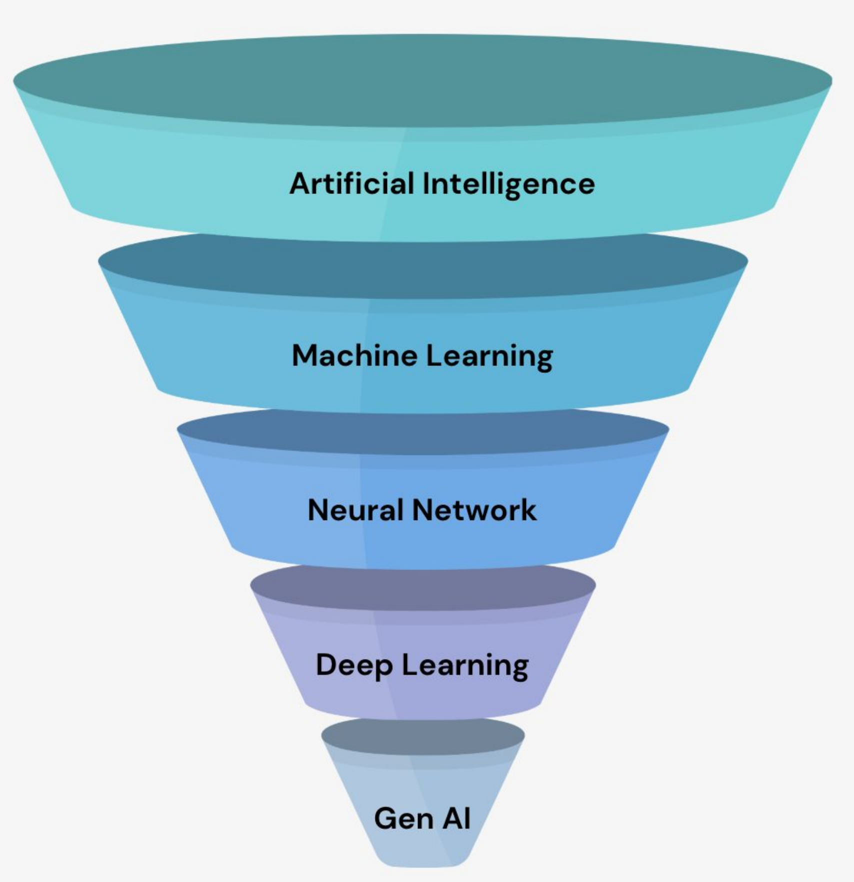
Agentic AI Theory

Agenda

1. What is GenAI ? What is Agentic AI ? What are AI Agents?
2. How are GenAI Models trained?
3. How to design Agentic Systems ?
4. Prompt Based Agents
5. RAG
6. Dynamic Agents
7. Demo - Langgraph
8. Demo - No Code tool

What is Generative AI ?



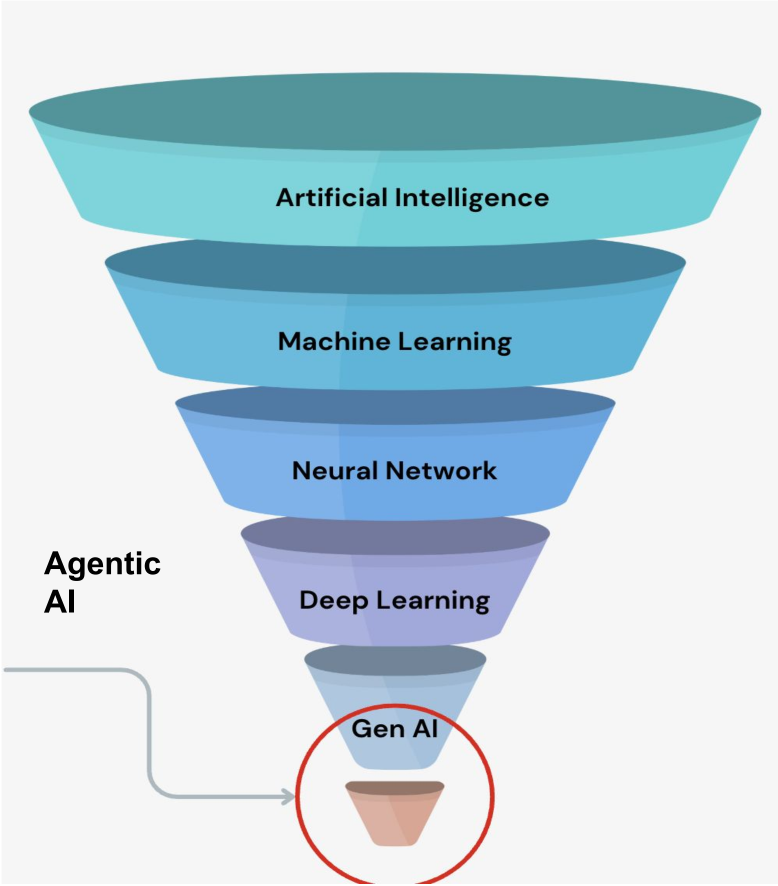
What is Generative AI?

Usually neural networks that can both understand and **generate** new data

- Generative AI is essentially **generative deep learning**, though we don't explicitly call it that.
- It follows the same principles as previous AI models
- While earlier models predicted from a limited set of categories (e.g., dog vs. cat),
- Gen AI predicts from a much larger and more open-ended set, such as words in a vocabulary, pixels in an image, or actions on a webpage.

What is Agentic AI ?

Agentic AI is an application layer built on top of Generative AI



Agentic AI decoded

Generative AI

+

Tools

Gen AI Models can call APIs or Tools
(Ex - Code executer , HTTP requests etc)

Memory

AI Models can retrieve external memory as required for a task

Plans

AI Models can generate a plan first which involves a set of actions

What are AI Agents?

*An Agent is a system that **leverages an AI model** to interact with its environment in order to achieve a user-defined objective.*

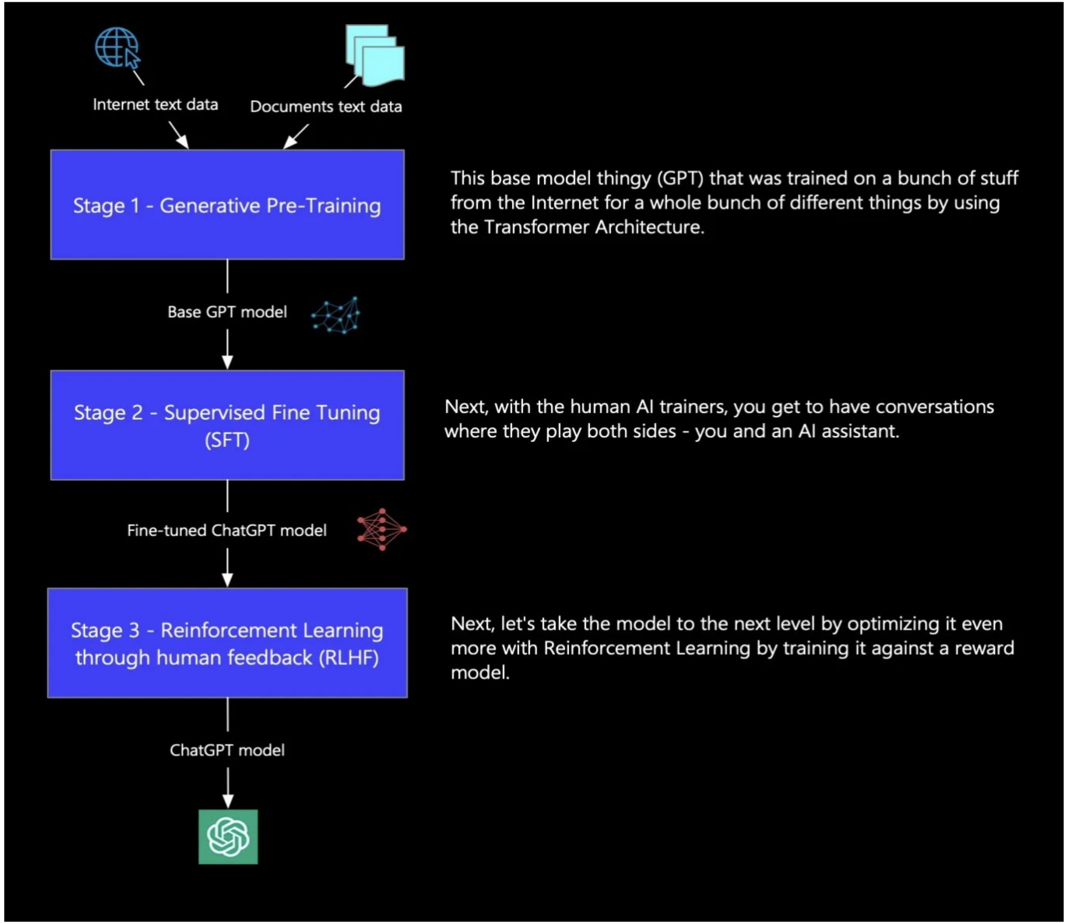
*It combines **reasoning, planning, and the execution of actions (often via external tools)** to fulfill tasks.*

