without available tools, memory and reasoning.

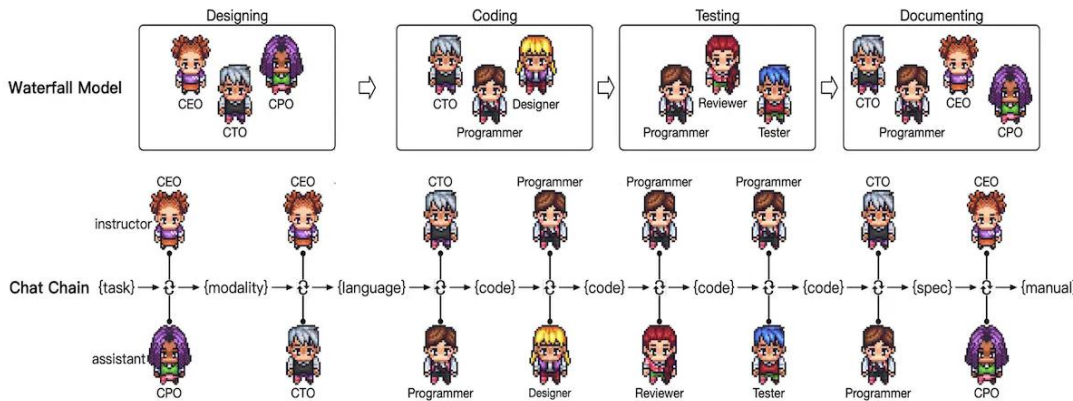
Agentic AI chatbots learn to adapt to user expectations over time, providing a more personalized experience and comprehensive responses.

Agentic Framework allows AI systems to operate through multiple agents, each with specific tasks or roles. This framework supports dynamic, iterative processes that mirror human problem-solving by allowing agents to collaborate and provide feedback to one another. Agentic frameworks can be integrated with Large Language Models (LLMs) and are ideal for automating complex workflows, enhancing decision-making, and improving the quality of AI outputs.

Multi-Agents systems consist of multiple AI agents, each specialized for different functions. These agents can communicate and collaborate, handling subtasks autonomously. For example, in a multi-agent system, one agent could gather information, while another processes it and yet another evaluates the output. This reduces the need for manual intervention and can lead to faster, more accurate solutions in tasks like software development, business operations, and content creation.



Agentic Design Patterns: Multi-Agent Collaboration



Proposed ChatDev architecture. Image adapted from "Communicative Agents for Software Development," Qian et al. (2023).

AI Workflows vs AI Agents

At the core of modern **Agentic Systems** lie two fundamental automation paradigms:

- **Workflows** — Structured, **rule-based automations** where the outcome is predefined.
- **Agents** — Dynamic, **decision-making systems** that adapt to different scenarios based on context.

Key features of an AI agent

- Reasoning:** This core cognitive process involves using logic and available information to draw conclusions, make inferences, and solve problems. AI agents with strong reasoning capabilities can analyze data, identify patterns, and make informed decisions based on evidence and context.
- Acting:** The ability to take action or perform tasks based on decisions, plans, or external input is crucial for AI agents to interact with their environment and achieve goals. This can include physical actions in the case of embodied AI, or digital actions like sending messages, updating data, or triggering other processes.
- Observing:** Gathering information about the environment or situation through perception or sensing is essential for AI agents to understand their context and make informed decisions. This can involve various forms of perception, such as computer vision, natural language processing, or sensor data analysis.
- Planning:** Developing a strategic plan to achieve goals is a key aspect of intelligent behavior. AI agents with planning capabilities can identify the necessary steps, evaluate potential actions, and choose the best course of action based on available information and desired outcomes. This often involves anticipating future states and considering potential obstacles.
- Collaborating:** Working effectively with others, whether humans or other AI agents, to achieve a common goal is increasingly important in complex and dynamic environments. Collaboration requires communication, coordination, and the ability to understand and respect the perspectives of others.
- Self-refining:** The capacity for self-improvement and adaptation is a hallmark of advanced AI systems. AI agents with self-refining capabilities can learn from experience, adjust their behavior based on feedback, and continuously enhance their performance and capabilities over time. This can involve machine learning techniques, optimization algorithms, or other forms of self-modification.

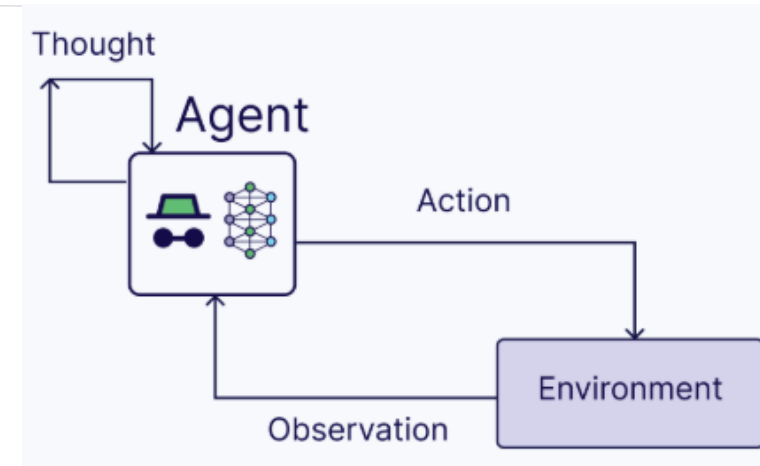
Agentic framework – Reasoning paradigms

One popular framework is the **ReAct framework**. A ReAct agent can handle sequential multi-part queries while maintaining state (in memory) by combining routing, query planning, and tool use into a single entity.

ReAct = Reason + Act (with LLMs)

The process involves the following steps:

