Agenda

1) Introduction → Traditional ML → MCP or A2A
2) Pydantic Python → pydantic

Prequisitics → Python → Must

History → imp

2013-2014 → Statistics → Conclusion → we started here

⇓

Statistical analysis ⇒ Observation, conclusion.

Machine learning Algorithm : DATA → Model → Pattern of DATA → Predictions.

Independent And Dependent features

f1 f2 f3 f4 — O/P — TABULAR

House size — Area — Price — O/P

ETL

2011 ⇒ BIG DATA FEATURES

BIG DATA ←

2007 : Fb, Instagram, Whatsapp

Huge amount of DATA

⇓

STORE

EXTRACT → TRANSFORM → LOAD ⇒ DATA Scientist

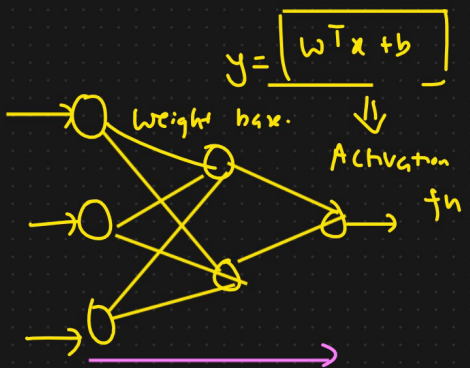
AI → Use this DATA → PATTERNS → Understand our customer.

Recommendation Engine. : Sapient

Neural NW

Multi layered NN.

$y = w^T x + b$



$w' = w - \alpha \frac{\partial h}{\partial w}$

DEEP LEARNING

- Tabular DATA → ANN
- Images → Computer Vision
  - CNN
  - ALEXNET, ROSNET
    VGG16, INCEPTION
  - Detectron 2
  - RCNN
  - MASKED RCNN
  - YOLO
- NLP
  - RNN
  - LSTM RNN
  - BIDIRECTIONAL LSTM RNN
  - Encoder Decoder
  - ATTENTION IS ALL YOU NEED
  - TRANSFORMERS → GENAI

Image algos
→ classify
- Alexnet

→ Detection
- Yolo

2K +

NLP DATA = Text

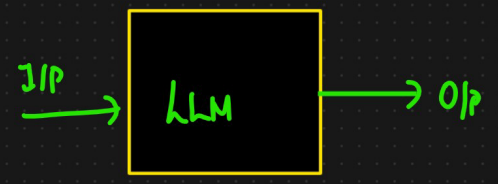
LLMS → NLP ⇒ TEXT

GAME CHANGING

3 Core : DATA SCIENCE + GENAI → MARCH

Agentic AI →

Generative AI



Dataset

UPSC

1 year

8-9 Subjects

10 Subjects

Books

Newspaper

[Generate]

Test paper

1st RANK

Open AI

LLM Model

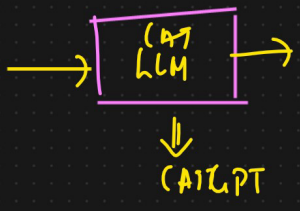
Google Gemini Model

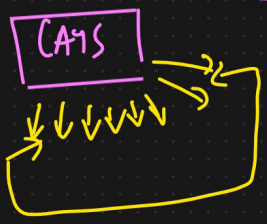
LLAMA

Examiner ← Metrics

Metrics

[Generate]





