Introduction to Generative AI and Agentic AI

Generative AI: is a category of AI where the objective is to generate the new content (i.e.,

text, images, audio, video, etc.)

Text Models: GPT, Llama, Gemini, Claude

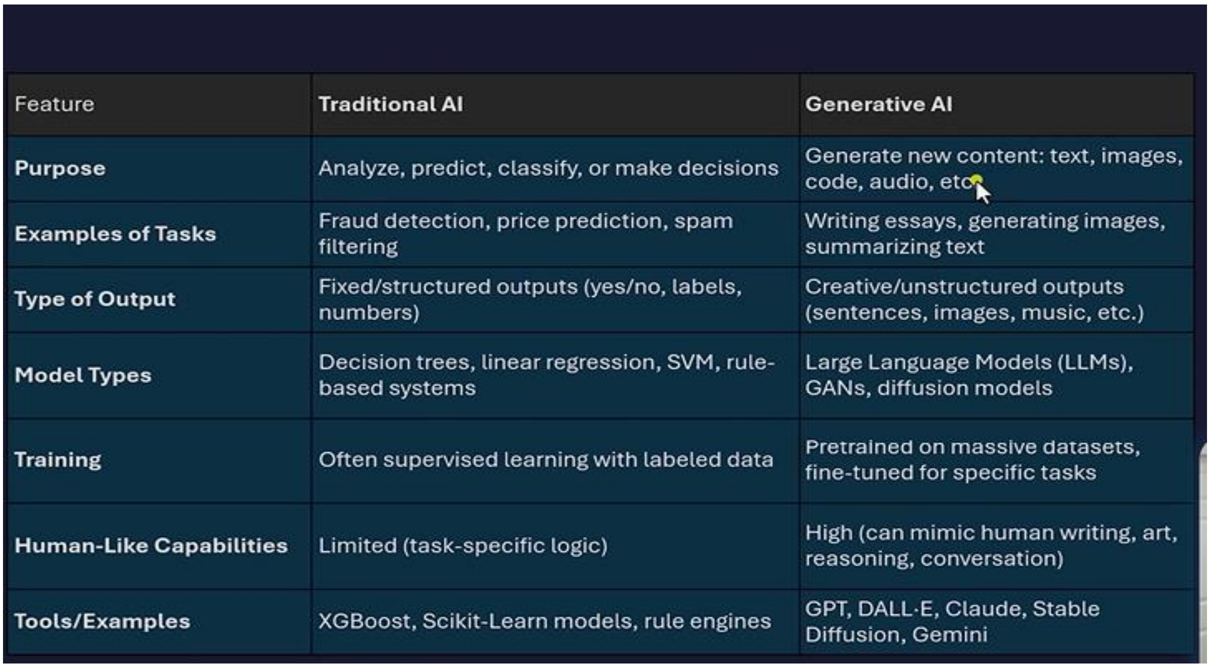
Image Models: DALL-E, Stable Diffusion,

Audio Models: Audio Gen, MusicLM

Video Models: SORA, FLOW

Non generative Ai: if you are not providing any novel content and making application on the existing data e.g.: Image classification, Spam Classification, Price prediction

Traditional AI vs Generative AI:



Feature	Traditional AI	Generative AI
Purpose	Analyze, predict, classify, or make decisions	Generate new content: text, images, code, audio, etc
Examples of Tasks	Fraud detection, price prediction, spam filtering	Writing essays, generating images, summarizing text
Type of Output	Fixed/structured outputs (yes/no, labels, numbers)	Creative/unstructured outputs (sentences, images, music, etc.)
Model Types	Decision trees, linear regression, SVM, rule-based systems	Large Language Models (LLMs), GANs, diffusion models
Training	Often supervised learning with labeled data	Pretrained on massive datasets, fine-tuned for specific tasks
Human-Like Capabilities	Limited (task-specific logic)	High (can mimic human writing, art, reasoning, conversation)
Tools/Examples	XGBoost, Scikit-Learn models, rule engines	GPT, DALL·E, Claude, Stable Diffusion, Gemini

What are AI Agents and Agentic AI?

AI systems Built Using LLMs:

Work Flows:

RAG Systems

Tool Augmented Systems

Agents: Ai agent can perceive its environment, make decisions, and take actions to achieve

Specific goals

Goal Oriented Planning [goal related tools] →

Multi step Reasoning →

Autonomous Decision Making

Related Tools, Knowledge and Memory

Example:

Onboard the new intern joining next Monday.