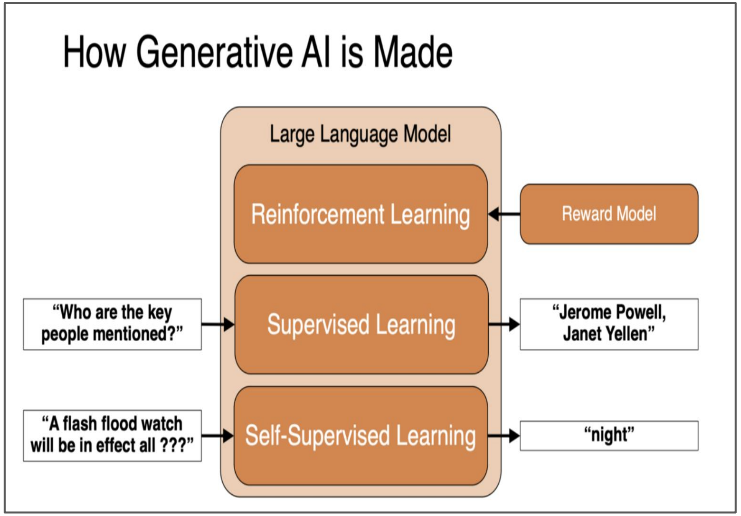
Hugging Face (Link)

In our discussion :
Agentic AI = AI Systems that use AI agents capable of acting towards an end goal . Think about AI Agents doing their parts and as a system they achieve a broader goal

This definition aligns with how openai , anthropic defines Agentic Ai

How today's Generative AI models are built





Link and Link

Note : All LLMs have a fixed context length (though increasing)

1. Scaling Limits: Compute, Memory, and Time
Think of an LLM (Large Language Model) like a **giant classroom** where every student (word/token) has to listen to every other student at the same time.

- If there are **10 students**, it's fine—they can all talk and listen.
- If there are **1,000 students**, suddenly every student is trying to listen to 999 others at once.
- As the classroom grows, it becomes **noisy, crowded, and super hard to manage**.

That's why very long inputs (lots of words) take too much time, energy, and memory for the model to handle efficiently.

2. Positional Encoding: Tracking Word Order

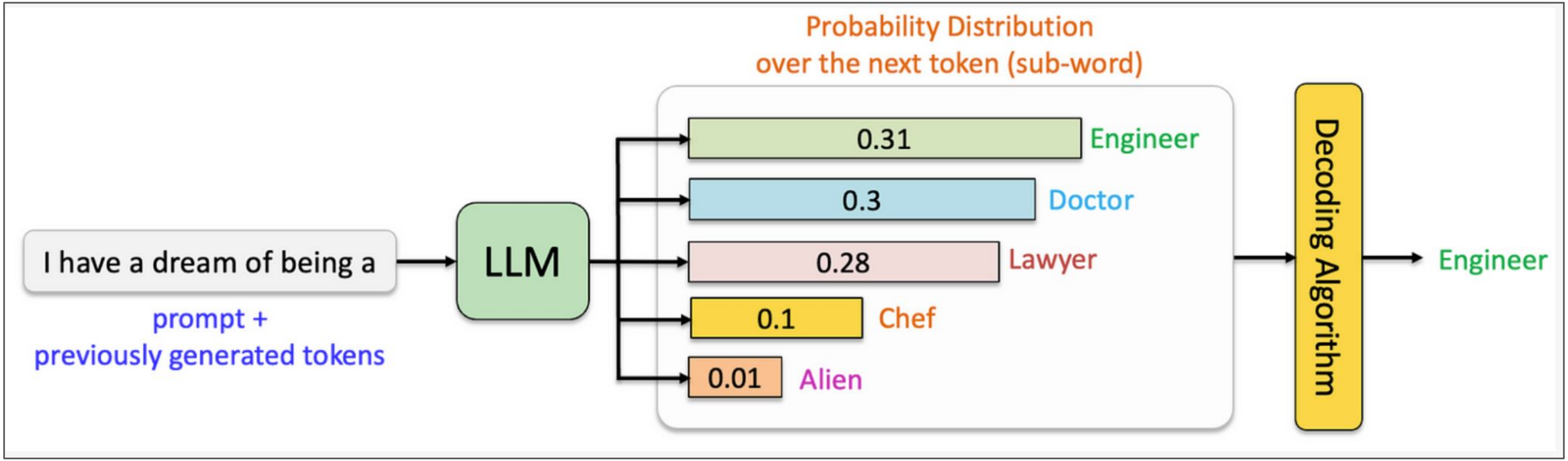
Imagine reading a sentence cut into flashcards. The model doesn't naturally know if "dog bites man" is different from "man bites dog."

- To fix this, we number the flashcards (like page numbers in a book).
- These numbers tell the model the order of words, even though it looks at all the words at once.

Older models used fixed numbering (like fixed seats in a theater).
Newer models use a rotating system (RoPE), which is like a carousel where positions keep shifting smoothly. But after too many rotations, the system gets confused and the accuracy drops.

Temperature

Most LLMs are **autoregressive** in nature, meaning they generate text token by token, with each token depending on the previously generated ones. They achieve this by assigning probabilities to different possible next words.

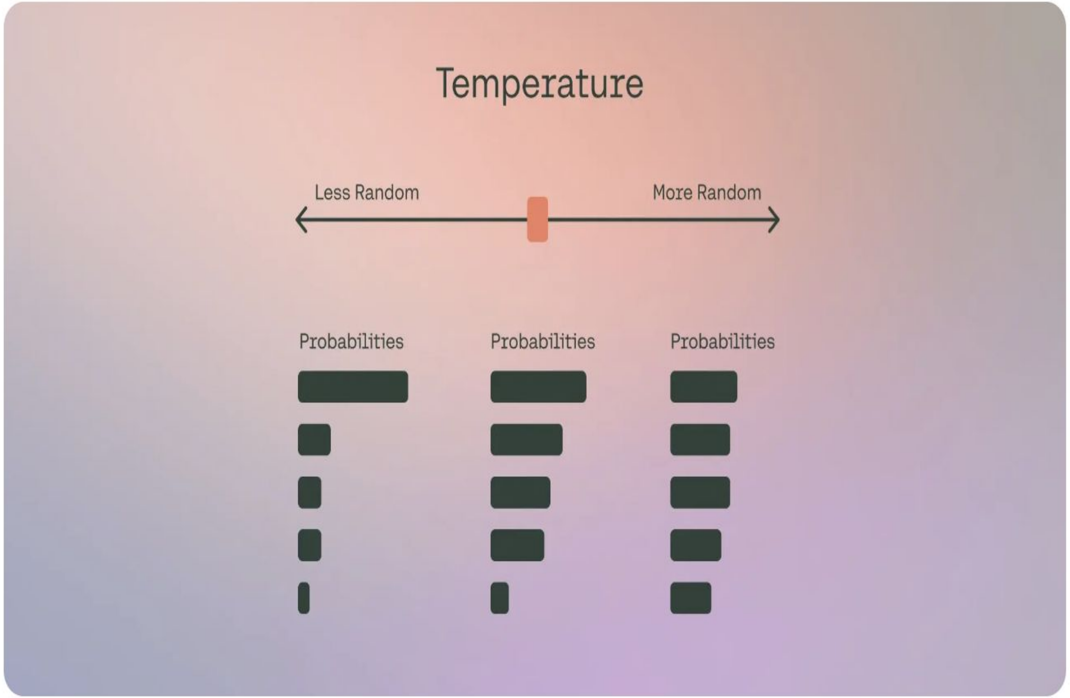


Temperature

Temperature controls this process by scaling these probabilities—

T < 1: Sharpening them for more deterministic output

T > 1: Flattening them to introduce randomness.