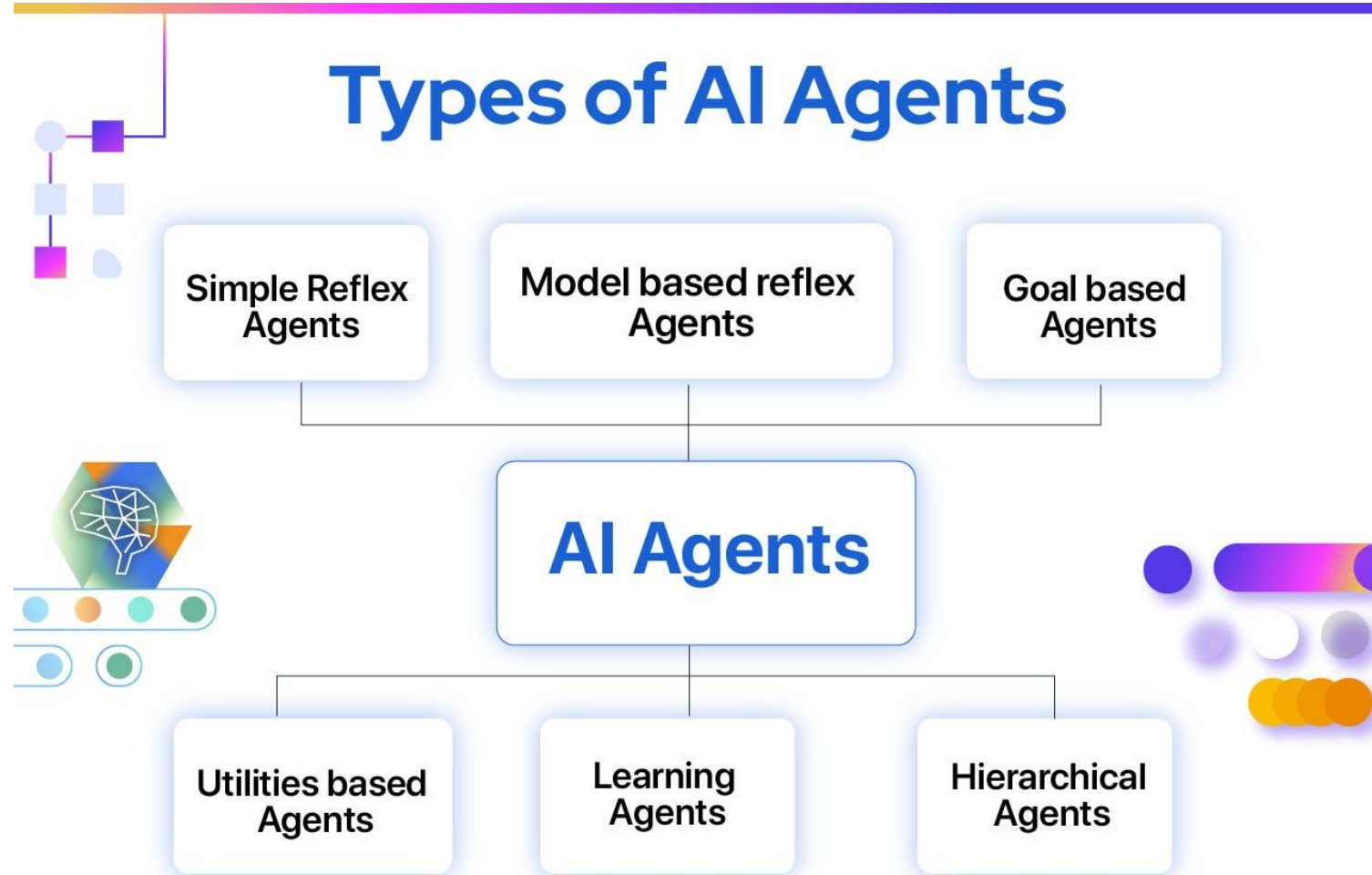
1. Thought: Upon receiving the user query, the agent reasons about the next action to take
2. Action: the agent decides on an action and executes it (e.g., tool use)
3. Observation: the agent observes the feedback from the action
4. This process iterates until the agent completes the task and responds to the user.



Introducing ReWOO: A Smarter Approach

The paper *ReWOO: Decoupling Reasoning from Observations for Efficient Augmented Language Models* presents a groundbreaking shift in ALM design. ReWOO (Reasoning Without Observation) restructures how LLMs interact with external tools by decoupling the reasoning process from tool execution. Instead of pausing after each tool call, ReWOO enables the LLM to generate a complete reasoning plan before executing any tool interactions.

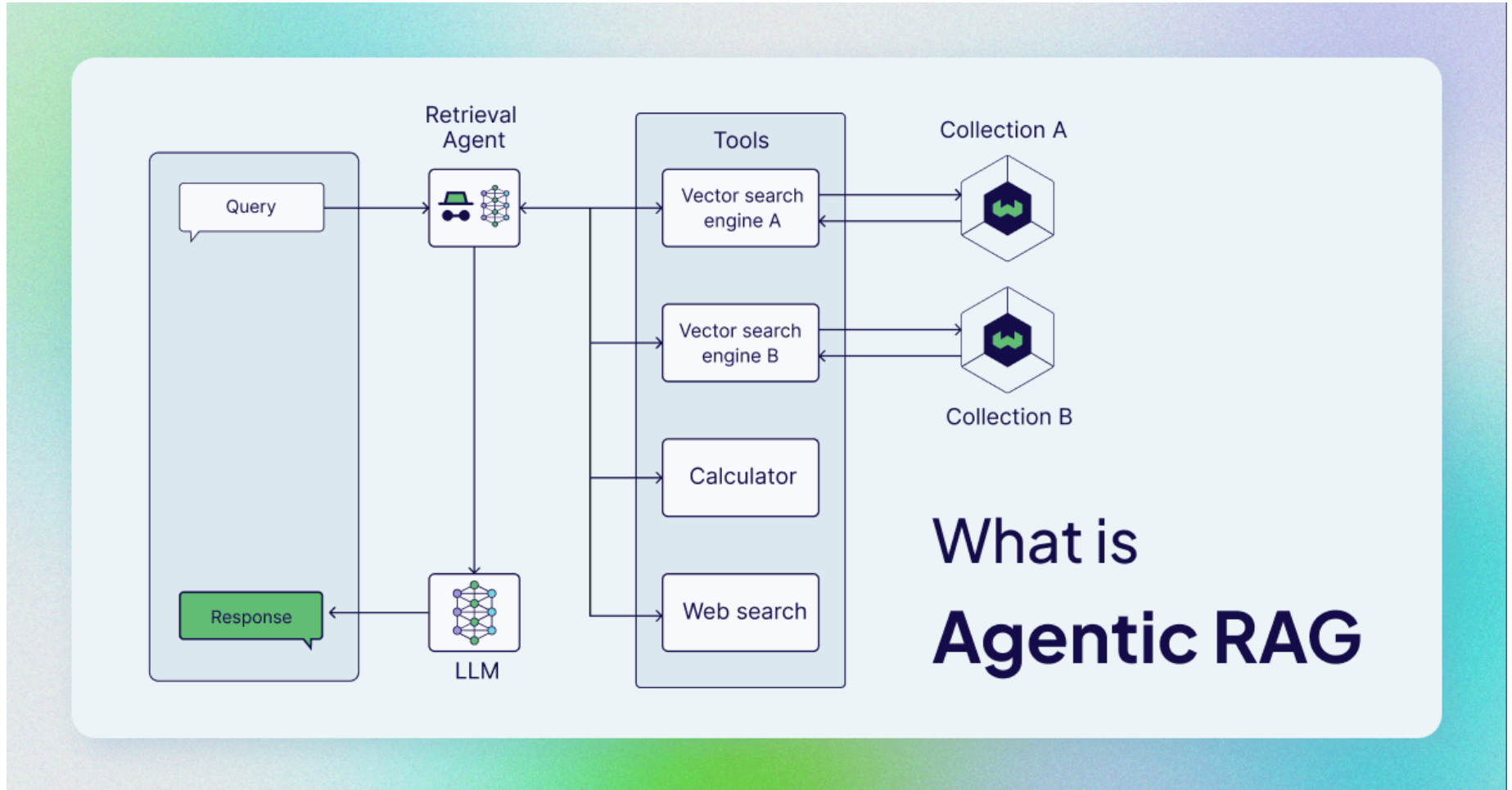
AI Agents Types



<https://www.ibm.com/think/topics/ai-agents#Types+of+AI+agents>
<https://www.digitalocean.com/resources/articles/types-of-ai-agents>