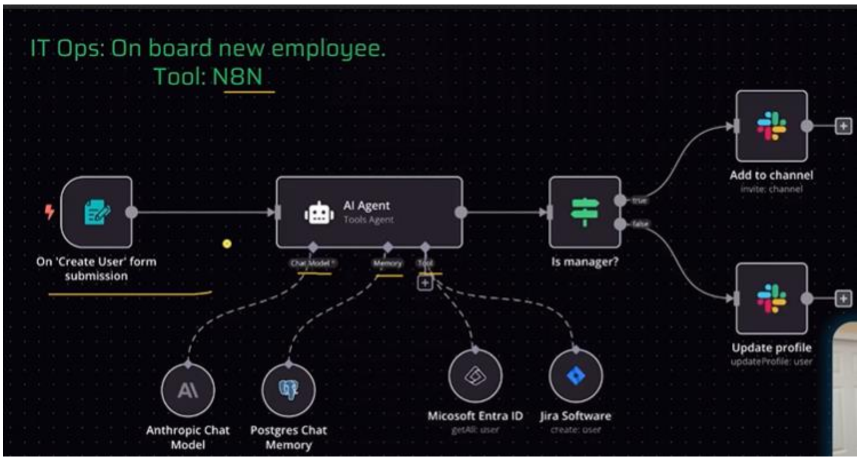
Schedule welcome meeting → LLM

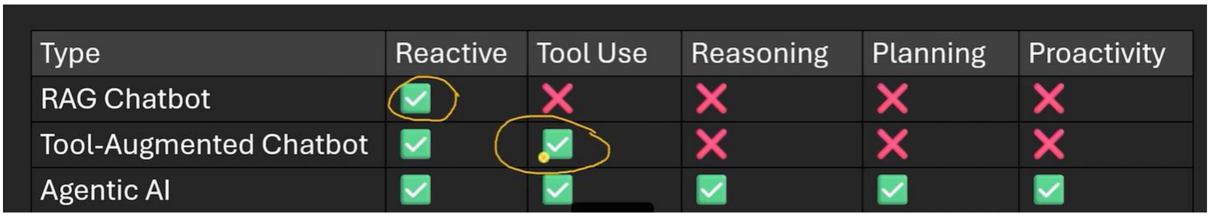
Create intern's profile in HR Management System

IT Helpdesk (Wi-Fi credentials, Email, Slack access)

Order Laptop, ID Card

Agentic AI: refers to an AI system having one or more advanced agents that operate with autonomy, reason through complex tasks, and proactively take multi-step actions to accomplish goals - without needing detailed instructions.





Type	Reactive	Tool Use	Reasoning	Planning	Proactivity
RAG Chatbot	✓	✗	✗	✗	✗
Tool-Augmented Chatbot	✓	✓	✗	✗	✗
Agentic AI	✓	✓	✓	✓	✓

Feature	RAG Chatbot	Tool-Augmented Chatbot	AI Agent	Agentic AI
Knowledge source	External documents (retrieval)	External tools, APIs	Any data/environment	Mixed (tools, memory, world models)
Task scope	Answering based on facts	Solving tasks using tools	Executing actions	Planning, decision-making, autonomous action
Autonomy	Low	Medium	Medium to High	High
Memory/Planning	Stateless	Sometimes	Limited	Strong (may include memory, planning stack)
Initiative	Reactive	Slightly proactive	Proactive (if goal given)	Fully proactive (can self-initiate)
Examples	Document Q&A bot	ChatGPT with plugins	Email organizer AI	AutoGPT, Devin, personal AI executive

TL; DR:

- **RAG chatbot** = Smart Q&A using documents.
- **Tool-augmented chatbot** = Chatbot that can use tools.
- **AI agent** = Acts to achieve tasks given to it.
- **Agentic AI** = Advanced AI that *thinks and acts* over time with autonomy and memory.

Takeaways

- Systems built using LLMs can be divided into two categories (1) Workflows (2) Agents.
- Workflows use LLM but do not have autonomy and complex reasoning.
- AI agents are autonomous software systems that perceive their environment, make decisions, and act to achieve specific goals with minimal human intervention.
- Agentic AI refers to advanced AI systems capable of reasoning, planning, and executing complex, multi-step tasks independently, often coordinating with other agents.
- Agentic AI systems will have one or more AI agents in it performing complex tasks.

Gen AI vs AI Agents vs Agentic AI:

Feature	Gen AI	AI Agent	Agentic AI
Goal	Generate content	Perform tasks	Achieve goals autonomously
Core Function	Predict and complete patterns	Sense → Decide → Act	Plan → Adapt → Execute (repeat)
Initiative	Reactive	Mostly reactive (unless event-driven)	Proactive, self-directed
Memory	Stateless (usually)	Short-term memory	Long-term memory & learning
Autonomy Level	Low	Medium	High
Time Horizon	One-shot responses	Single or short task cycles	Multi-step, long-term planning
Tech Stack	LLMs, diffusion models	LLMs + APIs + tools	LLMs + memory + tools + planners + agents
Analogy	Artist/writer	Assistant or tool-user	Project manager / autonomous co-worker

□ TL; DR:

- **Gen AI**: Generates text, images, etc. from prompts.
- **AI Agents**: Use Gen AI (and other tools) to complete user-defined tasks.
- **Agentic AI**: Autonomous systems that *think*, *plan*, and *act* toward complex goals.

Real-world Applications for Gen AI & Agentic AI:

AI coder → PyCharm inbuilt, copilot

Travel Assistant

Content Generation → chat gpt

Creative Design → DALL-e, SORA, photo room etc

Scientific Research → googles AlphaFolds

Work Flow Automation

Digital Personalities → HeyGen,11 Eleven Labs

Steps to Build Gen AI and Agentic Applications:

1. Do we really have the need of Gen AI [can you solve this problem without LLM]
   Traditional coding, Rules, Statistical Ml
   Don't cut with sword if it can be done by a knife

2. Data collection and Prep
   Create a proper Pipeline
3. Choose Right Architecture:

Commercial:

GPT: Most Expensive, Best Performance

Claude: Less Expensive, Good Performance

Open Source:

LlaMa: Free, Reasonable performance

Mistral: Free, Reasonable Performance

Based On the problem figure out what kind of the architecture we need to build

Are you building Work Flow or Agent

4. Model Training and Fine tuning:

   Use LLM as is

   Finetune LLM: Regular Fine tuning, Reinforcement fine Tunning

   RAG LLM

   Tool Augmented AI system

   Agentic AI

5. Evaluation:

   BLEU, ROUGE Score

6. Optimization and Deploy:

   Quantization, Knowledge Distillation

   AWS, Azure

7. Compliance and Ethics:

   Your application should Meet all the expected compliances and ethics

8. Monitor and Feedback:

   LLM observability

Gen AI Key Concepts and Foundation

What are Large Language Models: Large Language Models (LLMs) are deep learning models trained on massive amounts of text data to understand, generate, and manipulate human language.