[Link](#)

LLM Evaluation or Benchmarkings

- **Static, Ground–Truth–Based:** Most common method due to **low cost** and reproducibility.
- **Static, Human Preference–Based:** Uses fixed questions but evaluates based on human feedback.
- **Live, Ground–Truth–Based:** Continuously updated questions paired with definitive answers.
- **Live, Human Preference–Based:** Incorporates real-time questions and evaluates using ongoing human feedbacks.

		Question Source	
		Static	Live
Evaluation Metric	Ground Truth	MMLU, HellaSwag, GSM-8K	Codeforces Weekly Contests
	Human Preference	MT-Bench, AlpacaEval	**Chatbot Arena**

Chatbot Arena Leaderboard

Arena Overview

Scroll to the right to see full stats of each model

First Place Second Place Third Place

Default | Compact View

Model 234 / 234	Overall	Hard Prompts	Coding	Math	Creative Writing	Instruction Following	Longer Query	Multi-Turn
claude-opus-4-1-202…	1	1	1	1	1	1	1	1
gemini-2.5-pro	1	2	3	1	1	1	1	1
gpt-5-high	1	2	3	1	4	1	5	2
o3-2025-04-16	2	4	3	1	7	7	10	7
chatgpt-4o-latest-2…	3	5	3	12	2	5	3	1
claude-opus-4-1-202…	3	1	1	1	1	1	1	1
gpt-4.5-preview-202…	3	5	4	7	1	2	3	1
claude-opus-4-20250…	8	5	2	5	2	2	1	6
deepseek-r1-0528	8	8	5	8	7	13	10	13
glm-4.5	8	5	4	6	8	7	5	8
gpt-5-chat	8	5	4	6	5	4	3	1
grok-4-0709	8	9	9	1	3	5	5	6
kimi-k2-0711-preview	8	8	4	10	10	15	12	7
mistral-medium-2508	8	5	3	-	14	7	3	7
qwen3-235b-a22b-ins…	8	2	2	1	8	4	3	2
claude-opus-4-20250…	9	9	6	8	3	7	3	7
gpt-4.1-2025-04-14	13	9	10	33	7	8	5	7
grok-3-preview-02-24	13	10	16	20	7	8	5	13
gemini-2.5-flash	14	18	26	5	6	8	8	15
qwen3-235b-a22b-thi…	14	9	5	5	7	7	8	13
claude-sonnet-4-202…	17	9	3	7	7	6	4	8

Link

What are the key design questions to think about in designing Gen AI solutions

What’s unique about Generative AI System Design

Biggest challenge of GenAI systems are that it will give non deterministic outputs . But enterprises prefer very high predictability of outcome for seamless experience and automation. **How to get deterministic outcome from a non deterministic technology is unique?** Apart from that add - Hallucination, Sycophancy, Prompt sensitivity , Latency etc

Does the image represent a cat or a dog?

All these answers are right (but might not be useful for all use cases and workflows)

- dog
- yes, this is a cute brown dog
- well, this isn’t a cat for sure, it’s a dog

Key Design Questions:

- Choosing the right **AI model** for a given use-case
- **Choosing the right setup/application format**
- Designing your AI application:
    - Workflow Agents
    - RAG systems
    - Agentic Systems (Dynamic Agents - mostly multi agent)

Choosing the right AI model

Why Closed Source/Hosted models are Better for Early Use Cases

- Quick Deployment: Ready-to-use with minimal setup.
- Fully Managed: No need for in-house maintenance or support.
- Seamless Updates: Changes happen at the API level, avoiding infrastructure overhauls.
- Operational Benefits: Prompt caching etc.

When to Use Open Source

- High Security/Privacy Needs: Avoid sharing sensitive data with external providers.
- In-House Expertise: Skilled teams for model infrastructure and development.
- Mature Use Cases: Familiarity with LLM limitations allows fine-tuning.
- Niche Applications: Tailored use cases requiring domain-specific customization

Key Considerations: Choosing the right AI model

- **Performance:** Find a benchmark closer to your task, if not, it’s always okay to go with overall benchmarks. (Link1 and Link2)
- **Security/Guardrails:** *Anthropic* emphasizes strong guardrails for safer outputs, suitable for sensitive applications.
- **Long Context Support:** *Gemini* supports up to 10M tokens, ideal for handling lengthy documents or complex workflows.
- **Integration Overhead/Ecosystem Compatibility:** *OpenAI* models integrate seamlessly with Azure workloads, using existing cloud partnerships for easier deployment.
- **Customization Capabilities:** *Amazon Nova* offer cheap and easy custom fine-tuning services, enabling tailored performance for domain-specific tasks.
- **Global Language Support:** *Gemini* models perform best on multi-lingual tasks.

TL & DR from experience

Always start off with **medium-sized models** (50-70B parameters) or **mid-priced proprietary models** like GPT-4o, Gemini 2.0 Flash, and Sonnet 3.5/3.7

You can move up/down as you build

Don't get lost in **model-selection conundrum**

Choosing the right setup / Application format

DATA : Biggest indicator of what kind of application u wanna build . Always start with ~100 sample data

What data to collect :

- **Input** Data : Say customer takes help of a customer support exec
- **Output** Data : Customer support exec gave some solution
- **Understanding reasoning** : Same question if multiple answer try to understand the whys ?

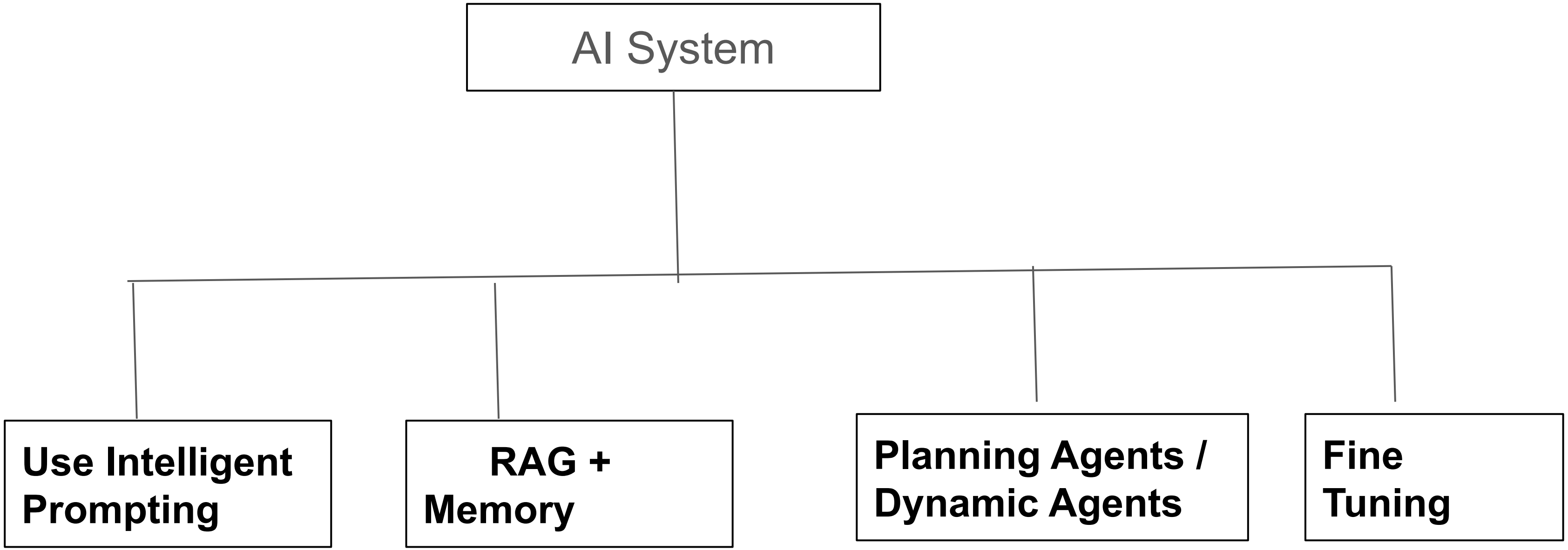
This step is most crucial because data will guide what kind of application design you want to have

Prototype first , optimization later

Design Principles (Step by Step)

1. Prioritize **effort**: Get a working prototype quickly, understand issues
2. Optimize for **performance**: Improve accuracy and reliability.
3. Focus on **cost and latency**: Try cheaper and faster models, now that you understand the problem.