Agentic RAG

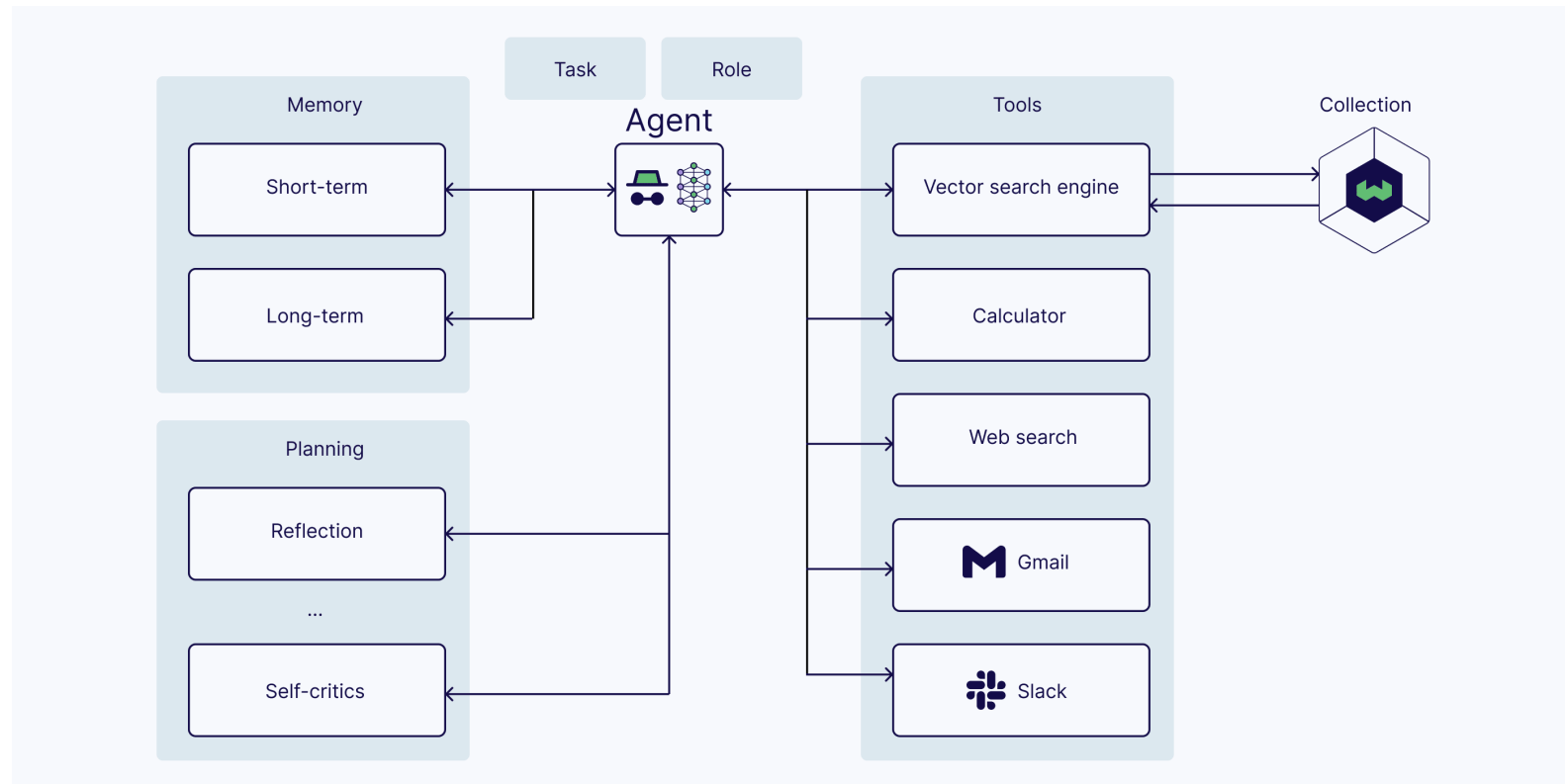
Agentic RAG describes an AI agent-based implementation of RAG. Specifically, it incorporates AI agents into the RAG pipeline to orchestrate its components and perform additional actions beyond simple information retrieval and generation to overcome the limitations of the non-agentic pipeline.



Core components of an AI agent

The core components of an AI agent are:

- LLM (with a role and a task)
- Memory (short-term and long-term)
- Planning (e.g., reflection, self-critics, query routing, etc.)
- Tools (e.g., calculator, web search, etc.)



Agentic frameworks

2025 AI Agent Tech Stack



<https://www.ibm.com/think/insights/top-ai-agent-frameworks>



Armand Ruiz  • Following
VP of Product - AI Platform @IBM
2d • 

Building AI agents is 5% AI and 100% software engineering.
You need:

1/ Smooth Integration: Seamlessly connect with existing systems.

2/ Coordinated Integration: Facilitate collaboration between agents and humans.

3/ Reliability: Deliver AI systems that perform consistently in production.

4/ High Scalability: Design for real-world loads.

5/ Constant Monitoring: Keep systems under continuous watch.

Your models & AI agents are only as good as the engineering that deploys them.



