Information Technology Institute

Advanced Gen-AI Development

Gen-AI based startups and real products

- **Use cases for real products using Gen AI:**

- Dxwand (AI-Powered Conversational & Knowledge Agent): https://dxwand.com
- Widebot (Arabic LLM – Exploring apps built over it): https://widebot.ai
- Velents AI (Recruitment and Interviews): : https://www.velents.com
- Nancy AI (Recruitment and Interviews): : https://nancy-ai.com
- Apriora AI (Recruitment and Interviews): https://www.apriora.ai
- Loop-x (AI Agents for different roles): https://loop-x.co
- Gemelo ai (Twin AI creator): https://gemelo.ai
- 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.

General

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

General

# Retrieval-Augmented Generation (RAG)

What is RAG?
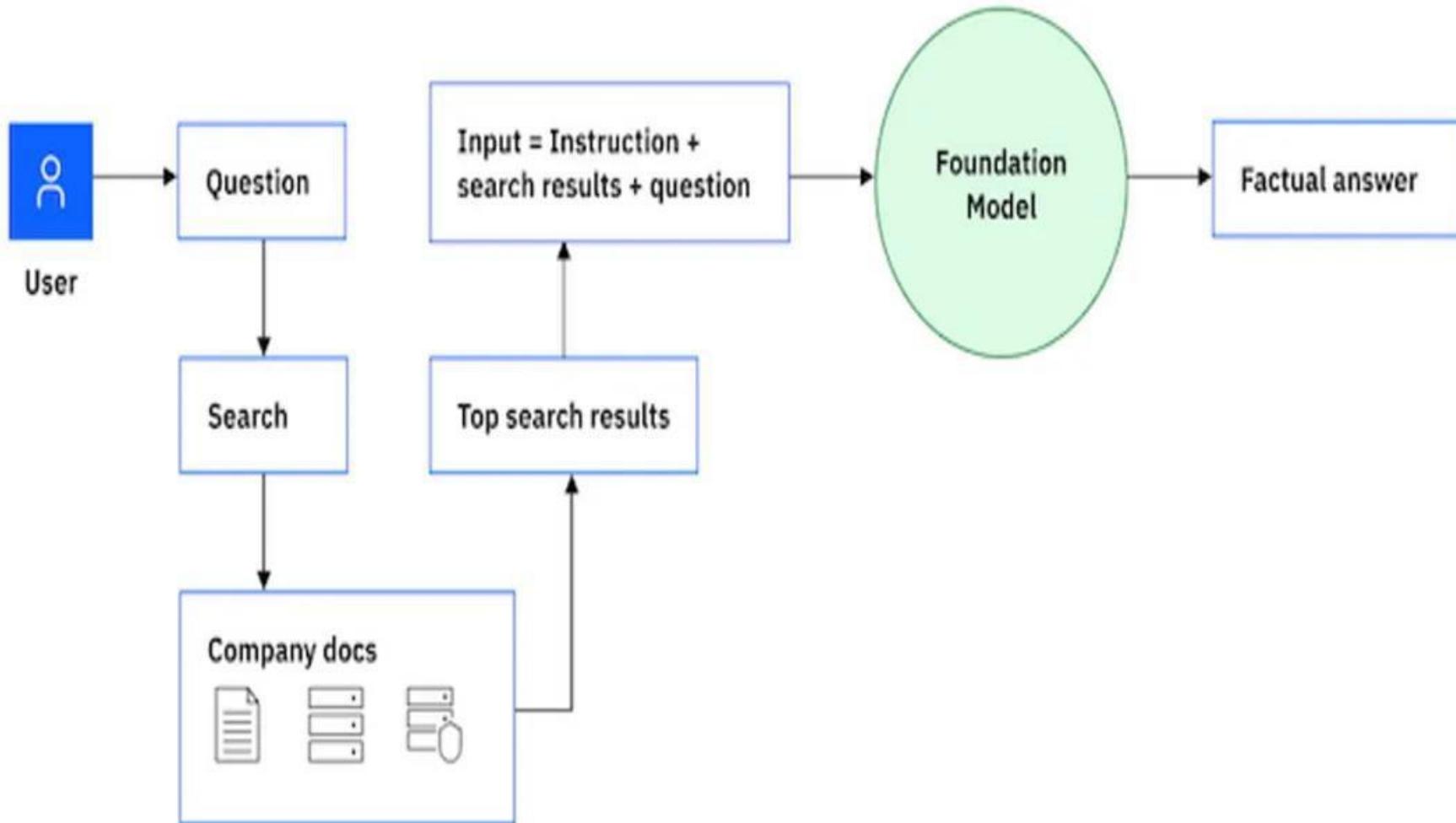
Retrieval-Augmented Generation (RAG) enhances a language model's capabilities by allowing it to access external data sources in real-time. It works by retrieving relevant information from databases or other sources and combining it with the user's query to generate a more contextually accurate and up-to-date 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/
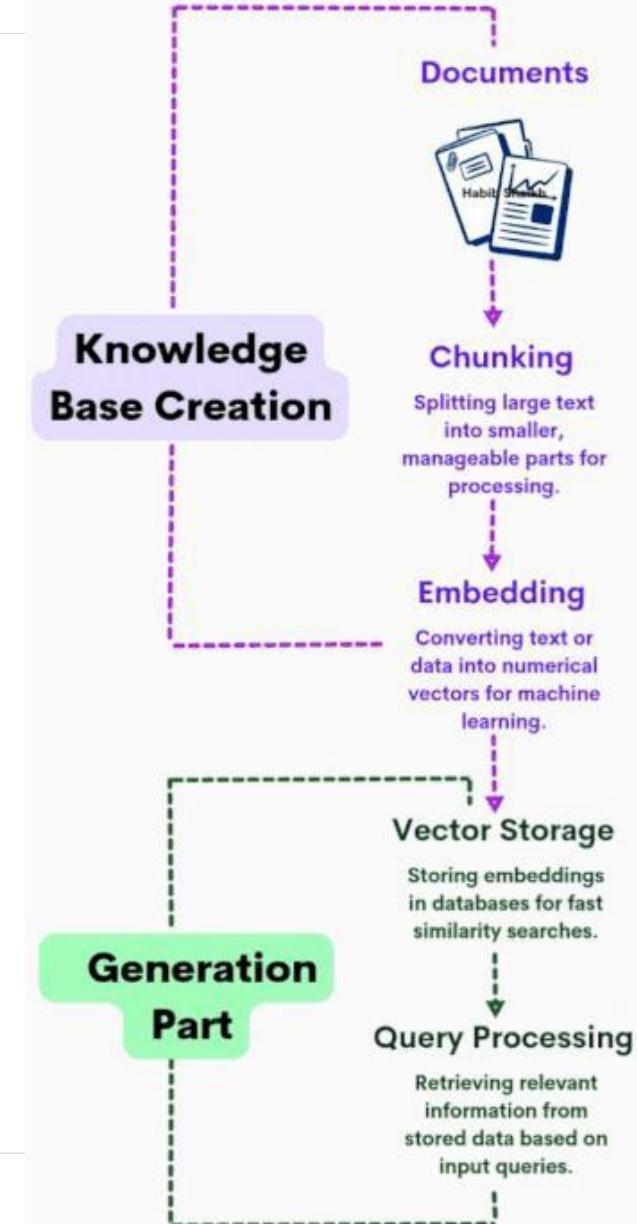
RAG data pipeline flow

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.

Chunking Strategies in RAG

General

Chunking Strategies in RAG

1. **Fixed-Size Chunking (Token-based or Character-based):**

    You simply split text into chunks of equal size — by number of characters, words, or tokens — regardless of meaning.

```csharp
int chunkSize = 500; // characters
int overlap = 100;   // characters overlap

List<string> ChunkFixed(string text)
{
    var chunks = new List<string>();
    for (int i = 0; i < text.Length; i += (chunkSize - overlap))
    {
        int end = Math.Min(i + chunkSize, text.Length);
        chunks.Add(text.Substring(i, end - i));
    }
    return chunks;
}
```

General

Chunking Strategies in RAG

2. Sentence/Paragraph-based Chunking:

Split text by sentences or paragraphs — and then group a few together until you reach a token or character limit.

```csharp
List<string> ChunkByParagraph(string text, int maxTokens = 200)
{
    var paragraphs = text.Split(new[] { "\n\n" }, StringSplitOptions.RemoveEmptyEntries);
    var chunks = new List<string>();
    string currentChunk = "";
    int tokenCount = 0;

    foreach (var p in paragraphs)
    {
        int pTokens = p.Split(' ').Length; // rough token count
        if (tokenCount + pTokens > maxTokens)
        {
            chunks.Add(currentChunk.Trim());
            currentChunk = "";
            tokenCount = 0;
        }
        currentChunk += p + "\n\n";
        tokenCount += pTokens;
    }

    if (!string.IsNullOrWhiteSpace(currentChunk))
        chunks.Add(currentChunk.Trim());

    return chunks;
}
```

Chunking Strategies in RAG

3. Sliding Window (Overlapping Chunking):

Each chunk **overlaps** with the next.

The overlap preserves context continuity — especially when a sentence is cut between two chunks.

```csharp
List<string> ChunkSlidingWindow(string text, int chunkSize = 500, int overlap = 100)
{
    var words = text.Split(' ');
    var chunks = new List<string>();

    for (int i = 0; i < words.Length; i += (chunkSize - overlap))
    {
        var chunkWords = words.Skip(i).Take(chunkSize).ToArray();
        if (chunkWords.Length == 0) break;
        chunks.Add(string.Join(" ", chunkWords));
    }
    return chunks;
}
```

General

Chunking Strategies in RAG

4. Semantic Chunking (Embedding-based Splitting)

Uses **embeddings** or **semantic similarity** to find natural boundaries between topics or ideas.