Choosing or designing the right AI Application



Why Prompt engineering is easiest way to build AI systems

Latency: You only incur the latency of the LLM itself, and in most cases, you can precisely estimate it—especially time to first token.

Cost: No additional components are required, making costs predictable since they scale with the number of tokens.

Skill/Effort: Prompt engineering is becoming increasingly automatable, reducing the expertise barrier.

Performance: The only real reason to consider a different approach is if prompt-based methods don't meet your performance needs.

Prompt Engineering

- **Skill Based Prompting** 2023 and Early 2024 Users need to acquire prompt engineering techniques like **Chain of Thought** and **ReAct** to guide the model effectively.

- **Automated Prompt Optimization** Late 2024 Application or Model developers add an optimization layer that refines and enhances prompts before they reach the model.

- **Self-Optimizing Models** >2025 The model automatically recognizes the nature of the query (e.g., reasoning tasks) and selects the best prompting strategy internally, without user intervention.

Skill based prompting - Simple rules

- **Zero-Shot:** Ask LLM to answer without examples
- **Few-Shot:** Including examples in your prompt reliably guides the model.
- **Role-Based Prompts:** Define a role (e.g., "You are a...") to set context.
- **Keep It Simple:** Use clear, straightforward English.
- **Comprehensive Explanation:** Share all relevant information to reduce ambiguity.
- **Natural Tone:** Instruct the model to respond as it would in everyday conversation (tonality control)

Skill based prompting - Complex rules

1. Chain of Thought (CoT)

- **Concept:** Breaks down complex problems into a series of logical steps.
- **How it Works:** Instruct the LLM to "think step by step" to show its reasoning.
- **Example:** For a math problem, the LLM will calculate intermediate steps before the final answer.
- **Use When:** The query requires complex reasoning or multi-step logic.

2. Decomposition Prompting

- **Concept:** Divides a large task into smaller, solvable sub-tasks.
- **How it Works:** You prompt the LLM to complete one sub-task at a time, then combine the results.
- **Example:** To write a full report, first ask the LLM for an outline, then for content for each section, and finally, for a combined draft.
- **Use When:** The task is too big or complex for a single prompt.

3. Ensembling Prompting

- **Concept:** Generates multiple answers to a single query and selects the best one.
- **How it Works:** Use different prompts or models to get varied responses. A "judge" model or human then picks the most accurate answer.
- **Example:** Get three different explanations for "black holes" and choose the clearest one.
- **Use When:** You need highly reliable and accurate answers, as it reduces the risk of a single bad response.

Automated prompting - Meta Prompting

Instead of requiring humans to learn and apply optimization techniques, what if you could integrate these techniques directly into a **model** and train it to generate useful prompts automatically?

Meta prompts are specifically designed to help the model generate prompts by leveraging best practices in prompt generation.

For example:

- https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/prompt-generator
- https://platform.openai.com/docs/advanced-usage

Automated prompting - DSPy

DSPy: *Programming*—not prompting—Foundation Models

Documentation: DSPy Docs

downloads/month 1M

DSPy is the framework for *programming—rather than prompting—language models*. It allows you to iterate fast on **building modular AI systems** and offers algorithms for **optimizing their prompts and weights**, whether you're building simple classifiers, sophisticated RAG pipelines, or Agent loops.

Link

Prompt based agents

LLM calling tools based on user prompts - rule based (?)

link link link

RAG

Fun fact

What's the coolest thing about AI Agents ?

- You can talk to them in natural language

What's the most painful thing about agents?

- They can take as much information as their **context length** (in fact much lower - needle in haystack funda) and they are stateless i.e they **forget** beyond their training data (ok …they used to forget a lot now it is getting better)

What are the implications and what can be done?

Implications :

- **Memory:** Any improvements or feedback provided through the prompt are short-lived and don't support continuous learning.
- **External Context:** In enterprise settings, there's GBs of data to learn from

Solutions:

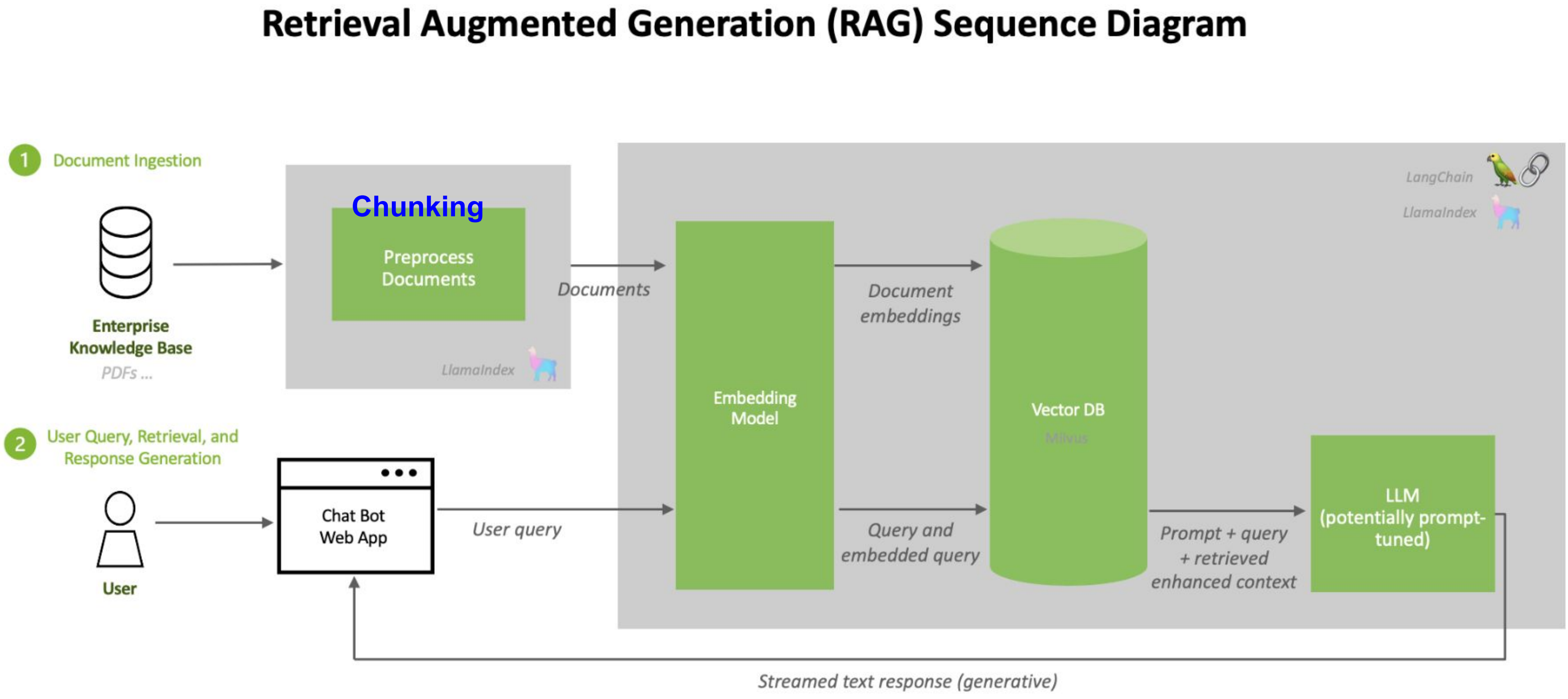
How can we enrich agents with contextual information as needed to:

- Provide enterprise-specific context
- Store and update memory of past conversations
- Dynamically incorporate relevant information from large datasets that don't fit within the LLM's context length

This is exactly where **Retrieval-Augmented Generation (RAG)** plays a crucial role.