Sri Laasya Nutheti   @n_sri_laasya · Feb 14

Building AI agents is 5% AI and 100% software engineering

 98

 145

 1.6K

 133K



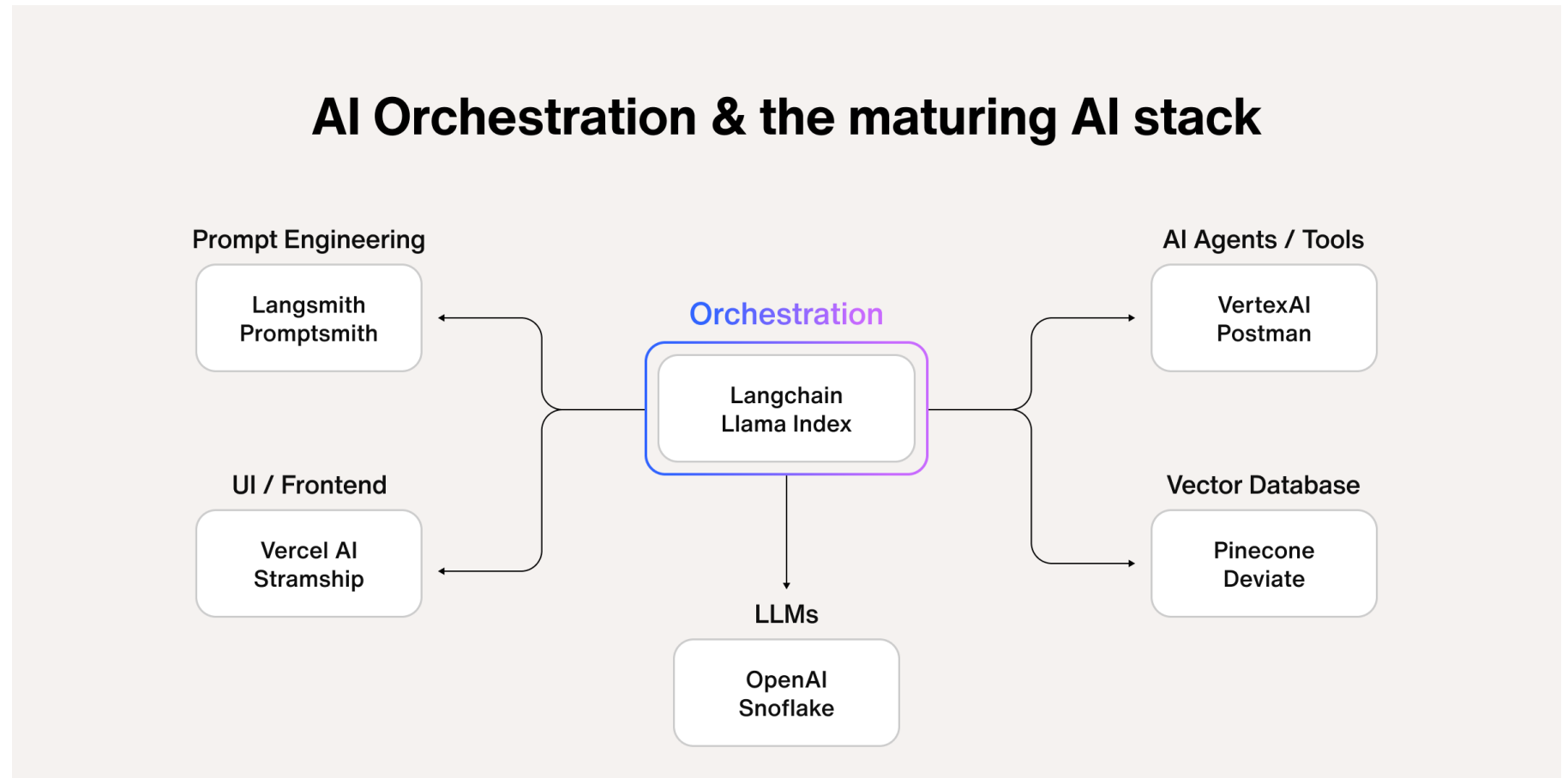
AI orchestration frameworks



AI orchestration

AI orchestration is the process of coordinating different AI tools and systems, so they work together effectively.

AI orchestration, if done well, increases efficiency and effectiveness because it streamlines processes and ensures the AI you're using communicate, share data, and function as one system.



LangChain

[LangChain](#) is an open-source framework for developing applications powered by language models. It provides a modular and composable API for building chains of operations that can be used to create a wide variety of applications, such as chatbots, question-answering systems, and document summarization tools.

LangChain is built on top of the popular Hugging Face Transformers library, which provides access to a wide range of pre-trained language models. This allows developers to focus on building their applications without having to worry about the underlying infrastructure.

LangChain is still under development, but it has already been used to create a number of impressive applications, including:

- A chatbot that can answer questions about the world
- A system that can summarize long documents
- A tool that can generate creative text formats, like poems, code, scripts, musical pieces, email, letters, etc.

LangChain is a powerful tool that can be used to create a wide variety of applications. It is easy to learn and use, and it is backed by a large and active community.

Here are some of the key features of LangChain:

- **Modular and composable API:** LangChain provides a modular and composable API that makes it easy to build chains of operations. This allows developers to create complex applications without having to write a lot of code.
- **Access to pre-trained language models:** LangChain is built on top of the popular Hugging Face Transformers library, which provides access to a wide range of pre-trained language models. This allows developers to get started quickly and easily.
- **Active community:** LangChain has a large and active community of developers who are constantly contributing new features and improvements. This ensures that LangChain is always up-to-date and that developers have access to the latest resources.

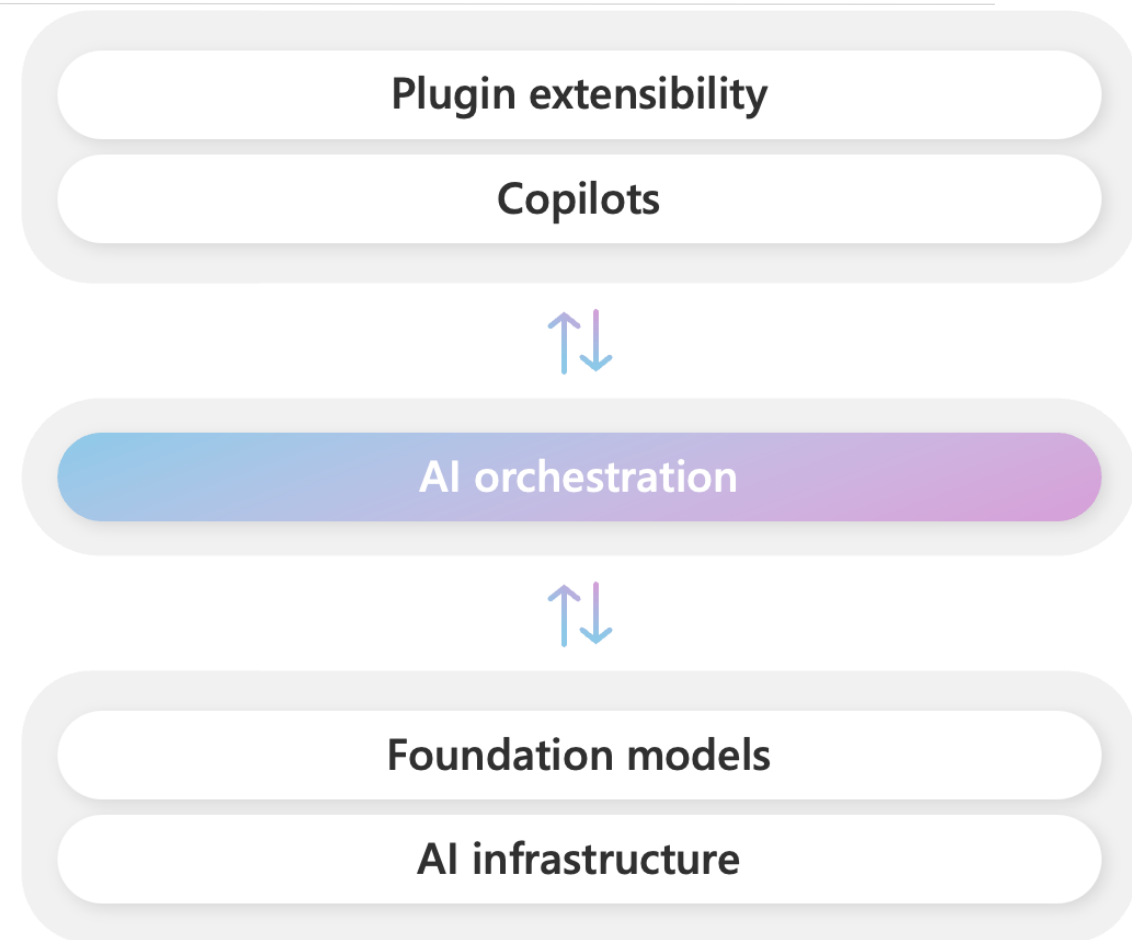
If you are interested in building applications powered by language models, then LangChain is a great place to start. It is a powerful tool that is easy to learn and use.