For example, if two paragraphs are semantically very different, they become separate chunks.

```
foreach (var paragraph in paragraphs)
{
    var embedding = GetEmbedding(paragraph);
    if (previousEmbedding != null)
    {
        double similarity = CosineSimilarity(embedding, previousEmbedding);
        if (similarity < 0.8) // semantic boundary
            CreateNewChunk();
    }
    previousEmbedding = embedding;
}
```

RAG – Embeddings

Embeddings are a way to represent words, sentences, or even entire documents as dense vectors in a high-dimensional space.
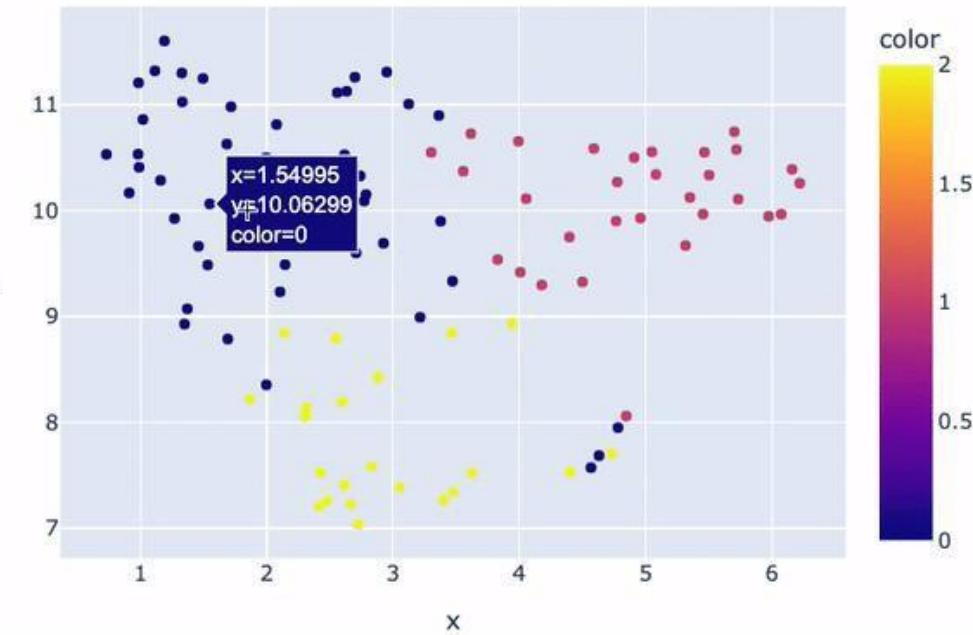
An embedding is a mathematical representation of an object, such as text. When a neural network is being trained, many representations of an object are created. Each representation has connections to other objects in the network.

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.)

	alive	mammal	rodent	bipedel	gender	plural
bunny	0.9	0.7	-0.1	-0.1	0.2	-0.3
rabbit	0.9	0.8	-0.1	-0.1	0.4	-0.1
hamster	0.9	0.6	0.7	-0.3	-0.4	-0.4
hutches	-0.9	-0.3	-0.1	-0.9	0.0	0.9

mother	0.9	0.1	0.1	0.2	0.8	-0.7
father	0.9	0.2	0.1	0.2	-0.8	-0.7
woman	0.9	0.8	-0.9	0.9	0.8	-0.8
man	0.9	0.8	-0.7	0.9	-0.8	-0.8
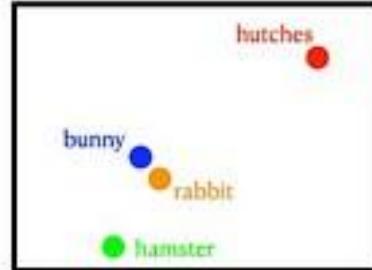
Word

Vector Embedding
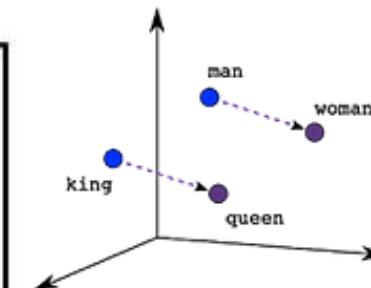
Dimensionality Reduction

Dimensionality

hutches

bunny
rabbit
hamster

woman
man
mother
father

2D Visualization

man — woman
king — queen

Male-Female

walked
walking
swam
swimming

Verb Tense

Turkey — Ankara
Canada — Ottawa
Spain — Madrid
Italy — Rome
Germany — Berlin
Russia — Moscow
Japan — Tokyo
Vietnam — Hanoi
China — Beijing
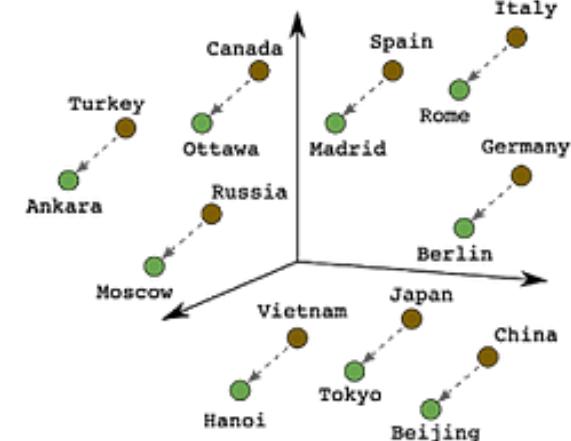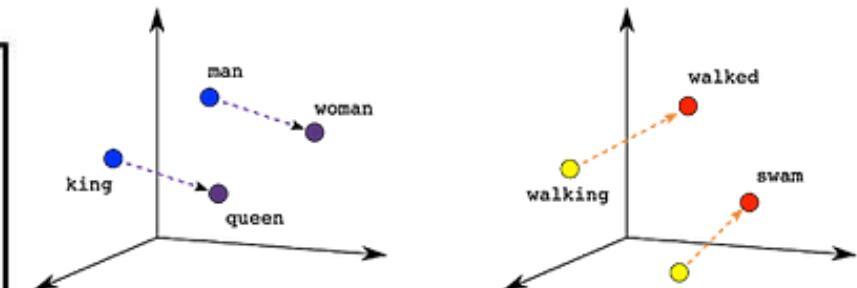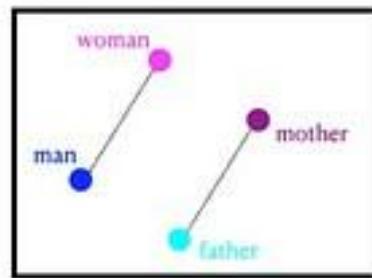
Country-Capital

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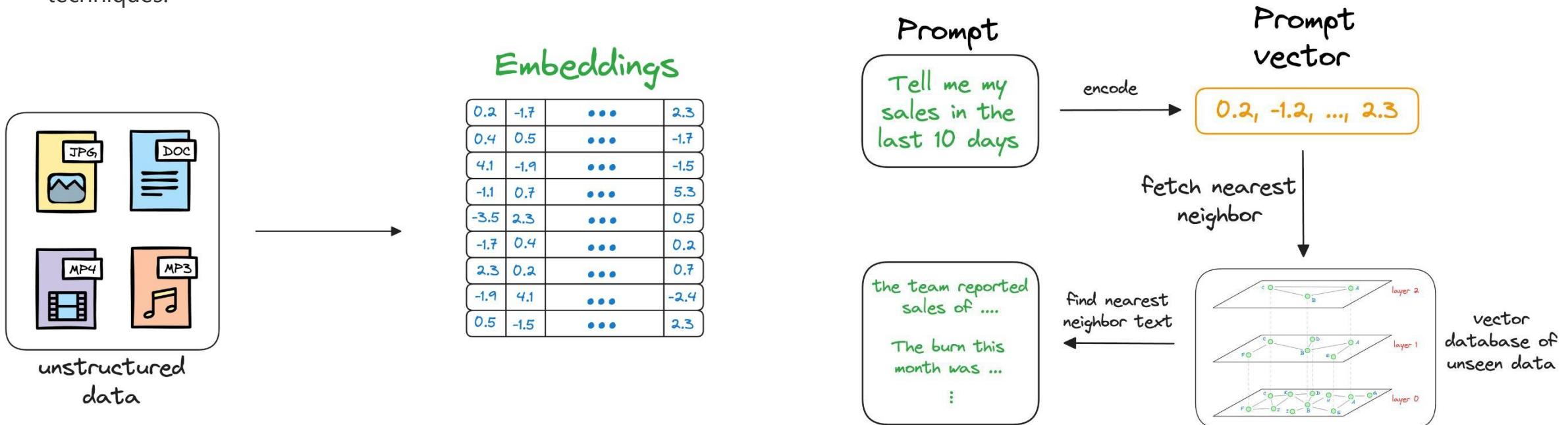
```
1  {
2    "object": "list",
3    "data": [
4      {
5        "object": "embedding",
6        "embedding": [
7          0.0023064255,
8          -0.009327292,
9          .... (1536 floats total for ada-002)
10         -0.0028842222,
11       ],
12       "index": 0
13     }
14   ],
```