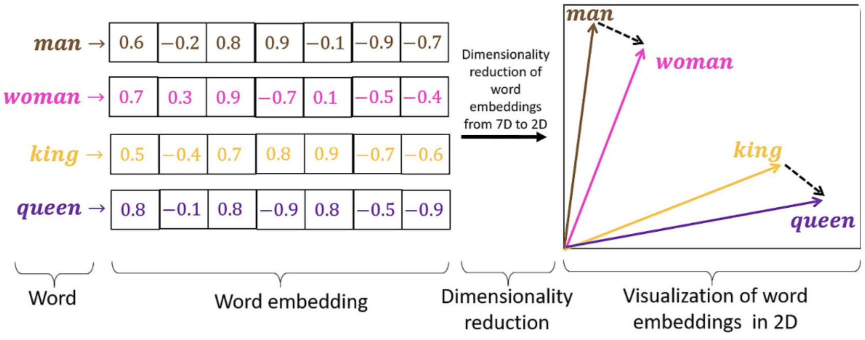
Enter RAG

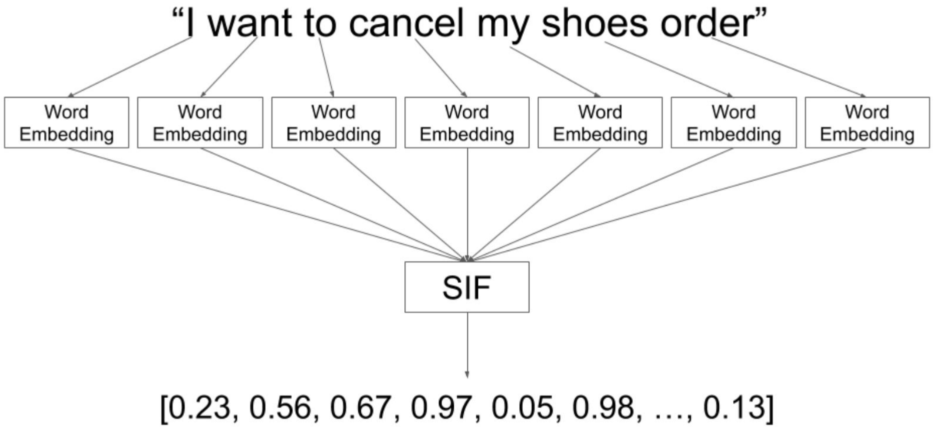
RAG is a technique that helps process large volume of data by efficiently retrieving and optimizing the most relevant information before passing it to LLM



[link]

What is Embedding ?





Word embedding represents a fundamental technique that transforms words into dense numerical vectors within high-dimensional space, where geometric relationships reflect semantic similarities between corresponding terms

Sentence embeddings extend the vector representation paradigm **from individual words** to encoding entire sentences, paragraphs, or documents
The fundamental distinction between sentence embedding vs word embedding lies in their scope and **contextual integration.**

Commonly used Embeddings:
General RAG → OpenAI (`text-embedding-3-large`) or `all-MiniLM-L6-v2`.

Cost-sensitive → `all-MiniLM-L6-v2`.

Accuracy-critical → `e5-large` or OpenAI `3-large`.

Multilingual → Cohere or LaBSE.

Domain-specific → BioBERT, FinBERT, LegalBERT.

Link

Calculating Similarity

Semantic similarity or **Vector Similarity** measures how close two pieces of text (words, phrases, or sentences) are in meaning by calculating the distance between their embeddings (vector representations).

- High semantic similarity: The two texts have nearly the same meaning.
- Low semantic similarity: The texts are very different in meaning.

For example: semantic_similarity(vector of sentence 1, vector of sentence 2) = 0.58

- 0.0 → No similarity (completely different meanings)
- 0.5 → Somewhat related (partial overlap in meaning)
- 1.0 → Identical meaning (very close in context)

Vector DB

A vector database is a specialized database designed to store, index, and query high-dimensional data as **vectors**, or numerical representations called **embeddings**

Feature	Traditional Relational Databases (e.g., MySQL, PostgreSQL)	Vector Databases (e.g., Pinecone, Weaviate)
Data Structure	Organized in tables with rows and columns, enforcing a strict schema.	Stores data as high-dimensional vectors (arrays of numbers).
Data Type	Optimized for structured data.	Designed for unstructured and semi-structured data.
Query Mechanism	Uses SQL for exact matches, joins, and filters based on predefined criteria.	Uses similarity search (vector search) to find data points with a similar meaning.
Search Type	Keyword-based search; finds exact matches.	Semantic or vector similarity search; finds data based on contextual meaning.
Primary Use Case	Transactional systems, data analysis, and applications requiring data integrity.	AI/ML applications, semantic search, recommendation engines, and chatbots.

RAG Evaluation

Retrieval Metrics (Assess how well relevant documents are retrieved)

- **Context Recall** – Measures how often the correct document appears in the top K retrieved results.
- **Context Precision** – Measures the proportion of relevant documents among the top K retrieved.
- **MRR (Mean Reciprocal Rank)** – Evaluates how high the first relevant document appears in the ranked list.
- **NDCG (Normalized Discounted Cumulative Gain)** – Weighs the relevance of retrieved documents, giving more importance to higher-ranked ones.

Generation Metrics (Can be semantic match or LLM Judges):

- **Faithfulness /Hallucination Rate** – Measures how well the generated response aligns with retrieved documents, avoiding hallucinations (Also called hallucination score)
- **Relevance** – Evaluates how useful and contextually appropriate the response is.
- **Fluency** – Checks whether the output is coherent and well-structured.
- **Factuality** – Ensures the generated response is factually accurate.

Link and Link

Enterprise RAG Design considerations

Decision Factors

- Document Parsing problem ?
- How should documents be chunked?
- What models can I use for generating vectors?
- How to setup my retrieval pipeline?

Retrieval Problems:

- What if chunks retrieved do not contain the right answers?

Generation Problems:

- What if the model cannot access information from context?

Optimization:

- Cost/Latency
- Performance

Document Parsing related problems

Traditional OCR Functions

- Amazon Textract – Ideal for structured documents.
- EasyOCR – Open-source and supports multiple languages.
- Docling – A powerful open-source document parsing tool.

Handling Images and Charts: Use Multimodal Embeddings (should be handled separately)

When PDFs are Clumsy: Use Multimodal Models

- Models like GPT-4o can directly process images, extract text, and even interpret tables, charts, and handwriting.
- These models understand context, making them useful for messy or unconventional document formats.
- Latest, Mistral: https://mistral.ai/news/mistral-ocr
- Use with caution: https://huggingface.co/spaces/echo840/ocrbench-leaderboard

Chunking related Issues

Different Chunking Strategies	Best for	How it works
Fixed-size Chunking	Simple documents, quick prototyping	Breaks text into chunks of a predefined size, often with a slight overlap to maintain context.
Sentence Splitting	Documents where sentence boundaries are clear	Splits text into chunks based on sentence endings (periods, question marks, etc.).
Recursive Splitting	Complex documents with varying structures	Iteratively splits text using different delimiters (e.g., paragraphs, then sentences, then words) until chunks are small enough.
Semantic Splitting	Documents where topics change frequently	Uses an LLM to identify changes in topic or meaning and splits the document at these semantic breaks.
Parent-Child Chunking	Maintaining context for questions that need detail	Creates small, detailed chunks for retrieval and pairs them with larger, parent chunks that provide broader context.

Loss of Context: When text is split at arbitrary points, sentences or logical units can be broken, scattering relevant information across multiple chunks.

Irrelevant Information: If chunks are too large, they can contain multiple topics, diluting the main semantic meaning and leading to less precise search results.

Incomplete Answers: The complete answer to a user's question might be spread across several different chunks, and if only one is retrieved, the LLM will provide an incomplete response.

Out-of-Order Chunks: Naive chunking can lead to adjacent chunks being presented to the LLM out of their original order, which causes confusion and can lead to hallucinations.

Popular Vector DB providers and when to use?