Microsoft Semantic Kernel

Semantic Kernel is an open-source SDK that lets you easily build agents that can call your existing code. As a highly extensible SDK, you can use Semantic Kernel with models from OpenAI, Azure OpenAI, Hugging Face, and more! By combining your existing C#, Python, and Java code with these models, you can build agents that answer questions and automate processes.

It integrates Large Language Models (LLMs) like OpenAI, Azure OpenAI, and Hugging Face with conventional programming languages like C#, Python, and Java. Semantic Kernel achieves this by allowing you to define plugins that can be chained together in just a few lines of code.

What makes Semantic Kernel special, however, is its ability to automatically orchestrate plugins with AI. With Semantic Kernel planners, you can ask an LLM to generate a plan that achieves a user's unique goal. Afterwards, Semantic Kernel will execute the plan for the user.



OpenAI APIs – Advanced tools



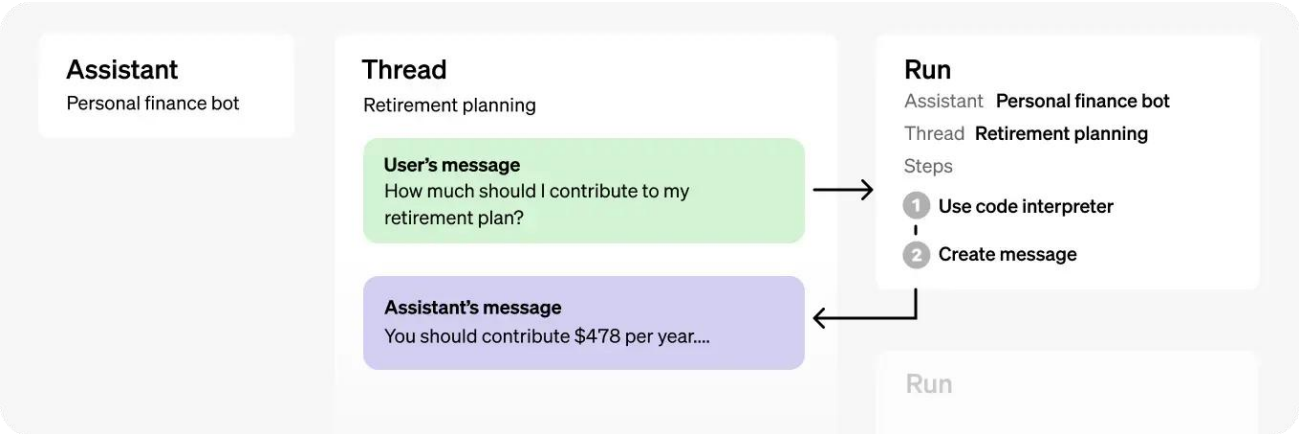
OpenAI APIs – Assistants API Overview

Assistants API Overview

The Assistants API allows you to build AI assistants within your own applications. An Assistant has instructions and can leverage models, tools, and files to respond to user queries. The Assistants API currently supports three types of [tools](#): Code Interpreter, File Search, and Function calling.

How Assistants work

- 1.Assistants can call OpenAI’s [models](#) with specific instructions to tune their personality and capabilities.
- 2.Assistants can access **multiple tools in parallel**. These can be both OpenAI-hosted tools — like [code interpreter](#) and [file search](#) — or tools you build / host (via [function calling](#)).
- 3.Assistants can access **persistent Threads**. Threads simplify AI application development by storing message history and truncating it when the conversation gets too long for the model’s context length. You create a Thread once, and simply append Messages to it as your users reply.
- 4.Assistants can access files in several formats — either as part of their creation or as part of Threads between Assistants and users. When using tools, Assistants can also create files (e.g., images, spreadsheets, etc) and cite files they reference in the Messages they create.



OBJECT	WHAT IT REPRESENTS
Assistant	Purpose-built AI that uses OpenAI’s models and calls tools
Thread	A conversation session between an Assistant and a user. Threads store Messages and automatically handle truncation to fit content into a model’s context.
Message	A message created by an Assistant or a user. Messages can include text, images, and other files. Messages stored as a list on the Thread.
Run	An invocation of an Assistant on a Thread. The Assistant uses its configuration and the Thread’s Messages to perform tasks by calling models and tools. As part of a Run, the Assistant appends Messages to the Thread.
Run Step	A detailed list of steps the Assistant took as part of a Run. An Assistant can call tools or create Messages during its run. Examining Run Steps allows you to introspect how the Assistant is getting to its final results.

Assistant API – Code interpreter

Code Interpreter

What it is: The Code Interpreter allows an OpenAI Assistant to write and run Python code within a secure environment.

Use Case: Suppose you need to analyze a CSV file containing sales data. The Code Interpreter can read the file, perform calculations (like summing up total sales), and even create a graph to visualize the data. If the code doesn't work initially, it can try different approaches until it gets it right.

Example: You ask, "Analyze this sales data and show me the total revenue." The Assistant can load your CSV file, write the necessary Python code, and present you with the result.