🔧 **How They Work:**

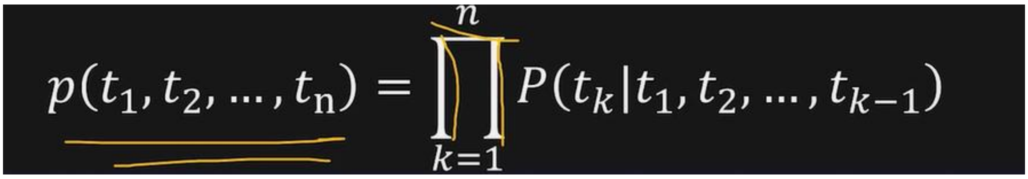
- Based on **transformer architecture** (like GPT, BERT, LLaMA).
- Trained using **unsupervised or self-supervised learning** — predicting the next word in a sentence.
- Once trained, they can:
  - Generate essays, code, emails, poems, etc.
  - Translate between languages.
  - Answer questions and summarize documents.
  - Reason and solve logic/math problems.

Feature	**Description**
“Large”	Refers to the **billions (or trillions) of parameters** (neural weights).
“Language”	Trained on text from books, websites, articles, etc.
“Model”	A machine learning system that learns and generalizes from data.
Architecture	Almost always based on the **Transformer** architecture.
General-purpose	Can be used for many NLP tasks (translation, summarization, Q&A, etc.).

Model	**Developer**	**Highlights**
GPT-4	OpenAI	Powers ChatGPT; strong general reasoning.
Claude	Anthropic	Safety-aligned, conversationally fluent.
Gemini	Google DeepMind	Multimodal capabilities.
LLaMA 3	Meta	Open-source research-grade model.
Mistral	Mistral AI	Lightweight, open-source, performant.

How LLM Works:

Uses Auto Generative Regressor: Predicting the next word and add previous to predict the next



$$\underline{\underline{p(t_1, t_2, \ldots, t_n)}} = \prod_{k=1}^{n} P(t_k | t_1, t_2, \ldots, t_{k-1})$$

Transformers:

Encoder

Decoder

System 1 Thinking:

Fast, Intuitive, automatic thinking

Examples:

Recognize Faces

Drive on a known route

System 2 Thinking:

Slow, analytical, and effortful

Example:

Getting married to a person

Building a business strategy

LLM are only capable of doing system 1 thinking

Pattern recognition and its fast and intuitive

Context Window, Temperature, Top-p and Top-k:

Context Window:

The Tokens[words] in your query is called context window [suppose 7 words in the query]

If you ask a follow up question it takes the previous as the context window [here the answer is 512 tokens and + your current query token = current window]

Window Length for

GPT-4o 128k Tokens

Llama 2 70b 8k

Gemini 1.5 pro 2M Tokens

If you over the limit of tokens then it takes latest context and backdown of this is if you ask any thing related to the discard token it won’t understand it properly

Use summarises for the generated answer use that further

Temperature:

Temperature Controls how random or creative the model’s output will be

0-2 is range

Top-p(nucleus) sampling:

📌 **What it is:**

Top-p (or **nucleus**) sampling selects the **smallest set of tokens whose cumulative probability ≥ p**, and then samples from that dynamic shortlist.

🔧 **How it works:**

- Sort tokens by probability (like in top-k).
- Include tokens until their cumulative probability **≥ p** (e.g., p = 0.9).
- Randomly sample from this **adaptive set**.

🎯 **Purpose:**

- More **adaptive and flexible** than top-k.

- Adjusts shortlist size based on the shape of the distribution — big when it's flat, small when it's sharp.

□ Example:

If the top 3 tokens together already make up 90% probability, **only those 3 are used** (even if k=10 would have included more).

Top-k Sampling:

📌 **What it is:**

Top-k sampling picks the next word only from the top k most probable tokens (words/subwords), then randomly selects one based on their probabilities.

🔧 How it works:

- Sort all vocabulary tokens by probability.
- Keep only the top k tokens.
- Normalize their probabilities and sample one at random.

🎯 **Purpose:**

- Restricts sampling to the most likely options.
- Reduces nonsense and out-of-vocabulary weirdness.
- Still allows creativity, but avoids rare junk.

□ TL; DR:

- Top-k: Picks from top k tokens → static cutoff.
- Top-p: Picks from top p% cumulative probability → dynamic cutoff.
- Both help make LLM outputs less boring, more diverse, and less repetitive.

Output length:

Output length of the query

Challenges: Hallucinations, Security and Cost

Why do AI models Hallucinate

Pattern Recognition vs True Understanding

Insufficient Training Data & Lack of Fine Tunning

Incomplete, Ambiguous Prompts

How to Tackle Hallucination

Representative Datasets

Fine Tuning and Validation

Knowledge-Based Systems

Security:

Jailbreak

Sweet talk to jailbreak the models

Cost:

Charge per Token

ROI on gen ai projects

What is a Vector Database?

A **vector database** is a **specialized database** designed to store, index, and search **high-dimensional vectors** (also called **embeddings**), which represent data like **text, images, audio, or video** in a mathematical form.

□ Why Vectors?