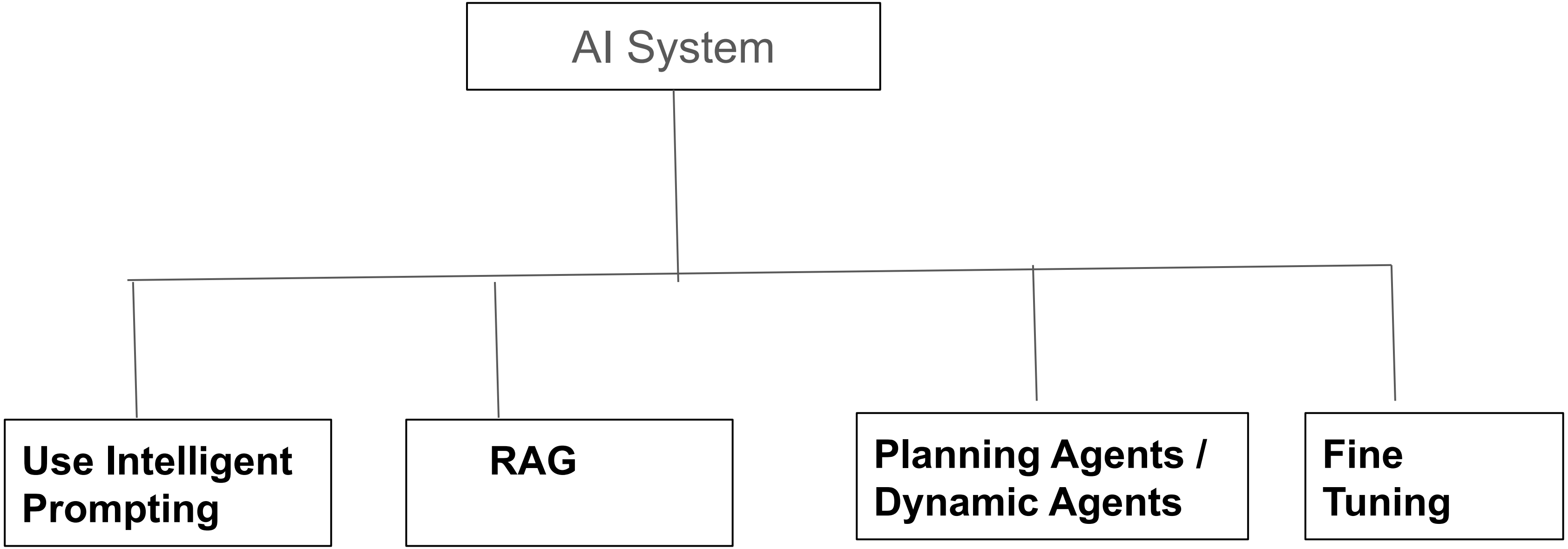
Vector Database	Type/Description	When to Use
ChromaDB	Open-source, in-memory	**Small-scale projects**, **prototyping**, and local development. It's fast and easy to set up, but data doesn't persist.
Pinecone	Fully-managed, cloud-native	**Large-scale, production-grade RAG applications,** especially for enterprises needing high performance and scalability with minimal management overhead.
Weaviate	Open-source (self-hosted or managed)	Projects that require a **production-ready solution with flexibility and control**. It has built-in vectorization and strong metadata filtering capabilities.
Qdrant	Open-source, built in Rust	High-performance, scalable, self-hosted solutions where developers need granular control, advanced filtering, and cost efficiency.
Milvus	Open-source, cloud-native	**Applications with massive datasets** (billions of vectors) and requirements for extreme performance and real-time similarity search.

Choice of Search algorithm

Algorithm/Method	Description	Key Features & Use Case	When to Use
BM25	A keyword-based ranking function that scores documents based on the frequency of query terms, their rarity across the document corpus (Inverse Document Frequency or IDF), and document length.	✅ Efficient for exact keyword matches and well-defined queries. ❌ Lacks semantic understanding; can fail with synonyms or paraphrased queries.	When the user query contains specific, well-defined keywords or entities, anc the documents are less conceptually complex. For example, searching for a specific product ID, a person's name, or a technical term in a structured knowledge base.
TF-IDF	A statistical measure that evaluates the importance of a word in a document relative to a corpus. It's a precursor to BM25.	✅ Simple and effective for basic keyword retrieval. ❌ Similar to BM25, it does not capture semantic meaning and can be less effective than BM25 in some cases.	In scenarios where simplicity and computational efficiency are the top prioritie and the dataset is small or the queries are strictly keyword-based.
Cosine Similarity	Measures the cosine of the angle between two non-zero vectors. A score close to 1 indicates high similarity, and a score close to 0 indicates low similarity.	✅ Excellent for semantic search with vector embeddings, as it's not affected by vector magnitude. ❌ Requires pre-processing to create vector embeddings.	When the user query is more conversational or conceptual, and you need to find documents with similar meaning, even if they don't share keywords. For example, finding articles about "climate change mitigation strategies" when th query is "how to reduce global warming".
Dot Product	A measure of vector similarity that takes both the angle and the magnitude (length) of the vectors into account.	✅ Can be faster to compute than cosine similarity. ❌ Highly influenced by vector length, which can skew results if embeddings are not normalized.	In specialized applications where vector embeddings are normalized or wher the magnitude of the vector is a meaningful feature, such as in certain recommendation systems or when speed is critical.
Hybrid Search	Combines a keyword-based method (like BM25) and a vector-based method (like Cosine Similarity) to create a single, combined score.	✅ Balances precision (from keywords) and recall (from semantic meaning). ❌ More complex to implement and tune than a single method.	As a general, robust solution for most RAG applications. Use it when queries can be a mix of specific keywords and broad, conceptual questions to ensure high-quality retrieval.
Re-ranking	An additional step after the initial retrieval to re-order the top documents. A separate model (often a cross-encoder) scores the query-document pairs to produce a more refined ranking.	✅ Significantly improves the relevance of the retrieved documents. ❌ Adds latency and computational cost to the retrieval process.	When the initial retrieval set is large or contains many potentially relevant but not perfect documents. Use it to refine the results of a hybrid or semantic search to ensure the best documents are presented to the LLM.

80% problems in RAG development are in document parsing and retrieval

Choosing or designing the right AI Application



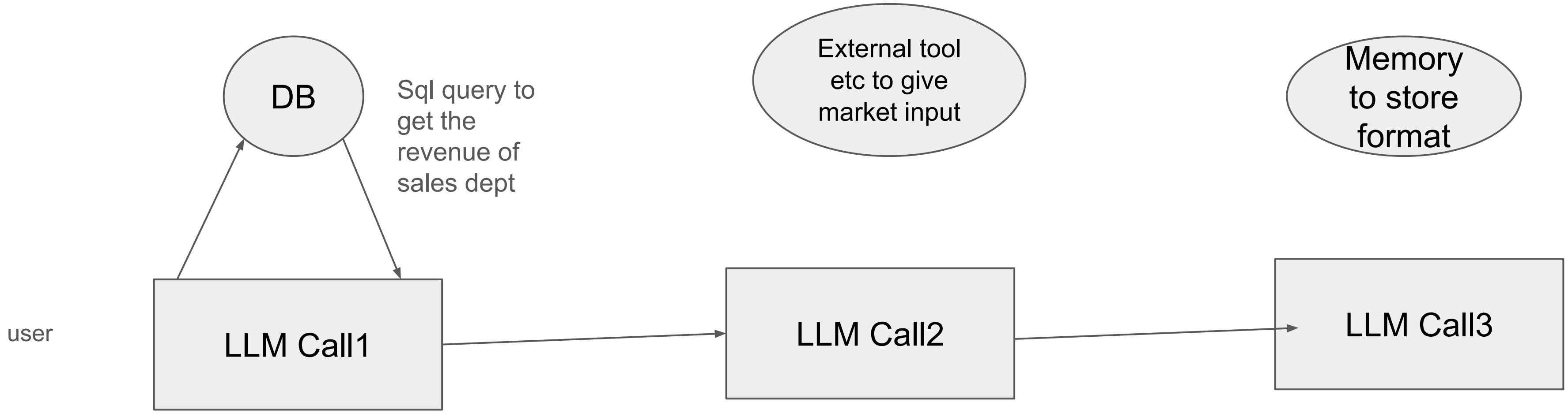
Difference between Prompt based agents vs Dynamic Agents

In **Prompt based workflow agents** , AI models focus on understanding and generating content, while planning and orchestration are typically handled by humans (Through pre-defined code). (Only Action Autonomy)

The key factor that sets **Dynamic or planning agents** apart from other agents—is the kind of planning autonomy they exhibit. (Action Autonomy + Planning Autonomy)

Prompt Based Agents - Example

GOAL : Generate a quarterly performance report for our sales department focussing on revenue growth



Get Revenue from Sales DB for JAS’25

Dissect the revenue data and find out hits and misses - compare with L2Y

Use all these insights and structured format to publish report

Autonomous / Planning Agents

Generate a Quarterly report for the sales dept focussing on revenue growth

LLM

I need to do these :