Assistant API – File search

File Search

What it is: File Search lets the Assistant access and retrieve information from documents you upload, beyond its pre-existing knowledge.

Use Case: Imagine you upload a manual for a product your company makes. The Assistant can search through that manual to answer specific questions, like "What's the warranty period?" by finding the relevant section in the document.

Example: You upload a PDF of a product guide and ask, "What are the installation steps?" The Assistant can locate and provide the steps directly from your document.

Assistant API – Function calling

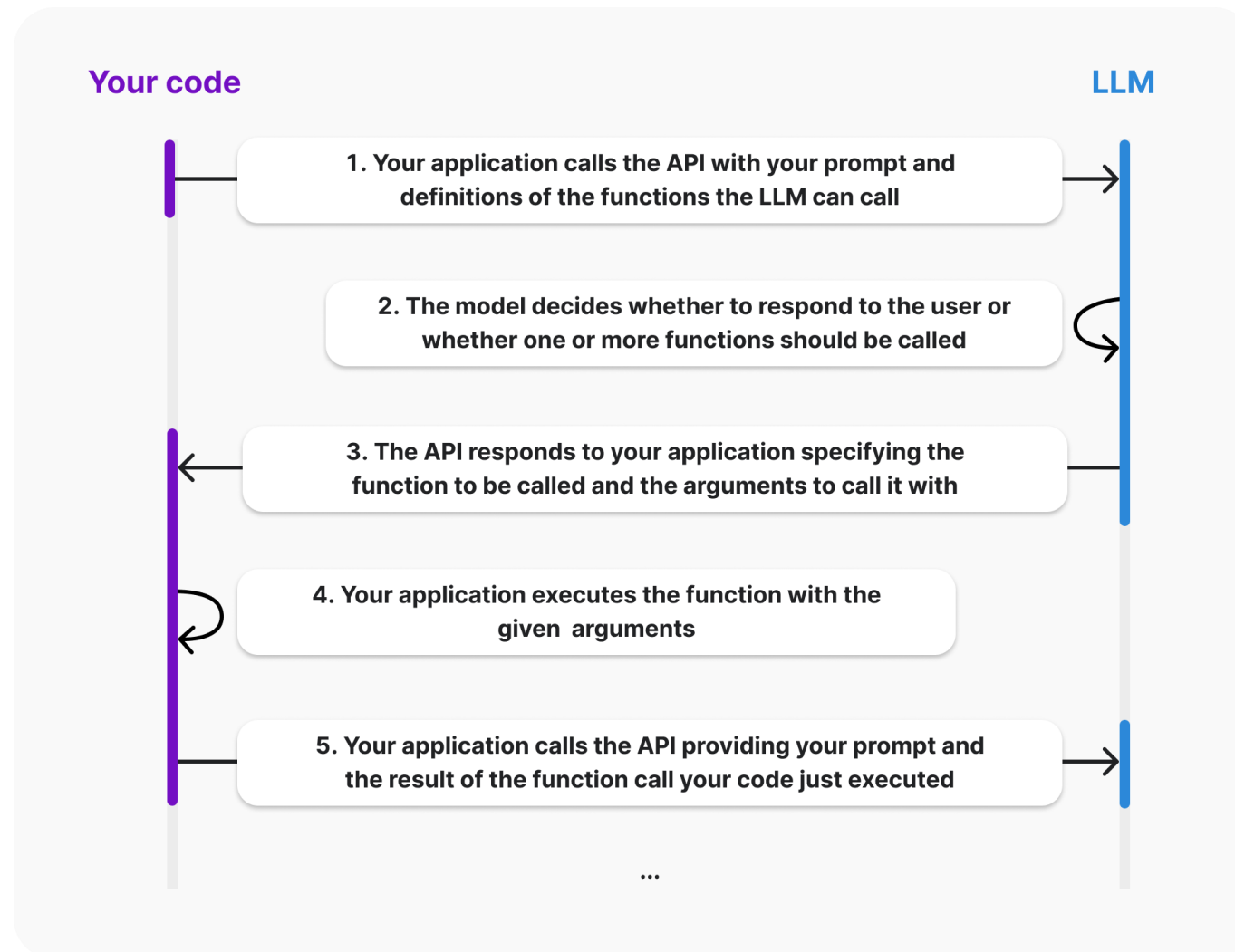
Function Calling

What it is: Function Calling allows you to define specific functions that the Assistant can call during a conversation. The Assistant knows when to use these functions and what arguments to pass to them.

Use Case: If you have a weather forecasting function, the Assistant can automatically call this function when you ask about the weather, providing you with accurate, up-to-date information.

Example: You might ask, "What's the weather like today?" The Assistant will recognize this as a request for weather data and use the appropriate function to fetch and return the current weather for your location.

Assistant API – Function calling (Cont.)



Structured output

What it is: Structured Outputs ensure that the responses generated by the OpenAI Assistant are always formatted according to a specified JSON Schema.

Use Case: Suppose you're building an app that needs to process user input, such as booking information. You define a JSON Schema that specifies exactly how the booking details should be structured (e.g., with fields like: name, date, time). The Assistant will always provide the output in this structured format, reducing the risk of errors.

Example: If you ask the Assistant to "Create a booking for John on September 10th at 3 PM," it will return the details in the exact JSON format you specified, like this:

```
{ "name": "John", "date": "2024-09-10", "time": "15:00"}
```

This ensures the data is always ready to be processed by your application without additional checks.

```
# Your OpenAI API key
openai.api_key = "your-api-key"

# Define your JSON Schema
json_schema = {
    "type": "object",
    "properties": {
        "name": {"type": "string"},
        "date": {"type": "string", "format": "date"},
        "time": {"type": "string", "format": "time"}
    },
    "required": ["name", "date", "time"]
}

# Make a request with the schema
response = openai.ChatCompletion.create(
    model="gpt-4",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Create a booking for John on September 10th at 3 PM."}
    ],
```


Open AI Agent SDK

Overview







Building agents involves assembling components across several domains—such as **models, tools, knowledge and memory, audio and speech, guardrails, and orchestration**—and OpenAI provides composable primitives for each.