- In AI, text or other data is converted into **vectors** using models like BERT, GPT, or CLIP.
- These vectors **capture meaning** (semantics) — so two similar sentences have vectors that are **close** in space.

□ How It Works:

1. **Convert** your data (e.g. documents) into **embeddings** (vectors) using a model.
2. **Store** these vectors in the vector database.
3. When a **query** comes in, convert it into a vector.
4. Use **approximate nearest neighbour (ANN)** search to find vectors closest to the query.

📊 Key Features:

Feature	**Vector Database**
Data Type	High-dimensional vectors (e.g., 768-d)
Query Type	Nearest-neighbour similarity search
Indexing Methods	HNSW, IVF, PQ, etc. (fast ANN algorithms)
Similarity Metrics	Cosine, Euclidean, Dot Product
Scale	Millions to billions of vectors

□ Popular Vector Databases:

Database	Highlights
Pinecone	Fully managed, scalable, RAG-ready
Weaviate	Open-source, supports hybrid (keyword + vector)
FAISS	Facebook’s open-source library, fast local search
Milvus	Scalable, open-source, supports real-time ingestion
Qdrant	Rust-based, open-source, REST and gRPC APIs

🔍 What is RAG (Retrieval-Augmented Generation)?

RAG stands for Retrieval-Augmented Generation, a powerful AI technique that combines external knowledge retrieval with generative models (like GPT or BERT) to produce more accurate, grounded, and up-to-date answers.

⚙️ How RAG Works (Step-by-Step):

1. **User asks a question** (e.g., "What are the side effects of aspirin?")
2. The question is turned into a **vector** using an embedding model.
3. The vector is used to **retrieve** relevant documents (from a **vector database**, like Pinecone or FAISS).
4. These retrieved documents + the question are passed into a **generative language model** (like GPT).
5. The LLM **generates a response** grounded in the retrieved content.

□ Why RAG is Powerful:

Problem With LLMs Alone	How RAG Fixes It
Hallucination (made-up facts)	Adds real, grounded context
Outdated information	Retrieves fresh content from your DB/docs
Domain-specific gaps	Pulls from your own knowledge base
Token/context limit	Only injects **relevant chunks** of info

📦 RAG Components

Component	**Function**
Embedder	Converts queries/documents into vectors
Vector Database	Stores and searches embeddings
Retriever	Finds similar documents to the query
LLM Generator	Generates answers based on context
(Optional) Reranker	Improves quality of retrieved results

🔁 RAG vs Traditional QA

Feature	**Traditional LLM Answer**	**RAG Answer**
Knowledge Source	Model's training data only	External docs + model knowledge
Custom Domain Ready	No	Yes
Up-to-date Answers	No	Yes (if data source is updated)
Accuracy	Lower	Higher (retrieved facts)
Explainability	Hard to trace	Can show which documents were used

Langchain and Prompting Essentials

✍ Elements of a Good Prompt (for LLMs like ChatGPT)

A **prompt** is the input you give to a language model. A **well-crafted prompt** gets better, clearer, and more useful responses.

Here are the essential elements of a **good prompt**:

A communication with models

No preamble (don’t give any text before code)

✅ 1. Clear Intent

Be **explicit about what you want**:

- Bad: "Explain planets"
- Good: "Explain the eight planets of the solar system in simple terms for a 10-year-old."

✅ 2. Specific Task or Format

Tell the model exactly what to do:

- "List 5 pros and cons of electric vehicles."
- "Summarize this article in bullet points."

- "Generate an email reply declining the offer politely."

✅ 3. Proper Context

Give background info or reference material:

- "Here is a paragraph from my essay. Rewrite it to sound more academic:"
  [Insert paragraph]

✅ 4. Target Audience or Tone

Mention who it's for or how it should sound:

- "Explain blockchain to a beginner with zero coding knowledge."
- "Write in a formal, professional tone."
- "Make it sound casual and witty."

✅ 5. Constraints or Examples

Set rules or show an example:

- "Answer in less than 100 words."
- "Use only emojis to describe the story."
- "Follow this format: [Title] - [1-sentence description]."

✅ 6. Step-by-Step or Multi-Step Prompts

Break down what you want:

- "First explain what recursion is. Then give a simple Python example."
- "Step 1: Summarize the article. Step 2: Highlight 3 key insights."

✅ 7. Role/Persona Assignment

Frame the model's role to match your need:

- "You are a career coach. Give feedback on this resume."
- "Act like a Shakespearean poet and describe a thunderstorm."

□ Pro Tip: Use the "Instructions Sandwich"

Context ➤ Task ➤ Format ➤ Constraints

Example:

"You're a data analyst. Given the sales data below, generate a concise executive summary (3-4 bullet points), focusing on trends and outliers. Avoid jargon."

📌 TL; DR: Good Prompt Ingredients

Element	Description
✅ Clear Intent	What exactly are you asking for?
✅ Specific Task	Define the output type (list, summary, etc.)
✅ Context	Give relevant info or data
✅ Tone/Audience	Who’s it for, and how should it sound?
✅ Constraints	Word count, format, examples, etc.
✅ Role	Assign a persona or perspective
✅ Stepwise Logic	Break complex tasks into ordered steps

□ Zero-Shot, One-Shot, and Few-Shot Prompting

These terms refer to how many examples you give the model to learn the task within the prompt itself — not during training.

🔹 1. Zero-Shot Prompting

❌ No examples provided
✅ Just a direct instruction

📌 Example:

Prompt:

“Translate this to French: ‘I am hungry.’”