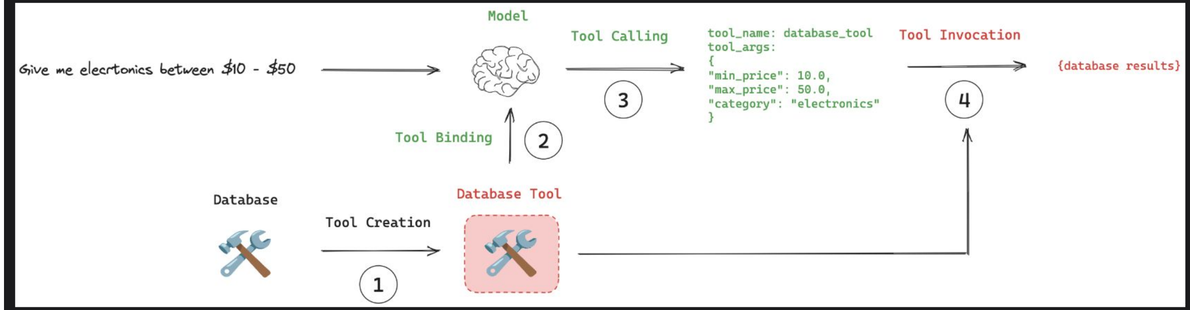
1) Get this years data from DB
2) Compare with L2Y and with market data to gather insights
3) Publish all these in a format that i used last time

DB, External tool , Memory etc

How Tool calling works

Key concepts

1. **Tool Creation:** Use the @tool decorator to create a tool. A tool is an association between a function and its schema.
2. **Tool Binding:** The tool needs to be connected to a model that supports tool calling. This gives the model awareness of the tool and the associated input schema required by the tool.
3. **Tool Calling:** When appropriate, the model can decide to call a tool and ensure its response conforms to the tool's input schema.
4. **Tool Execution:** The tool can be executed using the arguments provided by the model.



Tool calling steps:
- Agent is given a JSON description of tools.
- Agent **decides** the right tool to call with arguments - doesn't call the tool.
- **User executes the tool** and returns the result.
- Repeat above two steps until the plan is completed.
- Agent finally returns the answer.

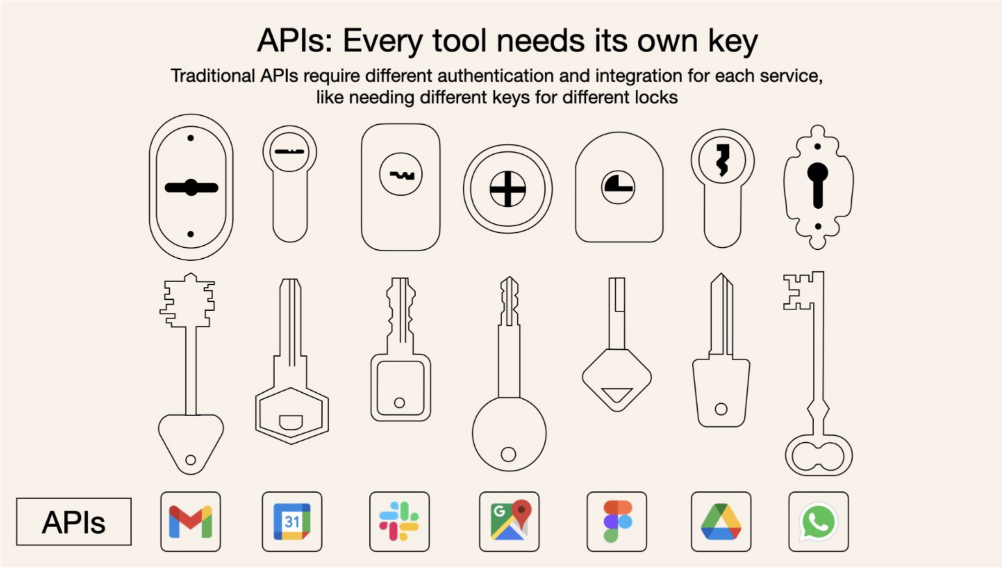
Tools can provide both read/write functionality. E.g. Check for flights, book a flight.

Link

Problem with tool calling

Every LLM provider and every tool has different custom glueing

For M LLM provider and N tools possible combinations are like MxN - there is no standardization and very difficult to scale



Enter MCP

[Link]

What do you mean by Agentic Frameworks ?

- **Langchain** (misses <u>orchestration</u> - it is not suited for agentic application ; it only able to support applications with sequential /chain like LLM calls. No dynamic / planning agents can be written using Langchain)

- **Langgraph** (Agents and tools are nodes connected via graphs ; inspired by statemachine) . **<u>Coding heavy</u>** ; its extremely flexible and low leve agentic framework

- **Autogen** (They focus a lot on different types of multi agent conversations - 1x1 , group , sequential , nested) Very strong orchestration ; very resilient in enterprise setup **(Mid Code) debate heavy agents**

- **CrewAI** (Team philosophy , good for multi agent workflows ; each team member is an agent. They have specialized roles assigned and they either collaborate among themselves or they talk to their manager) - **<u>Low Code</u>**

- **Agno (Phidata)** - very similar to crewAI ; it is like writing a automation script (lagging behind , but like crewai user friendly)

All these frameworks abstracts the process of building , orchestrating and managing AI Agents doable

- State management , memory management , Communication protocol between agents
-

Agenda :

Goal : Knowing and learning Agentic Frameworks and building Agentic App

- Langflow (Low no code)
- Langgraph
- CrewAI
- Flowwise / n8n (this is not this week)

Github link : https://github.com/agenticgogol/Classcode_B2

Question Link: Link

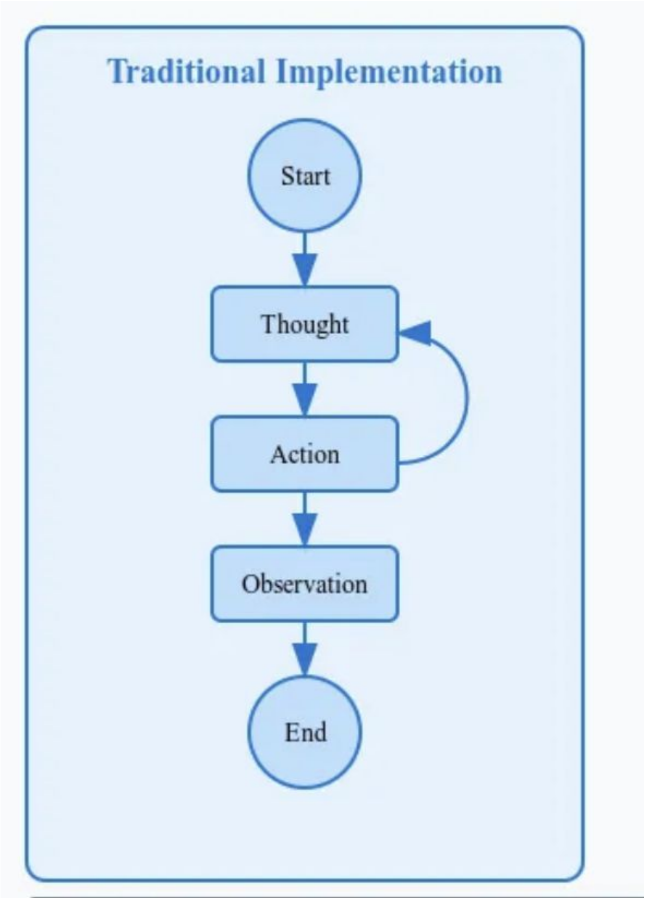
Langflow (No / Low Code) installation

1. Create a directory in your local system and go there from terminal
2. Create a virtual env and activate the same
   - uv venv --python 3.11.8
   - uv python pin 3.11.8
   - source .venv/bin/activate
3. Install Langflow
   - uv pip install langflow==v1.2.0
4. Run Langflow
   - python -m langflow run
5. Open the langflow on browser

Langgraph

1. Nodes , edges , graph that we have to build
2. Nodes - functions , agents (with LLM)
3. Edges - fixed edges , conditional edges
4. Tool definition custom (with descr string) and tool binding with LLM
5. How to call standard built in langchain tools
6. Multiple tools that are binded with LLM (LLM based on question figures out how to call the right tool) - description string

ReACt Agentic Pattern (Think - Act -> Observe in loop)



ReAcT pattern is especially helpful :
When in order to achieve a business goal , we have to call multiple tools and very often outcome of one tool is input to the other ones

Multi step processes - we cannot just call one tool and finish …based on the output of one tool , LLM decides what is the next action to be taken

Agentic RAG

1. Just like what we saw when there are multiple tools , bind with LLM - the agent at the runtime figures out which tool to call based on user query

2. Similarly , in Agentic RAG we can have multiple external data sources bind with LLMs - the agent at the runtime figures out which tool to call based on user query.