API Reference - OpenAI API
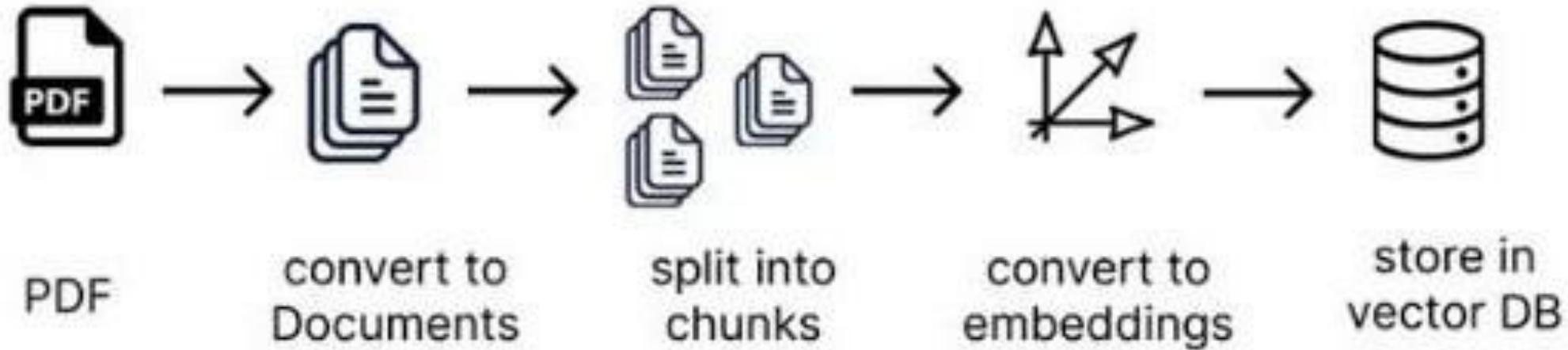
RAG – Vector databases

Vector database stores unstructured data (text, images, audio, video, etc.) in the form of vector embeddings.
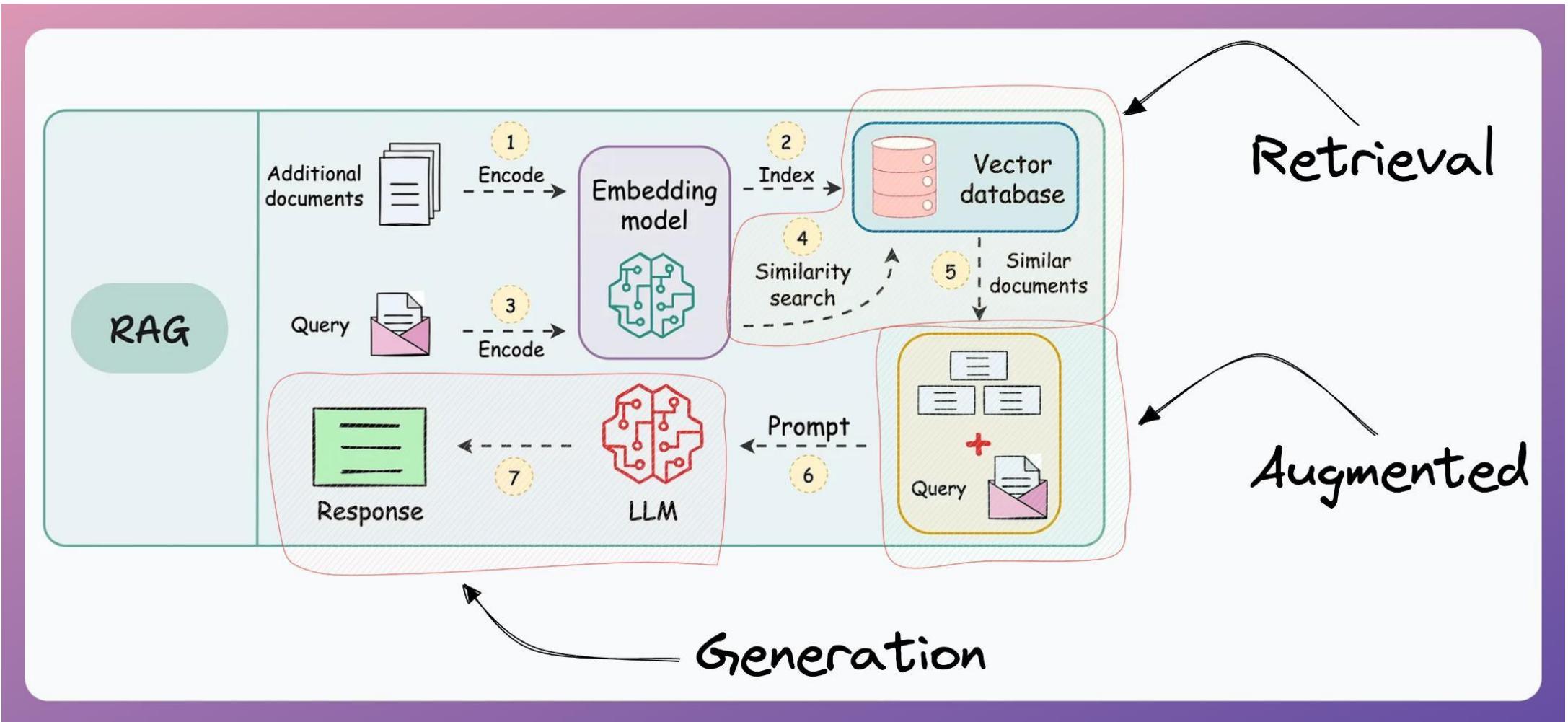
Each data point, whether a word, a document, an image, or any other entity, is transformed into a numerical vector using ML techniques.

Embeddings

0.2	-1.7	•••	2.3
0.4	0.5	•••	-1.7
4.1	-1.9	•••	-1.5
-1.1	0.7	•••	5.3
-3.5	2.3	•••	0.5
-1.7	0.4	•••	0.2
2.3	0.2	•••	0.7
-1.9	4.1	•••	-2.4
0.5	-1.5	•••	2.3

unstructured data

Prompt

Tell me my sales in the last 10 days

encode

Prompt vector

0.2, -1.2, ..., 2.3

fetch nearest neighbor

the team reported sales of ....

The burn this month was ...
:

find nearest neighbor text

layer 2

layer 1

layer 0

vector database of unseen data

RAG – Putting all together

PDF → convert to Documents → split into chunks → convert to embeddings → store in vector DB
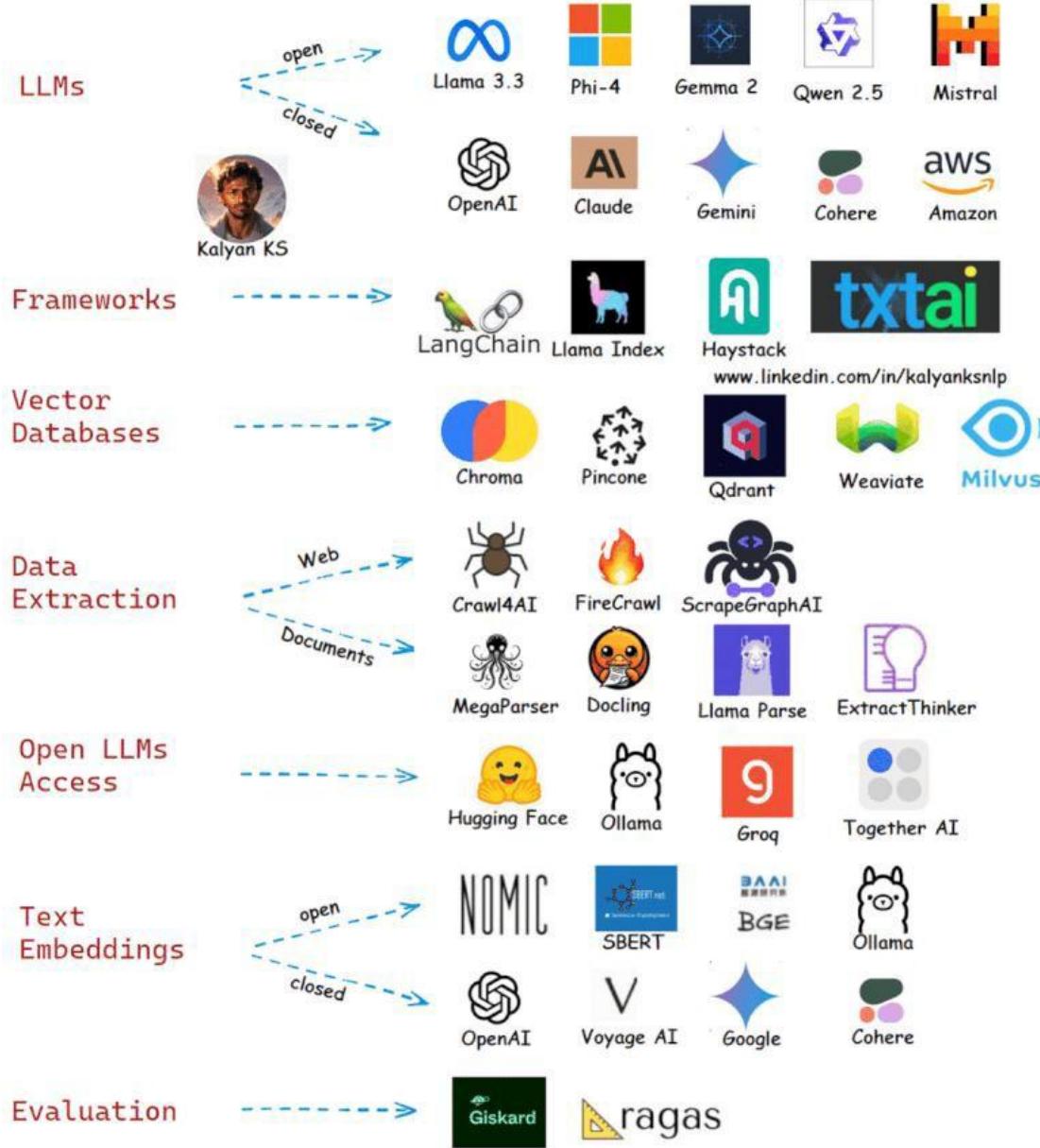
RAG re-represented