HALLUCINATED

GaurdRails

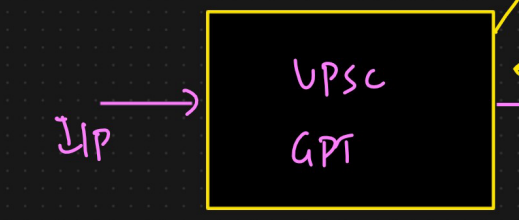
RAG → Retrieval Augment Generation

UPSC → PRELIMS, MAINS

NEWSPAPER ⇒ 30 days

LAST Month

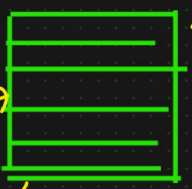
External DATA SOURCE



Vector DB ⇒ DB

30 days

{ Tell about Today NEWS }



Vector DB ⇒ Vectors

Text ⇒ Vectors

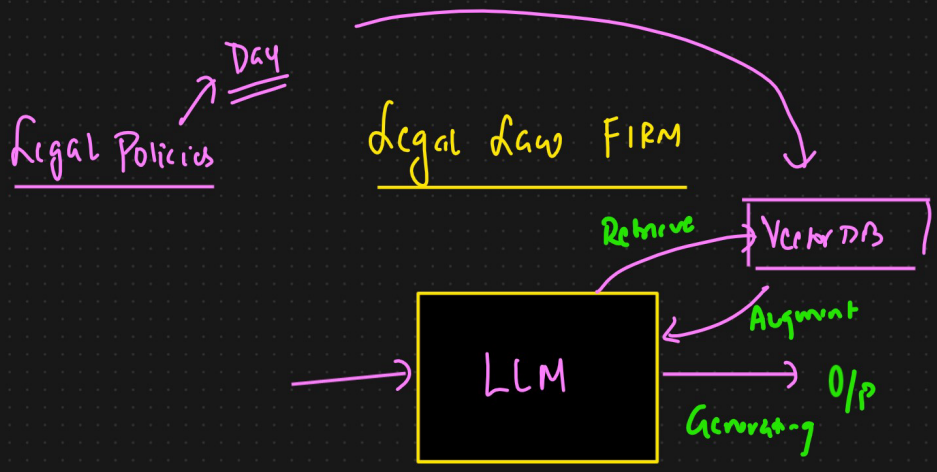
Embedding 10th May

Context

O/P

Prompt ←