📎 Use when:

- The task is simple and well-known (e.g., translation, summarization)
- The LLM is already strongly trained in general language understanding

---

🔹 2. One-Shot Prompting

✅ One example provided
➕ Helps guide the model's format and logic

📌 Example:

Prompt:

"Translate this to French:
English: I am tired
French: Je suis fatigué

English: I am hungry
French:"

📎 **Use when:**

- Task needs structure or a format clue
- You want to bias the output style

🔷 3. Few-Shot Prompting

✅ A few examples (typically 2–5) provided
➕ Builds pattern recognition for more complex or custom tasks

📌 **Example:**

Prompt:

"Translate English to French:
English: I am tired
French: Je suis fatigué

English: I am thirsty
French: J'ai soif

English: I am hungry
French:"

📎 **Use when:**

- Task is ambiguous or custom (e.g., sentiment classification, style transfer)
- You need higher accuracy and consistency

✅ Comparison Table

Prompt Type	# of Examples	Best For	Model Needs to Learn Format?
Zero-Shot	0	Simple, well-known tasks	Yes
One-Shot	1	Custom output formats	Less than zero-shot
Few-Shot	2–5	Complex or non-standard tasks	No (pattern is clear)

□ Why This Matters

These prompting styles are the basis for in-context learning — where the model “learns” on the fly by examples in the prompt instead of retraining.

Langchain Installation

Groq and OLlaMA Setup:

Groq api: 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

Gemini api : AIzaSyB6fuBiYcwMLQLxHDfJgamoxGced66Ebko

Calling LLM from Langchain:

Import necessary libraries

For groq use api use dotenv model for the api hiding

Prompt Templates & Chains
look in the documentation for code

Vector Database

📌 **What Is a Vector?**

A vector is a list of numbers that represents an object in a multi-dimensional space. For example:

- A sentence like “I love cats” might be turned into a 768-dimensional vector like:
[0.23, -0.10, ..., 0.57]
- An image could be embedded into a 512-dimensional vector using a model like CLIP.

These vectors capture semantic meaning, so similar items have similar vectors (i.e., closer in vector space).

✅ **Use of Vector Databases**

1. **Semantic Search**
   - Find documents, products, or content **based on meaning**, not exact keywords.
   - Example: Searching for “tall mountain” returns results with “Mount Everest” even if the exact phrase isn’t used.
2. **Recommendation Systems**
   - Suggest similar movies, songs, or products by comparing user/item embeddings.
3. **Question-Answering & Chatbots (e.g. RAG Systems)**
   - Vector databases store document embeddings. When a user asks a question, the system:
     - Converts the query into a vector.
     - Finds the most relevant documents using vector similarity.
     - Feeds them to an LLM to generate a precise answer.

4. **Image & Video Search**
   - Retrieve visually similar images or video clips using visual embeddings.
5. **Anomaly Detection**
   - Detect outliers by measuring how far a vector is from the "normal" cluster.

🔍 How It Works

1. **Data Preparation**: Convert text/images into vectors using models like:
   - BERT, OpenAI Embeddings for text.
   - CLIP, ResNet for images.
2. **Indexing**: Vector DBs use advanced algorithms (like HNSW, IVF, PQ) to index vectors for **fast similarity search**.
3. **Querying**: At query time, the user vector is matched to **nearest neighbours** in the vector space.

Feature	Traditional DB	Vector DB
Data Type	Structured (e.g., numbers, strings)	Unstructured (e.g., text, images)
Search Type	Exact match	Similarity search
Use Case	Inventory, finance	AI search, recommendations, RAG

Chromadb: Introduction and Installation:

The fundamentals and concept behind vector database

How it is used for GenAI application

Don’t focus on the syntax

Basic Operations in Chromadb:

□ 1. Initialize Chroma Client and Collection

```
import chromadb

# Create or connect to a Chroma client (in-memory by default)

client = chromadb.Client()

# Create a collection (like a table)

collection = client.create_collection(name="my_collection")
```

📥 2. Add Data (with or without embeddings)

```
# Example with user-generated embeddings

collection.add(
  documents=["What is AI?", "How does a rocket work?"],
  embeddings=[[0.1, 0.2, 0.3], [0.9, 0.8, 0.7]],
  ids=["doc1", "doc2"],
  metadatas=[{"topic": "tech"}, {"topic": "space"}]
)
```

You can add:

- documents: the raw text
- embeddings: vector representations (optional if using Chroma’s built-in embedding support)
- ids: unique identifiers
- metadata’s: optional metadata

🔍 3. **Query (Semantic Search):**

Semantic matching in ChromaDB refers to the process of comparing and retrieving data based on meaning or context, rather than just exact keywords

```
results = collection.query(
  query_embeddings=[[0.1, 0.2, 0.3]],  # vector of the query
  n_results=2
)
print(results)
```

Result includes:

- matching documents
- their ids
- distances or similarity scores

□ 4. Get Entries by ID:

```
docs = collection.get(ids=["doc1"])
print(docs)
```

□ 5. Delete Data:

Delete specific documents by ID:

```
collection.delete(ids=["doc1"])
```

Or clear all:

```
collection.delete()  # Wipes entire collection
```

□ 6. Update Documents or Metadata

As of now, Chroma does **not** support partial updates to a document. To "update", you typically:

1. Delete the item.
2. Re-add it with new data.

🧱 7. **List or Modify Collections:**

```
# List all collections
collections = client.list_collections()


# Get existing collection
my_collection = client.get_collection("my_collection")


# Drop a collection
client.delete_collection("my_collection")
```

Can use pretrained models of sentence transformers (check the documentation)

✅ Euclidean Distance vs. Cosine Distance

Both **Euclidean distance** and **Cosine distance** are commonly used to **measure similarity between vectors**, but they differ fundamentally in **what they measure**.