RAG Developer's Stack
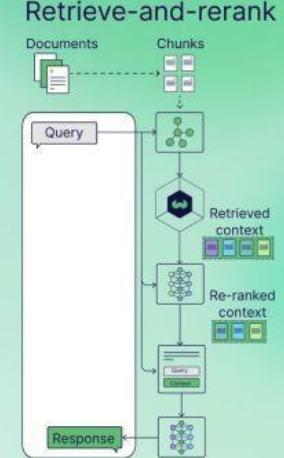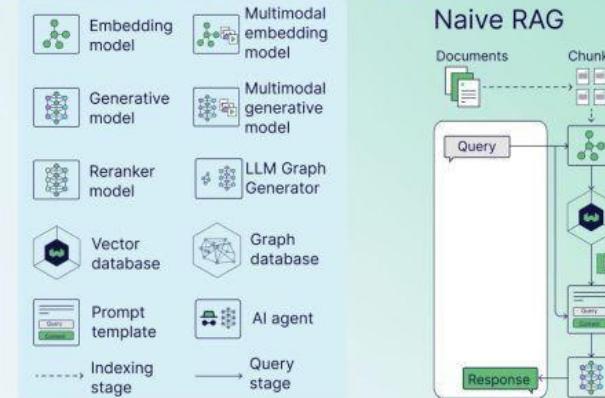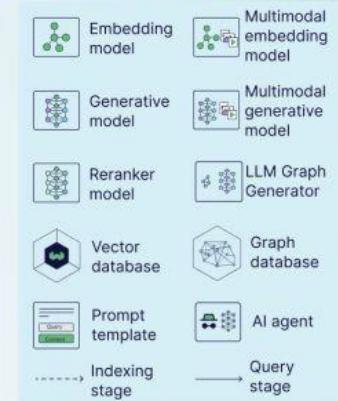
RAG developer' stack
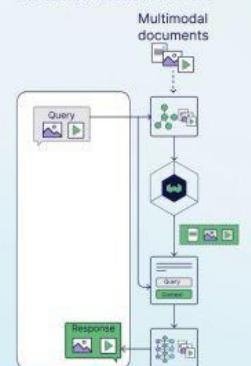
RAG Architectures

Retrieval–Augmented Generation Architectures
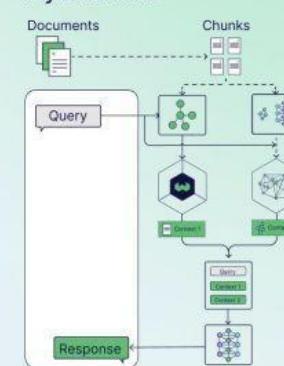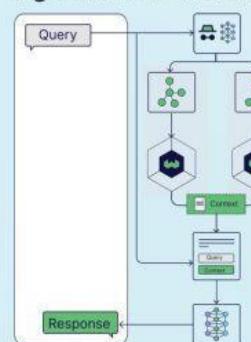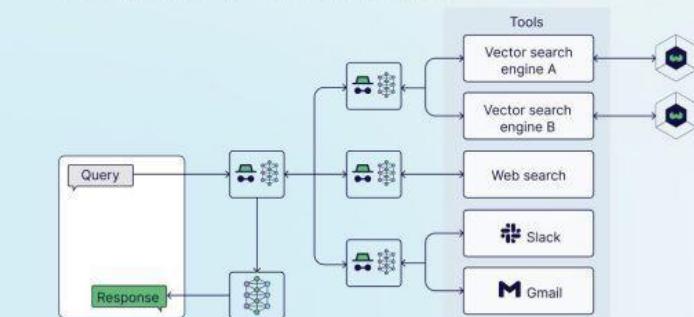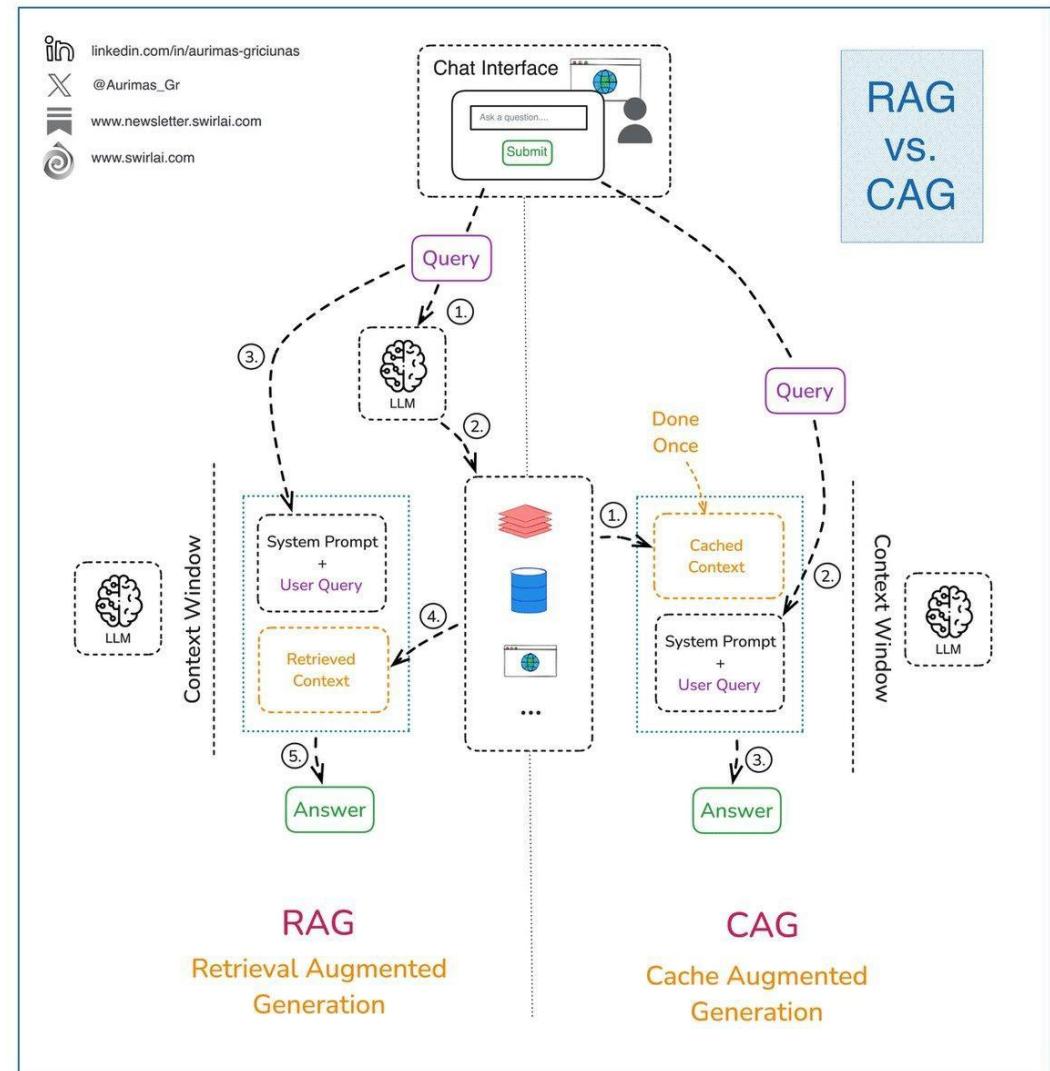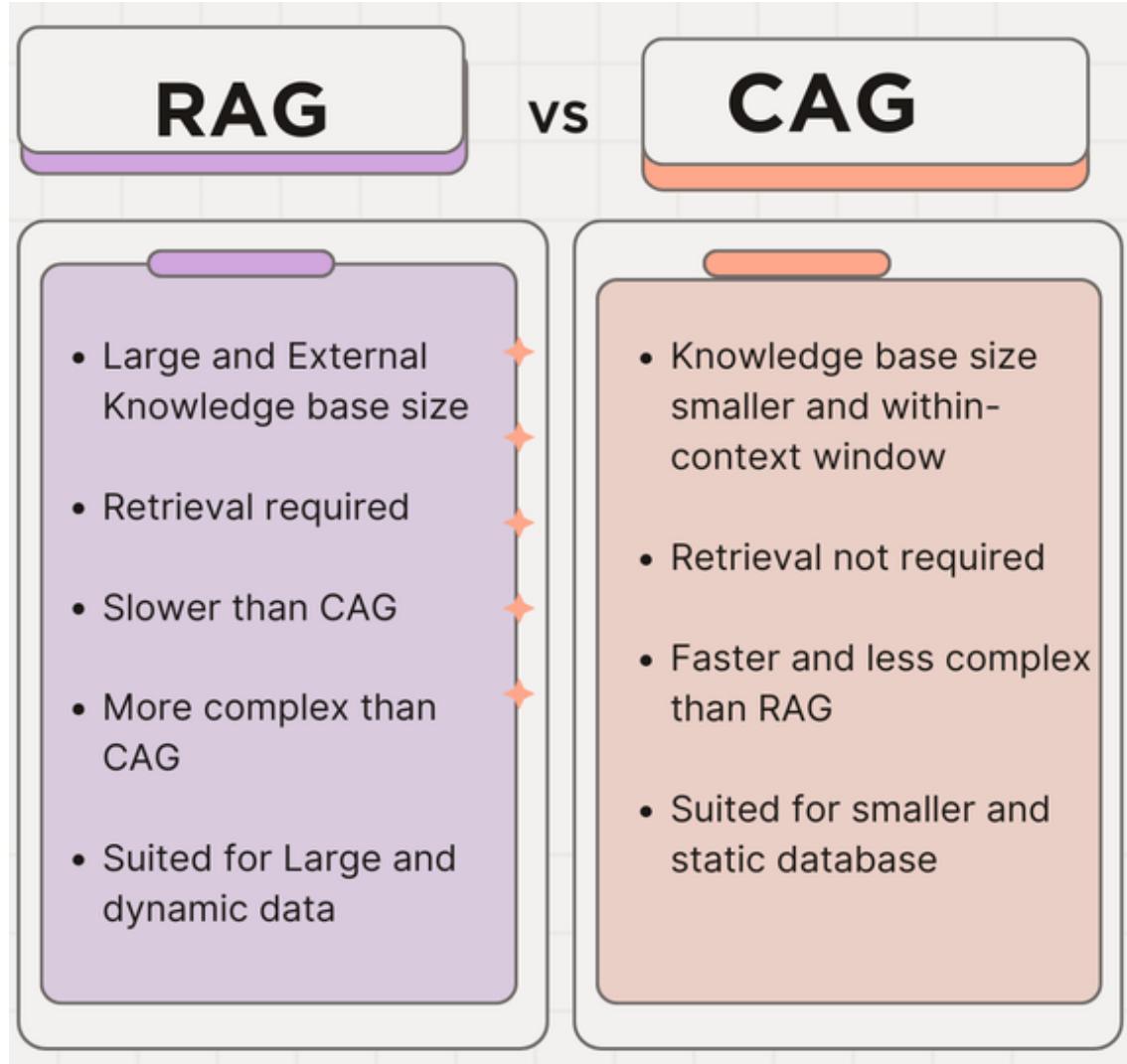
General

# Cache-Augmented Generation (CAG)

Retrieval-Augmented Generation (RAG)

RAG use cases/ examples:

Retrieval-Augmented Generation (RAG) enhances large language models (LLMs) by retrieving relevant external data before generating responses, improving accuracy and reducing hallucinations. Research suggests RAG is particularly effective in knowledge-intensive tasks, with evidence leaning toward its broad applicability across industries like tech, finance, and healthcare. Below are core use cases, drawn from real-world implementations:

- **Customer Support Chatbots**: Dynamically pulls from knowledge bases to provide accurate, context-aware answers, reducing resolution times by up to 28%.