DOMAIN	DESCRIPTION	OPENAI PRIMITIVES
Models	Core intelligence capable of reasoning, making decisions, and processing different modalities.	o1, o3-mini, GPT-4.5, GPT-4o, GPT-4o-mini
Tools	Interface to the world, interact with environment, function calling, built-in tools, etc.	Function calling, Web search, File search, Computer use
Knowledge and memory	Augment agents with external and persistent knowledge.	Vector stores, File search, Embeddings
Audio and speech	Create agents that can understand audio and respond back in natural language.	Audio generation, realtime, Audio agents
Guardrails	Prevent irrelevant, harmful, or undesirable behavior.	Moderation, Instruction hierarchy
Orchestration	Develop, deploy, monitor, and improve agents.	Agents SDK, Tracing, Evaluations, Fine-tuning

<https://platform.openai.com/docs/guides/agents>
<https://openai.github.io/openai-agents-python/>

OpenAI Cookbook

New APIs

<div>Introduction to Structured Outputs</div> <div> Katia Gil Guzman</div> <div>Aug 6, 2024</div> <div>COMPLETIONSFUNCTIONS</div>	<div>Introduction to GPT-4o and GPT-4o mini</div> <div> Juston Forte</div> <div>Jul 18, 2024</div> <div>COMPLETIONSVISIONWHISPER</div>	<div>Batch processing with the Batch API</div> <div> Katia Gil Guzman</div> <div>Apr 24, 2024</div> <div>BATCHCOMPLETIONS</div>
<div>Assistants API Overview (Python SDK)</div> <div> Ilan Bigio</div> <div>Nov 10, 2023</div> <div>ASSISTANTSFUNCTIONS</div>	<div>Processing and narrating a video with GPT's visual capabilities and the TTS API</div> <div> Kai Chen</div> <div>Nov 6, 2023</div> <div>COMPLETIONSPEECHVISION</div>	<div>Using GPT4 Vision with Function Calling</div> <div> Shyamal Anadkat</div> <div>Apr 9, 2024</div> <div>CHATVISION</div>

Model Context Protocol (MCP)

[Introducing the Model Context Protocol \ Anthropic](#)

What's next?

Schedule for 1st run (Development tracks)



- **Part 2: Developing Gen AI and Agentic software applications**
 - **Day 3:** Thu, 20-Mar 2025 (Gen-AI APIs) | [Meeting Link](#)
 - **Day 4:** Sat, 22-Mar 2025 (RAG and Agentic software development) | [Meeting Link](#)
 - **Online courses (RAG, AI Agents) + Gen-AI projects Ideas preparation (23 -30 March)**

Option 1: Udemy (Subscription):

- [RAG, AI Agents and Generative AI with Python and OpenAI 2025](#)
- [Generative AI Architectures with LLM, Prompt, RAG, Vector DB](#)
- [2025 Master Langchain and Ollama - Chatbot, RAG and Agents](#)
- [Mastering Ollama: Build Private Local LLM Apps with Python](#)

Option 2: Free courses

- [LangChain for LLM Application Development](#)
- [LangChain: Chat with Your Data](#)
- [Multi AI Agent Systems with crewAI](#)
- [Open-Source Models with Hugging Face](#)
- [Run LLM Models Locally using Ollama](#)

- **Day 5:** Sat, 5-Apr 2025 (Recapping, Discussing **ideas**) | [Meeting Link](#)
- **Part 3: Final Project implementation: 1 – 30 April**



More resources

Different learning paths

Resources – Udemy (Subscription)



- **Gen AI and OpenAI APIs:**
 - Generative AI For Beginners with ChatGPT and OpenAI API: <https://banquemisr25.udemy.com/course/generative-ai-with-chatgpt-and-openai-api>
 - Complete OpenAI API Course: <https://banquemisr25.udemy.com/course/complete-openai-api-course-connect-to-chatgpt-api-more/>
 - Generative AI using OpenAI API for Beginners: <https://banquemisr25.udemy.com/course/open-ai-api-for-beginners-using-python/>
- **APIs, RAG and AI Agents:**
 - RAG, AI Agents and Generative AI with Python and OpenAI 2025 (Beginner: Including Python intro): <https://banquemisr25.udemy.com/course/generative-ai-rag/>
 - Generative AI Architectures with LLM, Prompt, RAG, Vector DB (Beginner: Hugging-face, Ollama, RAG, Vector DB) <https://www.udemy.com/course/generative-ai-architectures-with-llm-prompt-rag-vector-db/?couponCode=LETSLEARNNOW>
 - Basic to Advanced: Retrieval-Augmented Generation (RAG): <https://banquemisr25.udemy.com/course/basic-to-advanced-retrieval-augmented-generation-rag-course/>
 - Build Autonomous AI Agents From Scratch With Python (Beginner: simple Agent): <https://banquemisr25.udemy.com/course/build-autonomous-ai-agents-from-scratch-with-python/>
 - 2025 Master Langchain and Ollama - Chatbot, RAG and Agents (Intermediate): <https://banquemisr25.udemy.com/course/ollama-and-langchain>
- **Ollama & Local LLM:**
 - Mastering Ollama: Build Private Local LLM Apps with Python: <https://banquemisr25.udemy.com/course/master-ollama-python>
 - Zero to Hero in Ollama: Create Local LLM Applications: <https://banquemisr25.udemy.com/course/ollama-starttech>
 - Ollama Zero to Hero: Build Chat, Vision Games & AI Agents: <https://banquemisr25.udemy.com/course/ollama-docker-api-library-full-course/>
- **AI Agents (Intermediate):**
 - AI Agentic Design Patterns with Ollama & OpenAI Guide: <https://banquemisr25.udemy.com/course/ai-agentic-design-patterns/>
 - AI-Agents: Automation & Business with LangChain & LLM Apps (Intermediate): <https://banquemisr25.udemy.com/course/ai-agents-automation-business-with-langchain-llm-apps/>
- **GitHub Copilot:**
 - AI For Developers With GitHub Copilot, Cursor AI & ChatGPT: <https://banquemisr25.udemy.com/course/ai-for-developers-with-github-copilot-cursor-ai-chatgpt/>
 - GitHub Copilot Beginner to Pro - AI for Coding & Development: <https://banquemisr25.udemy.com/course/github-copilot/>
- **Other advanced topics (Optional)**
 - Generative AI application design and development (Intermediate: Hugging-face, LangChain, RAG): <https://www.udemy.com/course/generative-ai-app-dev/?couponCode=LETSLEARNNOW>
 - LLM Engineering: Master AI, Large Language Models & Agents (Intermediate: Generative AI, RAG, LoRA and AI Agents) <https://www.udemy.com/course/llm-engineering-master-ai-and-large-language-models/>
 - LangChain in Action: Develop LLM-Powered Applications: <https://banquemisr25.udemy.com/course/langchain-in-action-develop-llm-powered-applications/>
 - AI Agents: Building Teams of LLM Agents that Work For You (AutoGen, ChatGPT API, Streamlit, Google Cloud): <https://banquemisr25.udemy.com/course/ai-agents-building-teams-of-llm-agents-that-work-for-you/>
 - 2025 Bootcamp: Generative AI, LLM Apps, AI Agents, Cursor AI: <https://banquemisr25.udemy.com/course/bootcamp-generative-artificial-intelligence-and-llm-app-development>
- **Python intro (Optional)**
 - Python Basics Course: <https://banquemisr25.udemy.com/course/python-entry-level-programmer-certification-pcep/>