📐 1. Euclidean Distance

📌 What it Measures:

- The **straight-line distance** between two points in multi-dimensional space.

□ Formula:

For two vectors **A** and **B**:

$$Euclidean(A,B) = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2}$$

□ Intuition:

- Think of it like measuring the distance between two dots on a plane.
- Affected by the **magnitude (length)** of the vectors.

🔄 Range:

- From 0 (exact same point) to ∞.

📌 When to Use:

- When **magnitude matters** (e.g., physical space, clustering similar-sized entities).

📐 2. Cosine Distance (or 1 - Cosine Similarity)

📌 What it Measures:

- The **angle between two vectors**, not their magnitude.

□ Formula:

$$\boldsymbol{Cosine\ Similarity}(\boldsymbol{A},\boldsymbol{B}) = \frac{\boldsymbol{A}.\boldsymbol{B}}{||\boldsymbol{A}||\,||\boldsymbol{B}||}$$

$$\boldsymbol{Cosine\ Distance}(\boldsymbol{A},\boldsymbol{B})\ =\ \mathbf{1}\ -\ \boldsymbol{Cosine\ Similarity}(\boldsymbol{A},\boldsymbol{B})$$

□ Intuition:

- Measures how **aligned** two vectors are in direction.
- Ignores magnitude (length); only **direction matters**.

🔄 Range:

- Cosine **similarity**: [-1, 1] (commonly [0, 1] for non-negative vectors)
- Cosine **distance**: [0, 2] (commonly [0, 1] in practice)

📌 When to Use:

- When you care about **semantic similarity**, not magnitude.
- Common in **text embeddings**, **document search**, and **semantic matching**.

✅ Summary Table

Feature	**Euclidean Distance**	**Cosine Distance**
Measures	Absolute distance	Angle/direction difference
Sensitive to magnitude	✅ Yes	❌ No
Common in	Clustering, image, geo	Text, NLP, recommendations
Value range	0 to ∞	0 to 2
Similarity = 0	Same point	Same direction

Agentic AI: A Hands-on Approach

Agency in AI:

Regular chat bot it only answers what it is trained on

AI chat bot [with some inbuilt apis tools] and it use all these tools and databases and with previous knowledge of the user it takes independent actions and generates answer

Work Flows are like GPS apps with fixed routes

Agents are like drives who adapt to roadblocks and make their own decision

Build Your First Agent using Llama and Agno:

Level -1 with No autonomy

Level -2 with some

Agent with Custom Tool

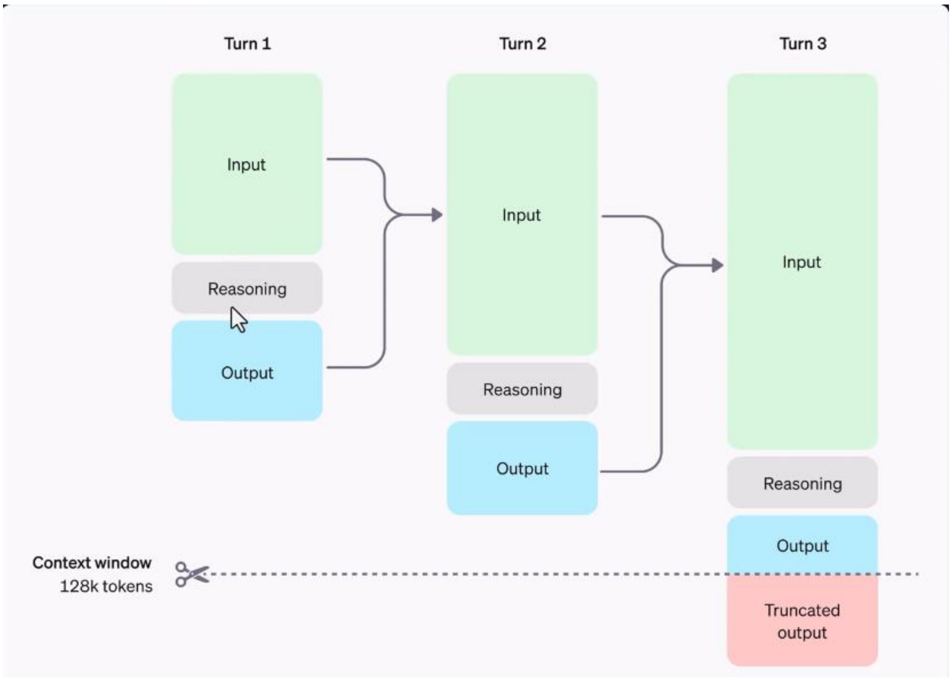
What are Reasoning Models?