- **Internal Knowledge Management**: Enables natural language queries over policies, documents, and databases, boosting employee productivity.

- **Content Generation and Summarization**: Automates research from diverse sources to create personalized content or distill long materials.

- **Analytics and Reporting**: Retrieves data from CRMs or ERPs to generate actionable insights and reports on demand.

- **Decision Support in Specialized Fields**: Supports healthcare diagnostics, legal research, or financial compliance by grounding responses in current guidelines.

- **Personalization and Recommendations**: Integrates user data for tailored suggestions, such as lead scoring or candidate matching.

Retrieval-Augmented Generation (RAG)

RAG use cases/ examples:

RAG's flexibility spans customer-facing tools, internal operations, and specialized analytics. Below, we detail categories with examples, including implementation nuances and outcomes. A comprehensive table follows for quick reference.

1. Customer Support and Engagement

RAG transforms static chatbots into dynamic assistants by pulling real-time data from help centers or CRMs, ensuring responses align with current policies. This reduces escalations and fosters trust through cited sources.

- **DoorDash Delivery Support**: A chatbot condenses driver conversations, retrieves from knowledge bases and historical tickets, and generates LLM responses with guardrails (e.g., LLM Judge for coherence). Benefits: Monitors hallucinations, achieves high relevance scores.
- **LinkedIn Tech Support**: Builds a knowledge graph from tickets to retrieve sub-graphs for Q&A, deployed at scale. Outcome: 28.6% faster resolutions.
- **Thomson Reuters Executive Support**: Chunks and embeds curated databases for similarity search, refining outputs with seq-to-seq models. Key: Maintains conversational flow while curbing errors.
- **General Enterprise Chatbots (Glean)**: Dynamically fetches from docs and policies, reflecting updates instantly. Reduces ticket volumes by enabling self-service.

Retrieval-Augmented Generation (RAG)

RAG use cases/ examples:

RAG's flexibility spans customer-facing tools, internal operations, and specialized analytics. Below, we detail categories with examples, including implementation nuances and outcomes. A comprehensive table follows for quick reference.

2. Internal Knowledge Management and Q&A

For enterprises, RAG powers semantic search over siloed data, respecting access controls and accelerating onboarding or policy lookups.

- **Bell Policies Chatbot**: Modular pipelines process multi-source docs with incremental indexing and DevOps integration. Supports batch updates for freshness.
- **Royal Bank of Canada (RBC) Arcane**: Navigates PDFs, Excels, and web platforms via chatbot, extracting concise info with sources. Addresses parsing challenges across formats.
- **Assembly's Dora AI**: Retrieves from file storage for employee queries (e.g., PTO policies), linking to originals. Streamlines HR access.
- **Enterprise Q&A (Glean)**: Natural language over emails/wikis, boosting productivity by 20-30% in info hunts.

General

Retrieval-Augmented Generation (RAG)

RAG use cases/ examples:

RAG's flexibility spans customer-facing tools, internal operations, and specialized analytics. Below, we detail categories with examples, including implementation nuances and outcomes. A comprehensive table follows for quick reference.

3. Content Creation and Summarization

RAG automates research-heavy workflows, grounding outputs in verified sources to produce high-quality, up-to-date content.

- **Harvard Business School ChatLTV**: Embeds course corpus (cases, notes, Slacks) into vector DB for Slack-integrated Q&A. Uses LLM judges for testing.
- **Vimeo Video Interaction**: Processes transcripts with multi-window summaries, retrieves for Q&A, and embeds playable clips. Pregenerates pairs for engagement.
- **AI Writing Assistant (Merge)**: Scrapes dictionaries for word definitions in feedback, with verifiable links. Handles evolving language dynamically.
- **Content Workflows (Glean)**: Pulls market/competitor data for blogs or summaries, saving writers hours on research.

4. Analytics, Reporting, and Fraud Detection

Integrating with data tools, RAG enables self-serve insights from structured/unstructured sources, ideal for non-technical users.

Grab Report Summarizer and A Bot*: Queries Data-Arks APIs for fraud data, summarizes via Slack. Saves 3-4 hours/report; enables cross-team analysis.

- **Pinterest Text-to-SQL**: Indexes table summaries for query embedding, selects top tables for LLM processing. Tackles table identification in analytics.
- **Ema's Sales Analyst (Merge)**: Retrieves CRM/ERP data for quota reports via text queries. Delivers actionable, customizable visuals.
- **Causal Financial Metrics**: Ingests P&L from accounting tools, computes on-demand (e.g., runway). Supports modifiable models for planning.

Retrieval-Augmented Generation (RAG)

RAG use cases/ examples:

RAG's flexibility spans customer-facing tools, internal operations, and specialized analytics. Below, we detail categories with examples, including implementation nuances and outcomes. A comprehensive table follows for quick reference.

5. Personalization, Recommendations, and Decision Support

RAG excels in tailoring outputs with user/context data, extending to high-stakes domains like healthcare and legal.

- **Telescope Lead Recommendations (Merge)**: Pulls CRM opportunity data for ML-driven suggestions. Improves personalization over static models.
- **Peoplelogic's Noah Interview Prep**: Fetches ATS details for reminders/notes. Enhances readiness without manual digging.
- **Ramp NAICS Classification**: Embeds business info against code DB via Clickhouse, with LLM refinement and guardrails. Resolves prior inconsistencies.
- **Healthcare Decision Support (Glean)**: Retrieves guidelines/patient data for diagnostics; surfaces new research. Aids evidence-based practice.
- **Legal Contract Review (Glean)**: Pulls precedents/clauses with citations for verification. Speeds due diligence, mitigates risks.
- **Financial Compliance (Glean)**: Analyzes regs/transactions for audits. Spots patterns with real-time context.
- **HR Candidate Matching (Merge)**: Combines HRIS data and web scrapes for role-fit recommendations. Boosts hiring precision.

6. Product and Sales Optimization

RAG uncovers insights from tickets or meetings, informing iterations without manual analysis.

- **Product Copilot (Merge)**: Retrieves ticketing data to prioritize issues/opportunities, with source links for trust.
- **Sales Meeting Feedback (Merge)**: Integrates CRM notes for performance takeaways. Drives skill improvements.

AI Agents and Agentic framework

AI agents vs. Agentic software

AI Agent: An autonomous system that perceives its environment, makes decisions, and takes actions to achieve goals.

Agentic Software: Software designed with autonomous components that can make decisions and take actions on behalf of users.

Key Difference:

AI Agent = Single autonomous entity (like a chatbot that can book meetings)

Agentic Software = Full application with multiple agent capabilities (like a CRM that automatically responds to customers, schedules followups, and analyzes sentiment)

Simple Analogy:

AI Agent = A smart employee

Agentic Software = An entire smart team working together

Agent Architectures

1. **ReAct (Reasoning + Acting)**

2. **Plan-and-Execute**

3. **Multi-Agent Swarms**

General

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.

23 https://cloud.google.com/discover/what-are-ai-agents
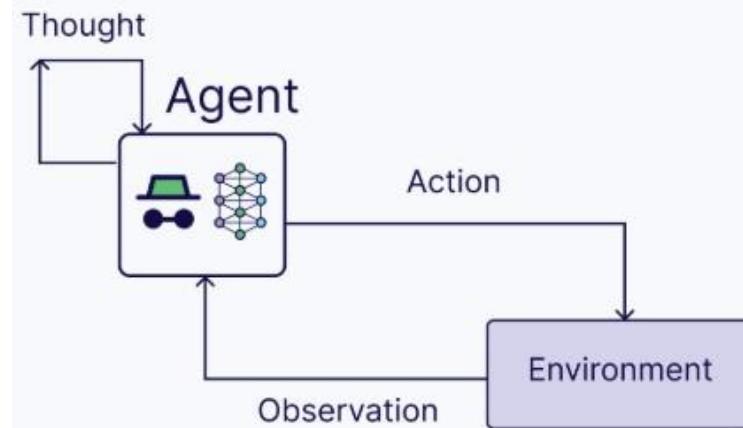
Agentic framework – Reasoning paradigms