Resources – MaharaTech talks (Free)



- **Staff development – Recorded sessions:**

- <https://drive.google.com/drive/folders/1A23GwazLX9L76XK0WZIFvD2szreR-yJA>

- **Mahara-Tech talk (Prepared from Staff Dev sessions):**

- Exploring Generative AI: Tools, Use Cases and Future Trends: <https://maharatech.gov.eg/course/view.php?id=2294>
- Gen-AI: Tools for the Modern Developer: <https://maharatech.gov.eg/course/view.php?id=2299>



Resources (DeepLearning.ai – Free)



- **Python intro (Optional):**
 - AI Python for Beginners: <https://www.deeplearning.ai/short-courses/ai-python-for-beginners/>
- **Prompt engineering for Devs, OpenAI APIs**
 - ChatGPT Prompt Engineering for Developers: <https://www.deeplearning.ai/short-courses/chatgpt-prompt-engineering-for-developers/>
 - Building Systems with the ChatGPT API: <https://www.deeplearning.ai/short-courses/building-systems-with-chatgpt/>
 - Reasoning with o1: <https://www.deeplearning.ai/short-courses/reasoning-with-o1/>
- **Open-Source models & Hugging-face:**
 - Open-Source Models with Hugging Face: <https://www.deeplearning.ai/short-courses/open-source-models-hugging-face/>
 - Introducing Multimodal Llama 3.2: <https://www.deeplearning.ai/short-courses/introducing-multimodal-llama-3-2/>
- **Ollama – Run LLM Locally:**
 - Learn Ollama in 15 Minutes - Run LLM Models Locally for FREE (YouTube): <https://www.youtube.com/watch?v=UtSSMs6ObqY>
 - Ollama Course – Build AI Apps Locally (YouTube – Intermediate - Optional): <https://www.youtube.com/watch?v=GWB9ApTPTv4>
- **LangChain, RAG, AI Agents:**
 - LangChain for LLM Application Development: <https://www.deeplearning.ai/short-courses/langchain-for-llm-application-development/>
 - LangChain: Chat with Your Data: <https://www.deeplearning.ai/short-courses/langchain-chat-with-your-data/>
 - AI Agents in LangGraph (Intermediate): <https://www.deeplearning.ai/short-courses/ai-agents-in-langgraph/>
 - Functions, Tools and Agents with LangChain (Intermediate): <https://www.deeplearning.ai/short-courses/functions-tools-agents-langchain/>
- **AI Agent and multi-agent:**
 - Multi AI Agent Systems with crewAI (Beginner): <https://www.deeplearning.ai/short-courses/multi-ai-agent-systems-with-crewai/>
 - AI Agentic Design Patterns with AutoGen (Beginner – Optional): <https://www.deeplearning.ai/short-courses/ai-agentic-design-patterns-with-autogen/>
- **RAG (Intermediate):**
 - Building and Evaluating Advanced RAG Applications: <https://www.deeplearning.ai/short-courses/building-evaluating-advanced-rag/>
 - Building Multimodal Search and RAG: <https://www.deeplearning.ai/short-courses/building-multimodal-search-and-rag/>
- **RAG using LlamaIndexL (Optional):**
 - Building Agentic RAG with LlamaIndex: <https://www.deeplearning.ai/short-courses/building-agentic-rag-with-llamaindex/>
 - JavaScript RAG Web Apps with LlamaIndex: <https://www.deeplearning.ai/short-courses/javascript-rag-web-apps-with-llamaindex/>
- **Other (Optional):**
 - How Transformer LLMs Work: <https://www.deeplearning.ai/short-courses/how-transformer-llms-work/>
 - Generative AI for Software Development: <https://www.deeplearning.ai/courses/generative-ai-for-software-development/>

Resources (YouTube - Free)



- **YouTube Playlists (Free)**

- Generative AI in a Nutshell: <https://www.youtube.com/watch?v=2IK3DFHRFfw->
- Generative AI Tools (From Edureka): https://www.youtube.com/watch?v=gMa_QHSAxOY&list=PL9ooVrP1hQOFIOOkGN2gbjvse8jcz-mux
- Introduction to Generative AI and LLMs (From Microsoft, covering intro RAG, Fine Tuning, and most dev topics): <https://www.youtube.com/playlist?list=PLlrXD0HtieHj2nfK54c62lcs3-YSTx3Je>
- RAG:
 - Learn RAG From Scratch – Python AI Tutorial from a LangChain Engineer: <https://www.youtube.com/watch?v=sVcwVQRHlc8>
- AI Agents:
 - How To Create Ai Agents From Scratch (CrewAI, Zapier, Cursor): <https://www.youtube.com/watch?v=PM9zr7wgJX4>
 - The Complete Guide to Building AI Agents for Beginners: <https://www.youtube.com/watch?v=MOyl58VF2ak>
 - Python Advanced AI Agent Tutorial - LlamaIndex, Ollama and Multi-LLM!: <https://www.youtube.com/watch?v=JLmI0GJuGIY>
 - ADVANCED Python AI Agent Tutorial - Using RAG: <https://www.youtube.com/watch?v=ul0QsodYct4>
- GitHub Copilot for developers (From Microsoft): <https://www.youtube.com/playlist?list=PLlrXD0HtieHgr23PS05FIncniH4dH9Na5>
- Ollama:
 - Learn Ollama in 15 Minutes - Run LLM Models Locally for FREE (YouTube): <https://www.youtube.com/watch?v=UtSSMs6ObqY>
 - Ollama Course – Build AI Apps Locally (YouTube – Intermediate - Optional): <https://www.youtube.com/watch?v=GWB9ApTPTv4>
- Other (Optional):
 - Advanced Dev topics + Practical project (From FreeCodeCamp.org): <https://www.youtube.com/watch?v=mEsleV16qdo>
 - Generative AI for Developers – Comprehensive Course: <https://www.youtube.com/watch?v=F0GQ0I2NfHA>
 - AI Agents: <https://www.youtube.com/watch?v=7WK0w9Z9mPE>



Resources (Microsoft - Free)



- **Microsoft free courses (Free):**

- [Generative AI for Beginners](#)
- [Generative AI for Beginners - .NET](#)
- [Generative AI with JavaScript](#)
- [AI for Beginners](#)
- [AI Agents for Beginners - A Course](#)
- [Data Science for Beginners](#)
- [ML for Beginners](#)
- [Mastering GitHub Copilot for C#/.NET Developers](#)
- [Mastering GitHub Copilot for Paired Programming](#)



Thank You