LLm Models

Previous Models (gpt 3.5) [next token prediction]

Reasoning Models (o1)

Trained to reason step-by-step handle multi-step thought process



Train with Chain-of-Thought (CoT) Examples

Basic LLM Models:

Content Writing

Text Summarization or Extraction

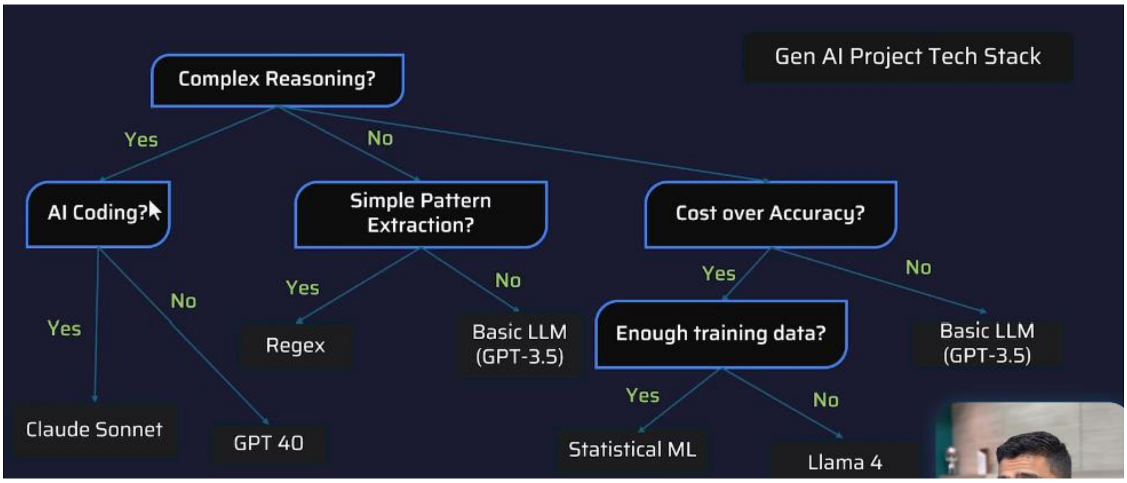
Simple Q&A

Reasoning Models[cost effective]:

Complex Problems Solving

Coding

Scientific Reasoning



Gen AI Project Tech Stack

Complex Reasoning?
- Yes → AI Coding?
  - Yes → Claude Sonnet
  - No → GPT 4O
- No → Simple Pattern Extraction?
  - Yes → Regex
  - No → Basic LLM (GPT-3.5)
- No → Cost over Accuracy?
  - Yes → Enough training data?
    - Yes → Statistical ML
    - No → Llama 4
  - No → Basic LLM (GPT-3.5)