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:
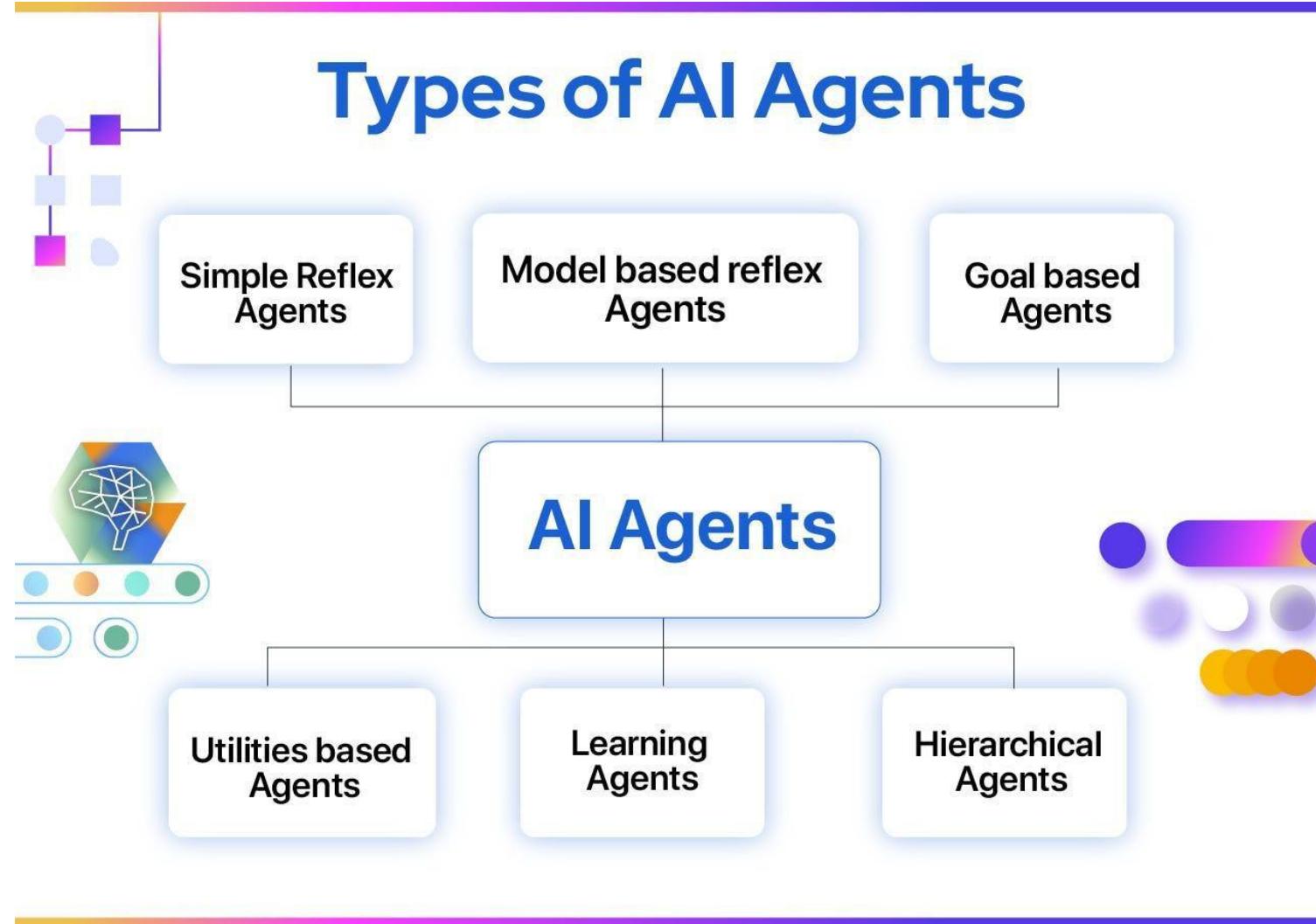
1. Thought: Upon receiving the user query, the agent reasons about the next action to take

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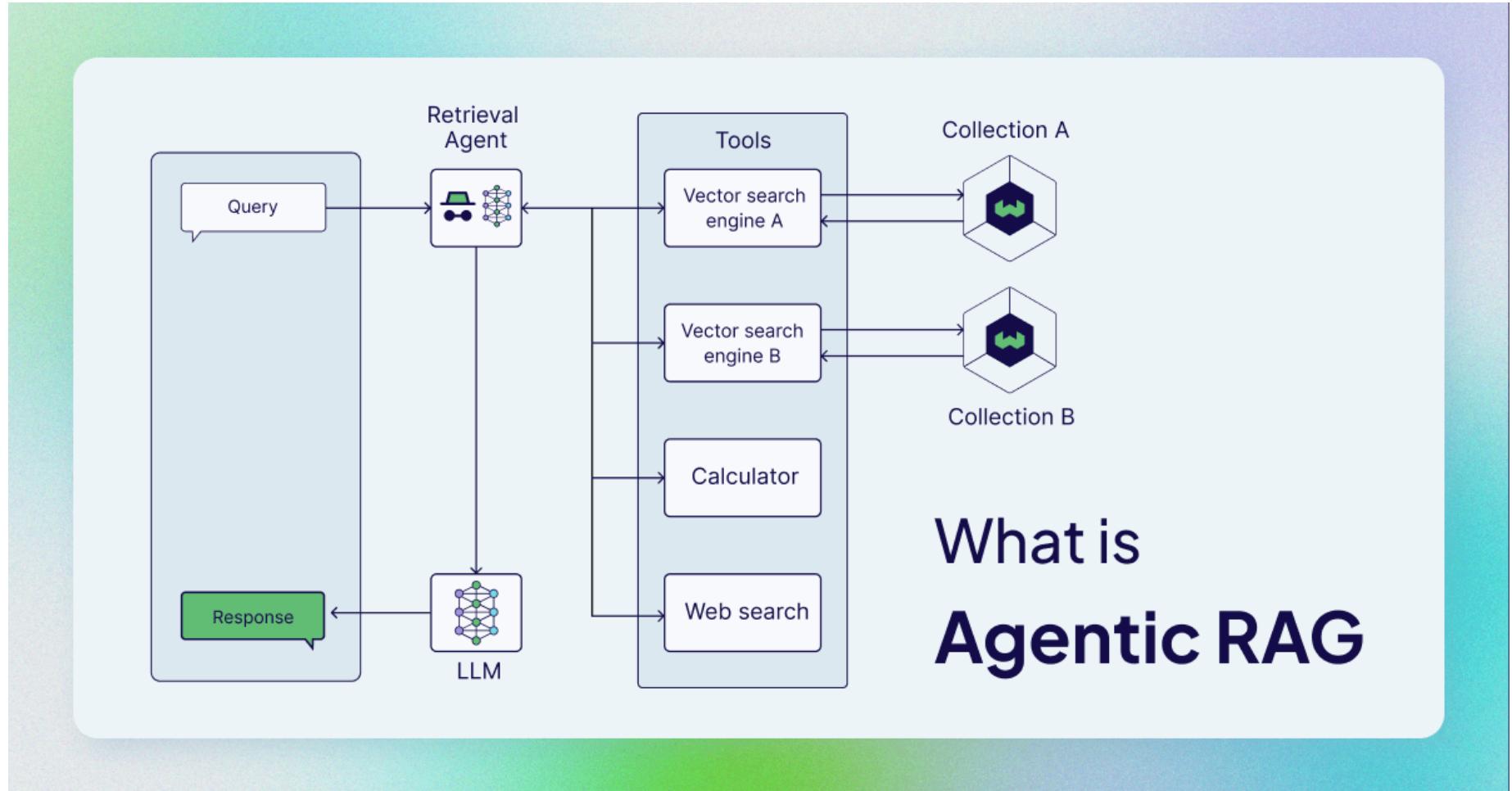
The paper *ReWOO: Decoupling Reasoning from Observations for Efficient Augmented Language Models* presents a groundbreaking shift in ALM design. ReWOO (Reasoning Without Observation) restructures how LLMs interact with external tools by decoupling the reasoning process from tool execution. Instead of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