Ideally show the code for basic rag —> agentic rag

Basic rag theory discussed

Agenda D3 (in the process of covering D2)

1. Basic RAG
2. Multimodal RAG
3. Agentic RAG with multiagents
4. Agentic RAG with multiple DBs
5. Adaptive RAG
6. RAG Eval
7. RAG Deployment

Basic RAG Steps :

Within MultiModalRAG Class :

__init__

1. Gemini Setup (Which LLM to use)
2. Embedding Initialization (Which Embedding to use)
3. ChromaDB Initialization (vectorDB) (Which VectorDB to use + Collection)
4. Text Splitter (Chunking) (Which Splitting mechanism to use)
5. Session Memory

Basic RAG Steps :

Within MultiModalRAG Class :

Extract_text_from_documents - We will extracting text from PDF, PPT , TXT DOCX all these formats

Process image with gemini : Image data to Gemini Vision for OCR and content understanding

Add Documents to Knowledge Base (vectorDB) :

- We will call Extract_text_from_documents and get the text
- Split text into chunks
- Generate embedding for each chunks
- Store embedding or vector representation vectordb

Basic RAG Steps :

Within MultiModalRAG Class :

Search Knowledge Base :

- Converts user query into embedding
- Searches text and image collection separately
- Combines the results and sorts by importance and returns top k documents

Generate Answer :

- Use retrieved document from above function as context
- Prompts the LLM to generate a comprehensive answer
- Retruns the answer with source document name

Get Session info:

- Returns details of all uploaded documents in the current session
- Tells us chunk count for all the documents

Basic RAG Steps :

Outside of MultiModalRAG Class :

Initialize system:

- Initializes the multimodal class and passes the LLM Keys

Gradio / UI Related coding:

Agentic RAG - Multi DB

This Agentic RAG System handles Multiple Vector Databases each specialized for specific domain / topic. Few components :

1. Gemini LLM : Used for reasoning and generating answers
2. OpenAI Embedding : for semantic search across all the databases
3. Chroma DB : store embeddings
4. VectorDBRegistry: Dynamically chooses the most relevant DB for a given query
5. Gradio UI : web interface

Agentic RAG - Multi DB

1. Class VectorDBRegistry:
- Manages multiple vector databases and dynamically selects DB for a query
- register_db()
  - Registers a database with its domain keywords
  - Computes the embedding for keywords to represent DB topic vectors
- choose_db(query)
  - Choose top -K DBs based on cosine similarity with query embedding
  - (When user asks a query that query is embedded and compared with the topic vectors To identify which database RAG should refer to)
- get_db

Agentic RAG - Multi DB

1. Class AgenticRAGMultipleDB:
- __init__ (initializes LLM , Embeddings, VectorDBRegistry, TextSplitter)
- register_chroma_db (creates a chroma db instance and registers it with the keywords in the registry. It enables dynamic selection based on query)
- **Ingest_to_db** (get text , split text, embed chunks, metadata, stored)
- **Retrive_for_query** (compute the query embedding , call the vectordDBregistery class to find relevant database and within the database relevant chunks , pass the retrieved context to Gemini LLM for answer generation)
- Ask_query (inside this function we call **Retrive_for_query to get the context and then call the LLM directly)**

Agentic RAG - Multi DB (Implementation approach-1)

Ingestion :

- We will upload documents
- Based on the document name : we will convert the document name to embedding
- We will map the domain keywords mapped to database into embedding
- We will compare the domain keyword embedding with filename to decide which DB the document should be ingested

Retrieval :

- User asks a query . That query is converted to embedding
- That query embedding is matched to domain keyword embedding to finalize which DB to call
- Then within that DB again a semantic search happens between query embedding and vector store
- Retrieved context is pushed to LLM for answer generation

Agentic RAG - Multi DB (Implementation approach-2)

Ingestion :

- We are getting documents related to topic1 and related to topic2
- Separately all the documents are converted to vectors and stored in 2 different databases after going through chunking and embedding
- After creating the database we are adding a descriptions for each of the databases

Agent Code :

- Inside this we are instantiating an agent (LLM + Tools) and we are binding the agent with tools (in this case the vector databases)
- Remember , the agent as expected has access to query the LLM

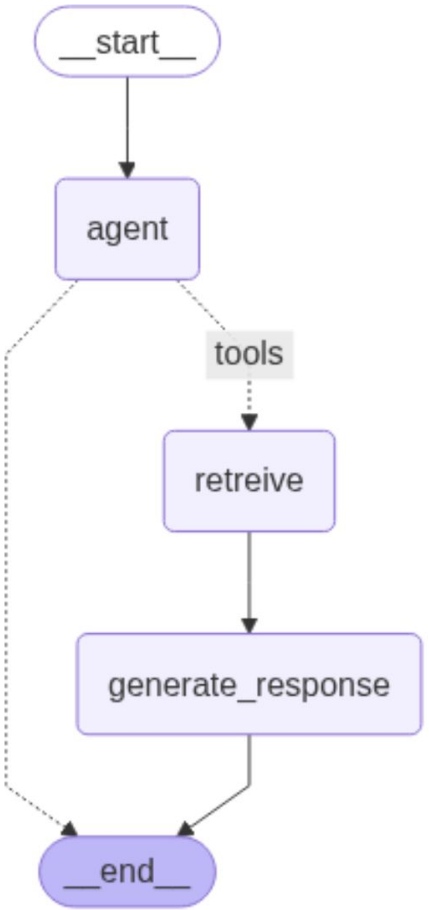
Agentic RAG - Multi DB (Implementation approach-2)

Function generate_response :

- Context + query + prompt to send it to LLM
- Inside this I have used prompt = hub.pull("rlm/rag-prompt")

Langgraph Workflow:

- Agent decides which tool to call or it directly says I don't know
- Agent sends the user query and the VectorDB descriptions - both to LLM
- LLM tells us which DB to use
- And the answer from that DB is then again passed to LLM and answer is generated



Difference between two approaches

1. Matching of Query to DB is done inside our system (match the embeddings ourselves) **[more control)**

2. Whereas when we bind the vectordb + descriptions with the LLM in Langgraph this choice of which DB to use is done by the LLM **(more autonomy)**

Multi Agent RAG

Multi Agent Workflow:

- Researcher Agent : Searches the knowledge base i.e Vector DB for relevant documents and from that calls LLM to extract key facts and gaps
- Synthesis Agent : Combines the research findings into coherent answer (takes help of LLM)
- Fact Checker Agent : Validates the synthesized answer for consistency and accuracy (again takes help from LLM)
- Follow-up Agent : Generates contextual follow up question for further exploration to the user
- Coordinator Agent : Orchestrates all agents and formats the final answer

RAG Evals

Retriever Evaluation (top K semantic similar documents that are retrieved) :

Recall@K: % of queries where at least one relevant document is in the top K retrieved result (You ask 100 questions to a RAG and 70 of them in the top 3 document you have a relevant document … the recall is 70%)
Precision@K: % of top K documents that are relevant (you have 3 retrieved docs for a given question - and 3 of them are relevant to the question - P = 100%)

- **Manual labels of query to relevant document**
- Synthetic query and relevant document generation (you give LLM the responsibility to label)

Generator Evaluation :

- **Faithfulness :** Does the answer of generation is using the retrieved data (faithful to retrieval)
- **Completeness ;** Did the answe cover all the relevant points
- **Factual correctness :** is the answer factually correct
- **Grammatical / language correctness :** is the language similar to how human talk

\- **LLM as a judge** to check the faithfull , correctness and language coherence (use more powerful LLM that that is used for generation)

\- Cosine Similarity or semantic similarity between retrieved and generated documents

<HUMAN EVALUATION> in RAG is very important for end to end performance measurement
However, industry is now relying more on LLM based automated evaluation