https://docs.agno.com/introduction refer

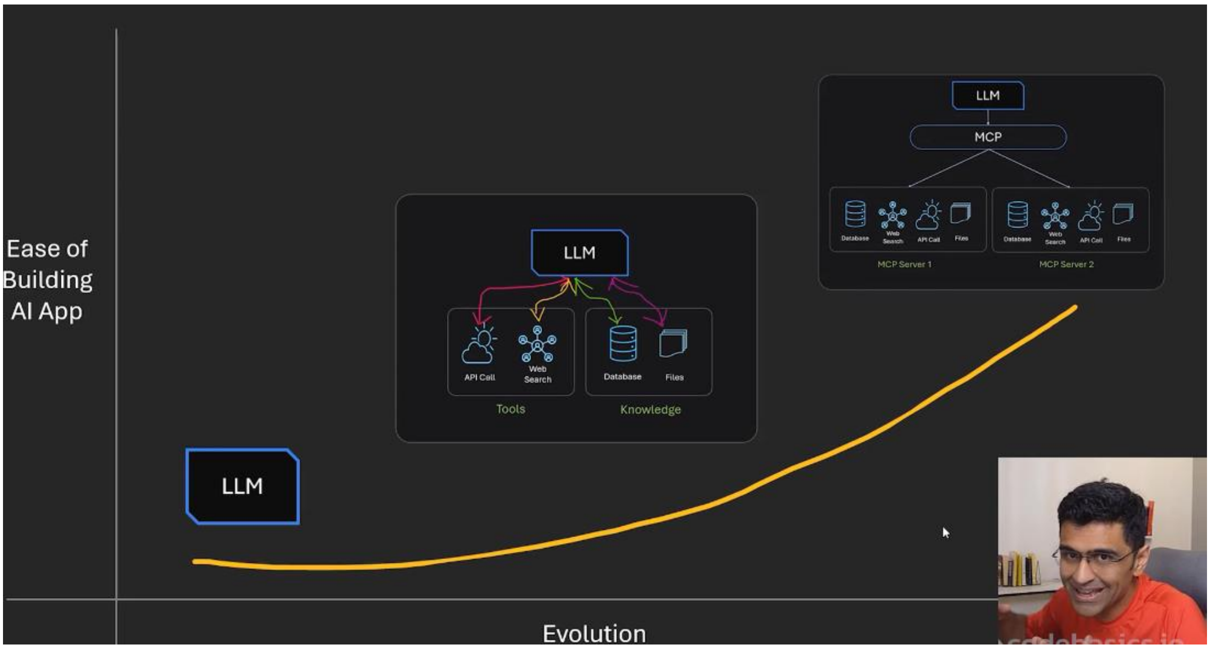
Multimodal Agent

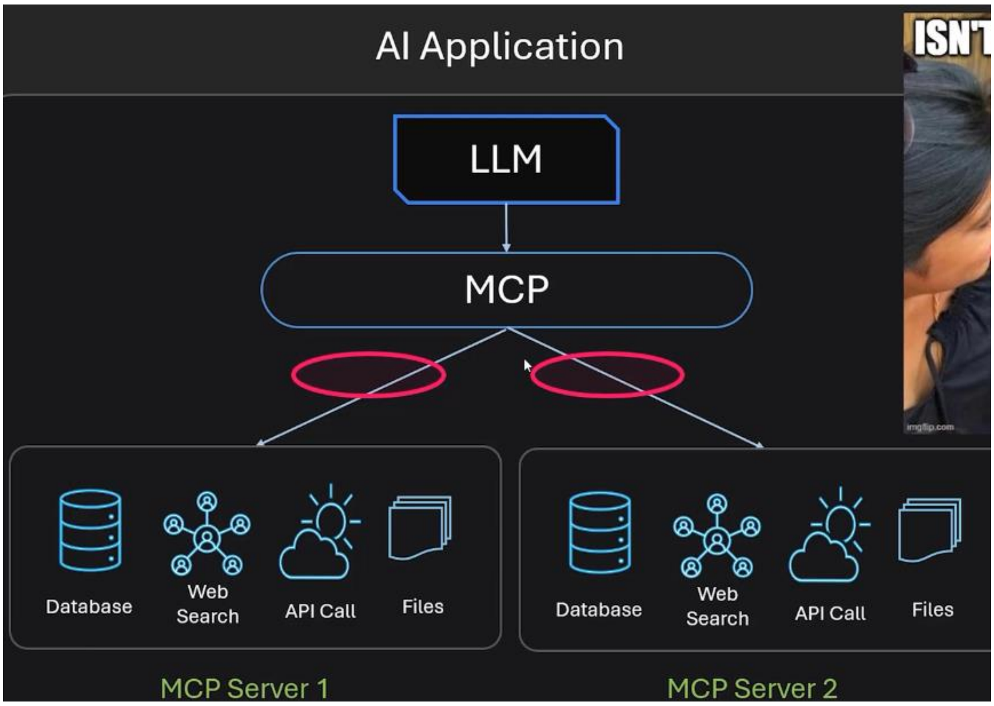
Audio, video formats or images getting info regarding those images what is that

Other Frameworks: Smolagents, Google ADK

Agentic AI: Architecture and Protocols

What is Model Context Protocol (MCP)?





The **Model Context Protocol (MCP)** is an open standard—introduced by Anthropic in November 2024—that enables LLM-based systems to **seamlessly integrate with external data sources and tools** using a uniform, secure interface timesofindia.indiatimes.com+15en.wikipedia.org+15theverge.com+15.

□ What MCP Does

- Acts like a universal "USB-C" connector for AI apps—letting models interact with files, databases, APIs, tools, apps, and more opencv.org+8modelcontextprotocol.io+8reddit.com+8.
- Eliminates the need for custom integrations for each tool — one MCP server exposes capabilities so any MCP-aware client can use them opencv.org+1modelcontextprotocol.io+1.
- Uses JSON-RPC 2.0 over studio or HTTP/SSE to manage sessions and capabilities between **Host**, **Client**, and **Server** components en.wikipedia.org+4opencv.org+4mcp.ai+4.

🛠 Core Components

1. **MCP Server**
   - Wraps a system (e.g., GitHub, Postgres, Google Drive) and exposes its functions via MCP.
2. **MCP Client**
   - Embedded in AI hosts; communicates with servers over the protocol.
3. **MCP Host**
   - The application environment (e.g., Claude Desktop, IDE) managing connections, permissions, and context

reddit.com+13mcp.ai+13activepieces.com+13arxiv.org+6opencv.org+6modelcontextprotocol.io+6theverge.com+5geeksforgeeks.org+5en.wikipedia.org+5.

✅ Why It Matters

- **Interoperability**: Build once, and your tool works across multiple AI agents—no vendor lock-in zapier.com+7blog.laratranslate.com+7reddit.com+7.
- **Context & Currency**: Enables real-time access to up-to-date data, reducing hallucinations reddit.com+4michaelwapp.medium.com+4reddit.com+4.
- **Agentic workflows**: Ideal for multi-tool, autonomous AI agents that fetch, reason, and act on data reuters.com+8en.wikipedia.org+8arxiv.org+8.
- **Security**: Offers permission boundaries and audit trails for safer integrations mcp.ai.

🌍 Adoption & Ecosystem

- Supported by OpenAI, Google DeepMind, Microsoft, Zapier, Replit, Sourcegraph, and more en.wikipedia.org+1zapier.com+1.
- Used in tools like Claude Desktop, Cursor IDE, and enterprise settings for database and document access geeksforgeeks.org+7en.wikipedia.org+7blog.laratranslate.com+7.
- Community-built servers cover GitHub, Slack, Gmail, Postgres, Microsoft Graph, and more reddit.com+1en.wikipedia.org+1.
- Microsoft's CTO likened MCP to the moral equivalent of HTTP for "agentic web" tools mcp.ai+15theverge.com+15activepieces.com+15.

⚠ Challenges

- **Security risks**: Vulnerabilities include malicious tool servers and "tool poisoning" arxiv.org+1arxiv.org+1.
- **Complexity**: Debate over whether MCP adds overhead compared to simpler API integrations arxiv.org+5reddit.com+5thoughtworks.com+5.
- **Governance**: Reliance on Anthropic's stewardship may affect standard openness de.wikipedia.org+2thoughtworks.com+2reuters.com+2.

💬 Community Insights

"Imagine it as a USB-C port — but for AI applications." arxiv.org+7reddit.com+7docs.anthropic.com+7
"Traditional APIs require separate code; MCP gives you dynamic discovery and two-way communication." reddit.com

🔚 TL;DR

MCP is a standardized, open protocol that lets LLMs and AI agents **securely access and act** upon external tools, apps, and data **in real time**—empowering more dynamic, multi-tool, agentic AI workflows without vendor lock-in.

Build Your First MCP Server: Leave Management:

Agentic AI: Multi-Agent System

Build Your First Multi Agent Program:

Have more than one agent