https://www.ibm.com/think/topics/ai-agents#Reasoning+paradigms

General

AI Agents Types

25

General
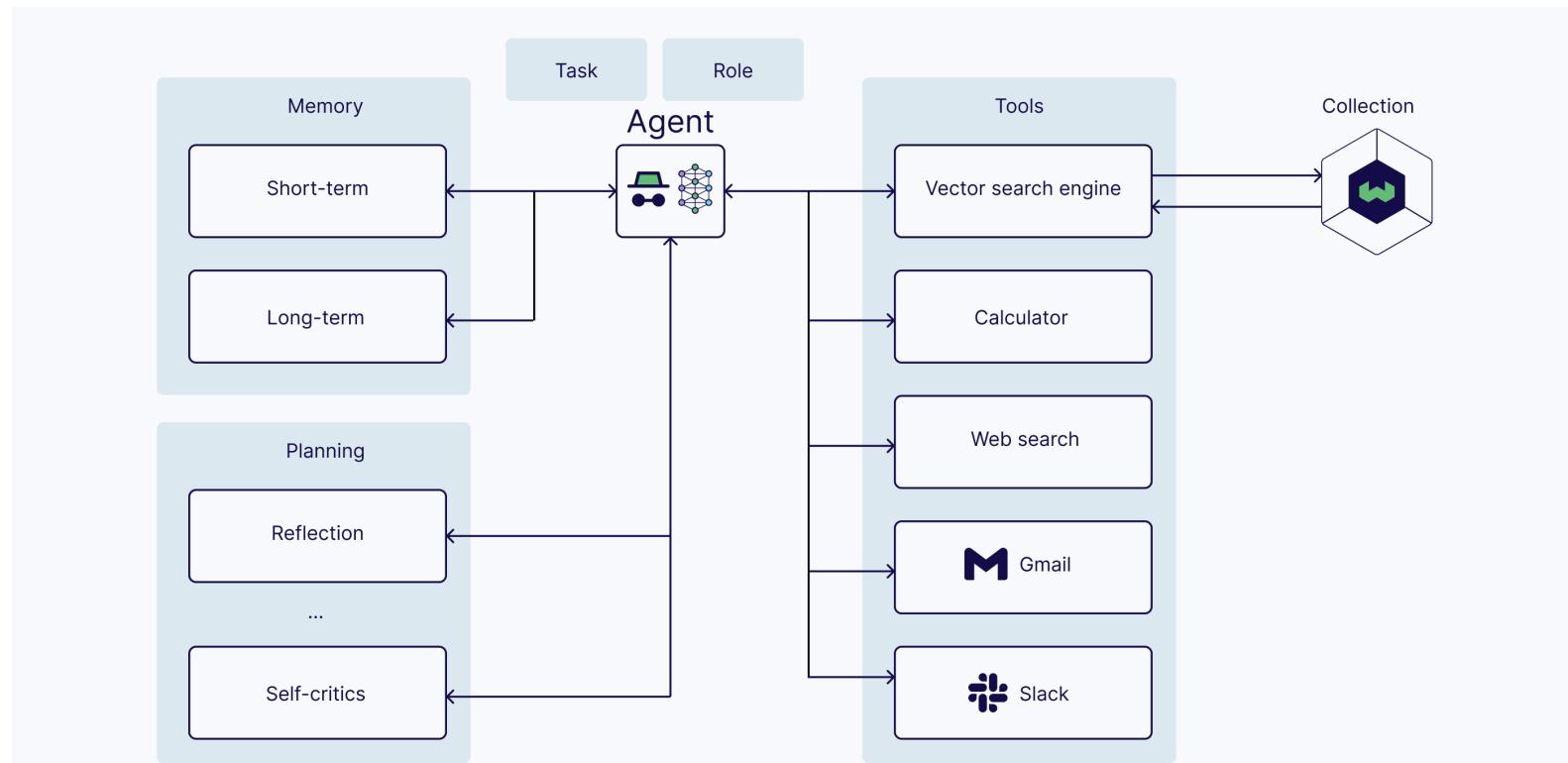
Agentic RAG

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.

https://weaviate.io/blog/what-is-agentic-rag

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.)

https://weaviate.io/blog/what-is-agentic-rag
https://www.promptingguide.ai/agents/components

General

Agentic frameworks


2025 AI Agent Tech Stack
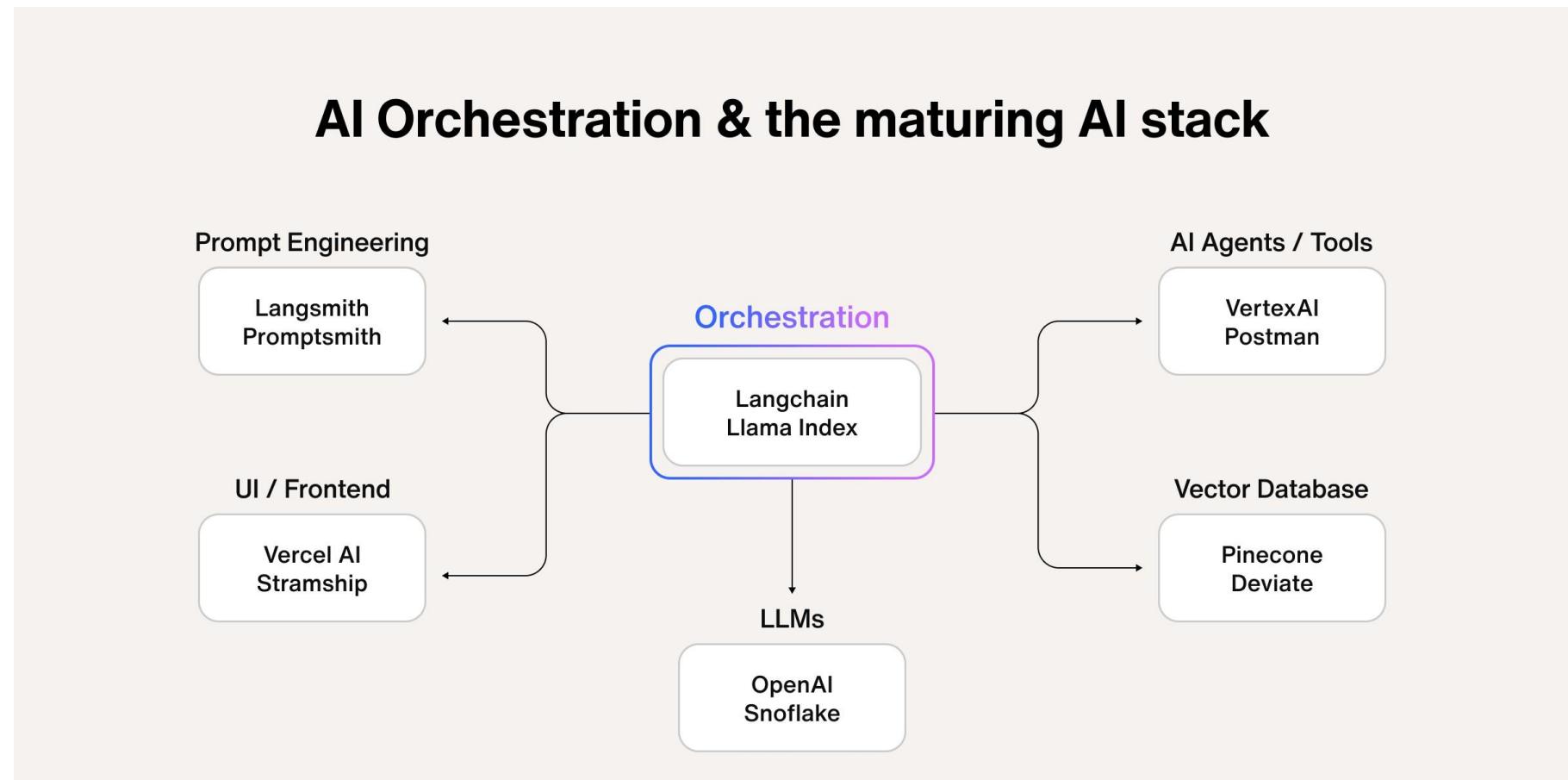
AI orchestration frameworks

General

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.

AI Orchestration & the maturing AI stack

Prompt Engineering

Langsmith
Promptsmith

Orchestration

Langchain
Llama Index

AI Agents / Tools

VertexAI
Postman

UI / Frontend

Vercel AI
Stramship

Vector Database

Pinecone
Deviate

LLMs

OpenAI
Snoflake

General

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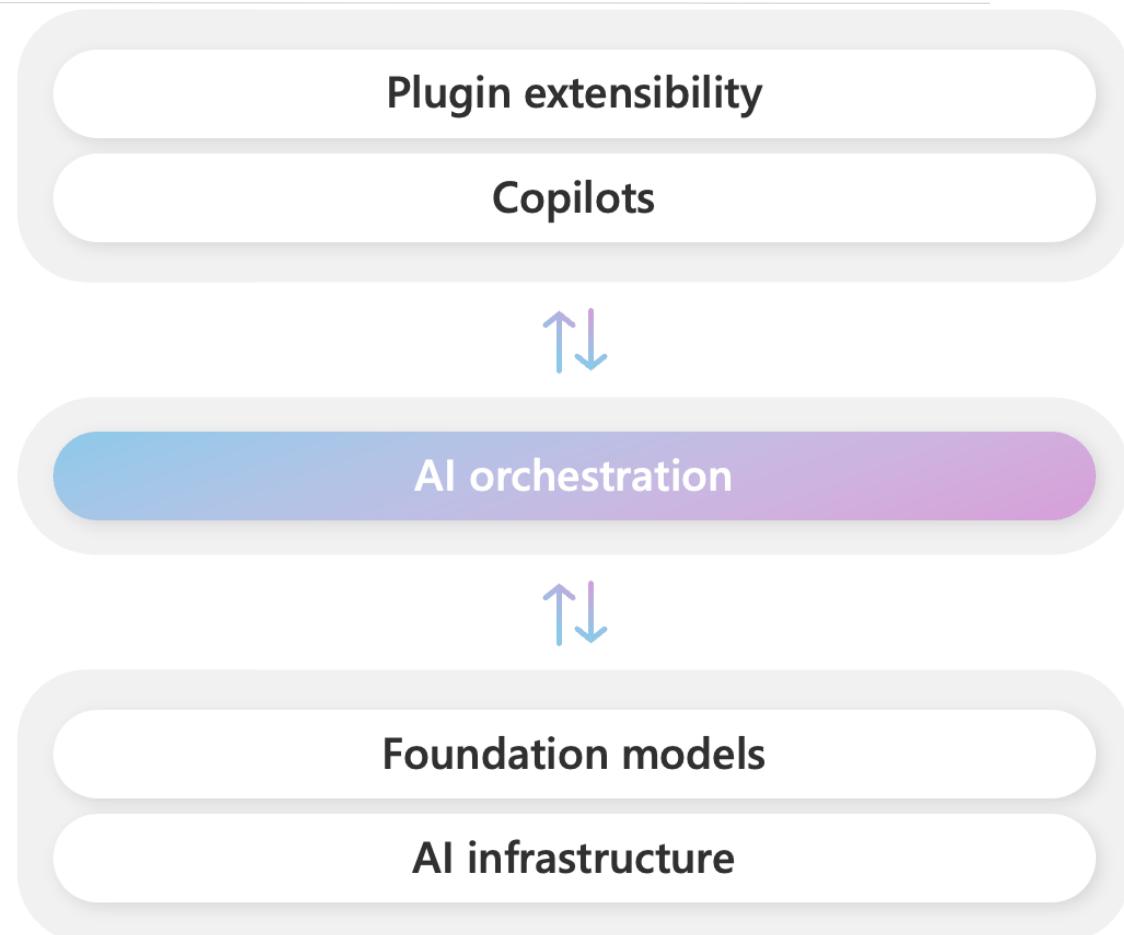
If you are interested in building applications powered by language models, then LangChain is a great place to start. It is a powerful tool that is easy to learn and use.

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.

| Plugin extensibility |
| Copilots |

⇅

| AI orchestration |

⇅

| Foundation models |
| AI infrastructure |

OpenAI APIs – Advanced tools
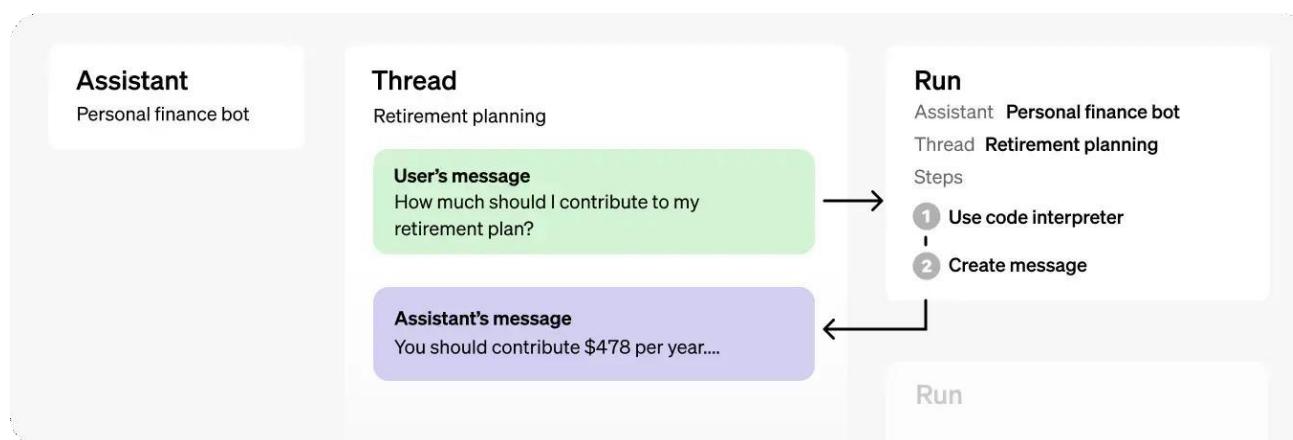
OpenAI APIs – Assistants API Overview

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
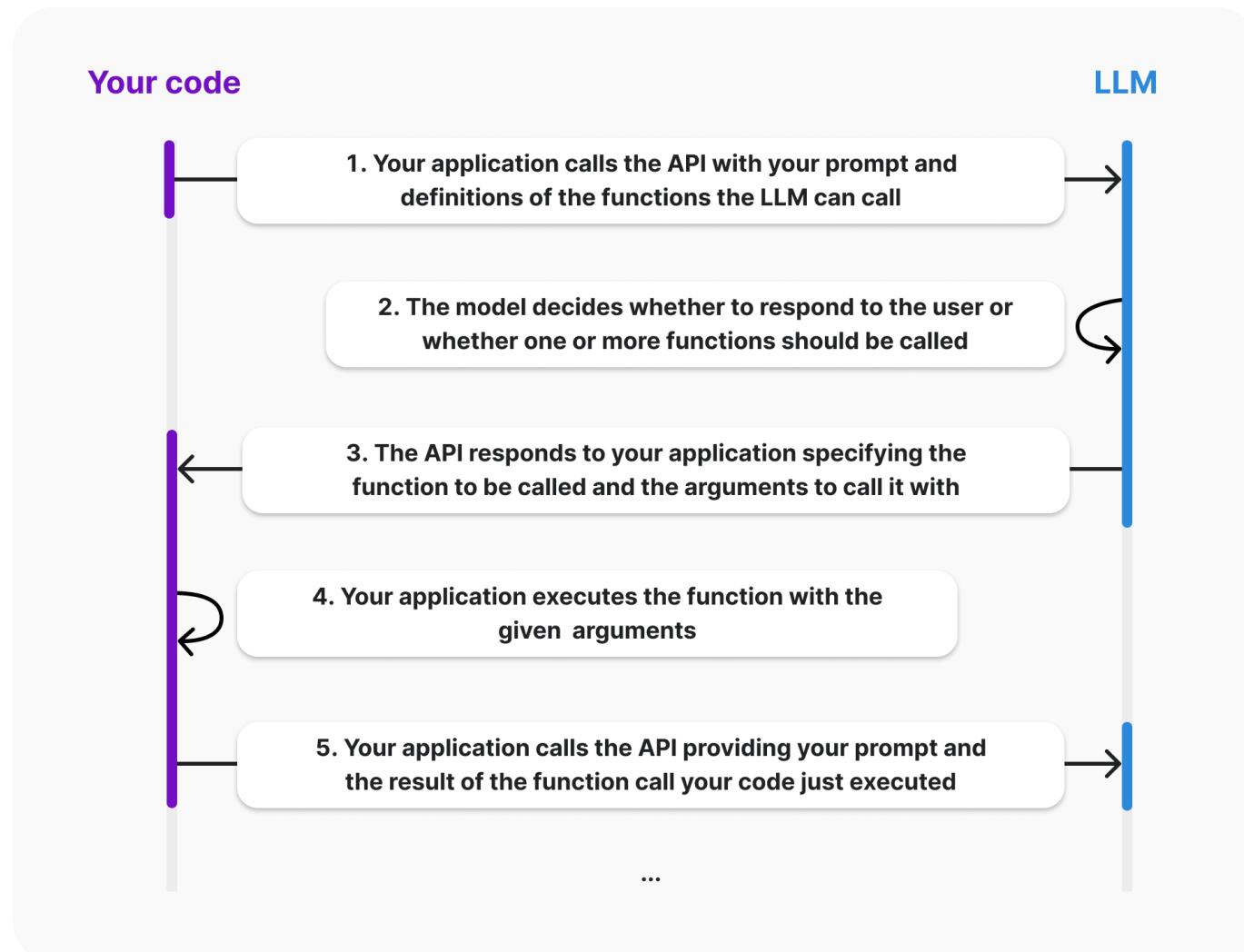
Function Calling

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.)

Your code   **LLM**

1. Your application calls the API with your prompt and definitions of the functions the LLM can call

2. The model decides whether to respond to the user or whether one or more functions should be called

3. The API responds to your application specifying the function to be called and the arguments to call it with

4. Your application executes the function with the given arguments

5. Your application calls the API providing your prompt and the result of the function call your code just executed

...

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.

```python
# 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

Introduction to Structured Outputs

Katia Gil Guzman

Aug 6, 2024

COMPLETIONS FUNCTIONS

Introduction to GPT-4o and GPT-4o mini

Juston Forte

Jul 18, 2024

COMPLETIONS VISION WHISPER

Batch processing with the Batch API

Katia Gil Guzman

Apr 24, 2024

BATCH COMPLETIONS

Assistants API Overview (Python SDK)

Ilan Bigio

Nov 10, 2023

ASSISTANTS FUNCTIONS

Processing and narrating a video with GPT's visual capabilities and the TTS API

Kai Chen

Nov 6, 2023

COMPLETIONS SPEECH VISION

Using GPT4 Vision with Function Calling

Shyamal Anadkat

Apr 9, 2024

CHAT VISION

General

Model Context Protocol (MCP)

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

What's next?

Suggested online courses for the trainees [Required]

- **After course:** Gen AI, prompt Engineering and AI code assistants (Non-Development tracks)
  - [Deeplearning.ai: prompt-engineering-with-llama-2 (Free)](#)

- **After course:** Gen AI, prompt Engineering and AI code assistants (Development tracks)
  - [Deeplearning.ai: ChatGPT Prompt Engineering for Developers (Free)](#)

- **After course:** Advanced Gen-AI programing: Advanced Gen-AI programing: RAG and Agentic software

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](#)
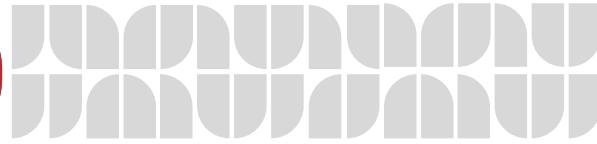
Option 2: Free courses
• [Open-Source Models with Hugging Face](#)
• [Run LLM Models Locally using Ollama](#)
• [LangChain for LLM Application Development](#)
• [LangChain: Chat with Your Data](#)
• [Multi AI Agent Systems with crewAI](#)

More resources
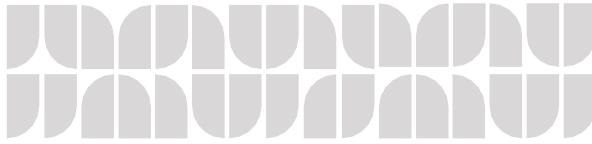
Different learning paths

Complementary online courses [Optional]

- **Advanced Gen-AI Development track – Online courses:**
  - **Open-source models and hosting LLM locally:**
    - Open-Source Models with Hugging Face
    - Run LLM Models Locally using Ollama
  - **Models Orchestration frameworks & communication protocols**
    - LangChain for LLM Application Development
    - Functions, Tools and Agents with LangChain
    - MS semantic Kernel (YouTube) – For .Net tracks.
    - MCP: Build Rich-Context AI Apps with Anthropic
  - **Building Real-world RAG systems**
    - LangChain: Chat with Your Data
    - Retrieval Augmented Generation (RAG)
    - Building and Evaluating Advanced RAG Applications
  - **Building Real-world AI Agent system**
    - Multi AI Agent Systems with crewAI
    - Building Code Agents with Hugging Face smolagents
    - DSPy: Build and Optimize Agentic Apps
    - Evaluating AI Agents
  - **Guardrails & Optimization & Cost control**
    - Safe and Reliable AI via Guardrails
    - Finetuning Large Language Models
    - Prompt Compression and Query Optimization
    - Prompt Engineering with Llama 2 & 3

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=sVcwVQRHIc8
  - 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=JLmI0GJuGlY
    - 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=F0GQ0l2NfHA
    - 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