TRADITIONAL RAG's => Agentic RAG

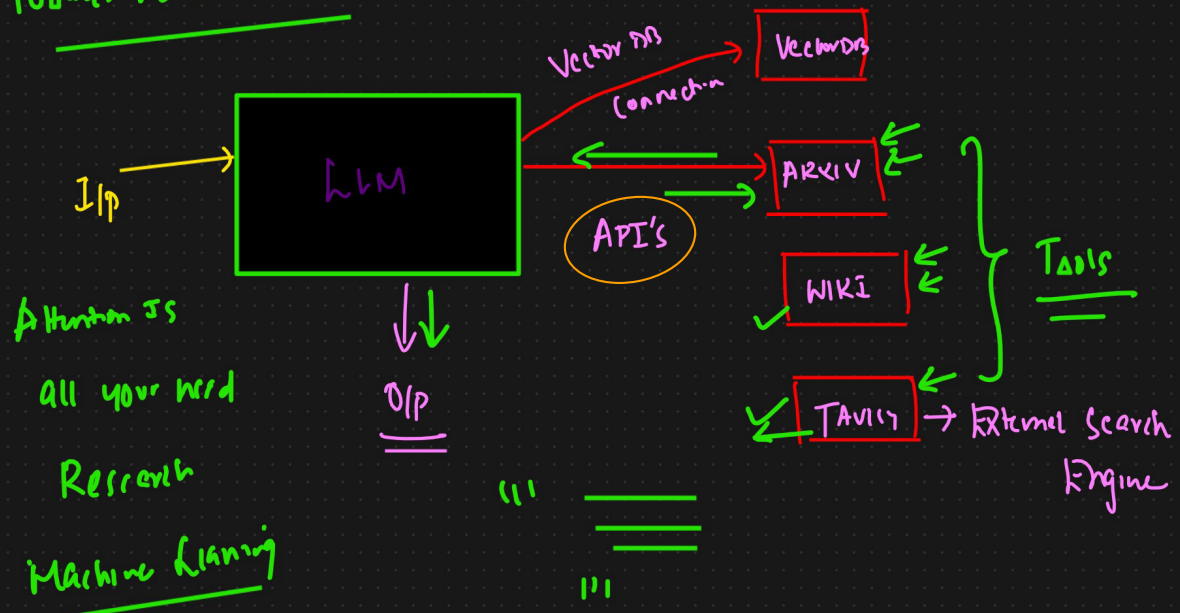


RAG

Pinecone.
Chroma DB
DATASTAX
Qdrant
ASTRA DB
FAISS DB

Tools use on AI

Gen AI Application



Companies

→ ARXIV →

WIKI

TAVILY

—

—

Attention is all your need

Research

Machine Learning

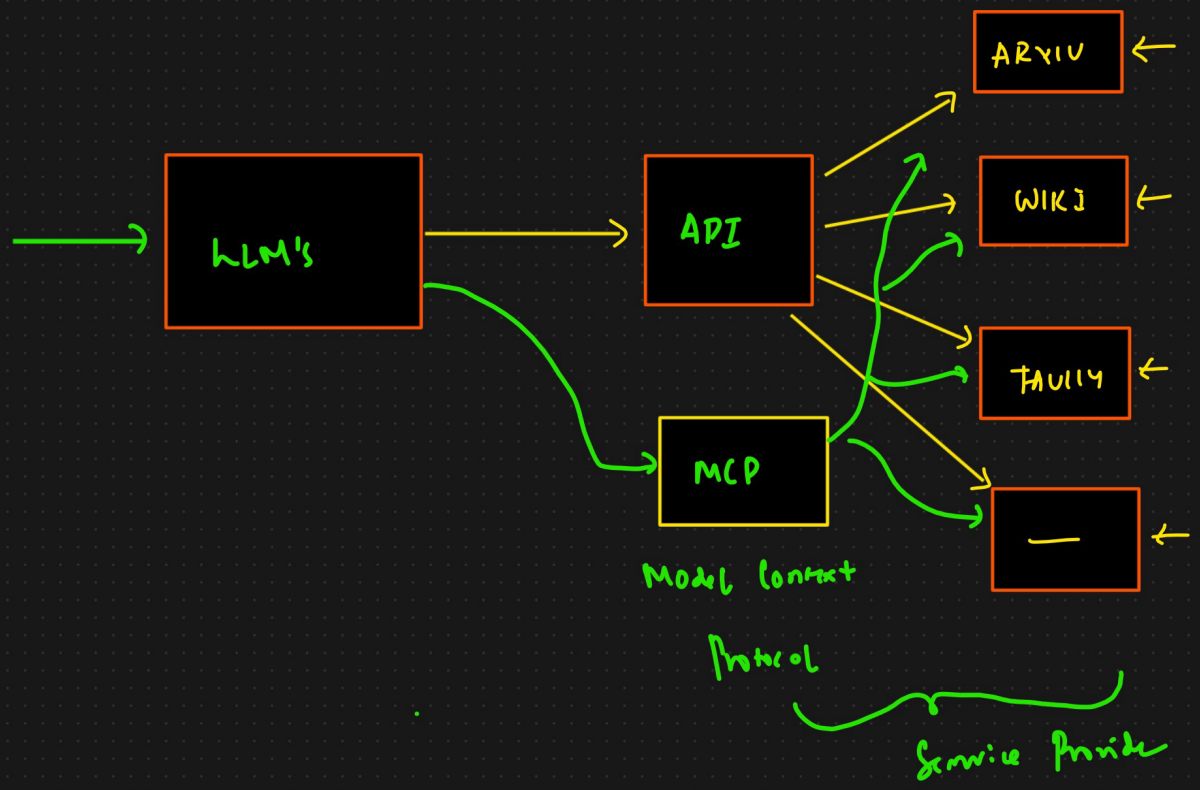
```
'''
'''
```

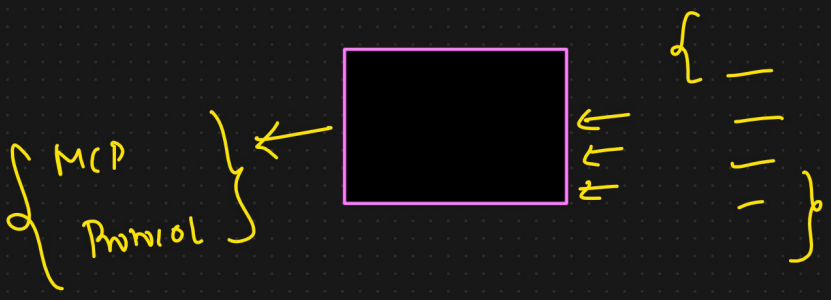
```
func():
    ''' Comment In python '''
```

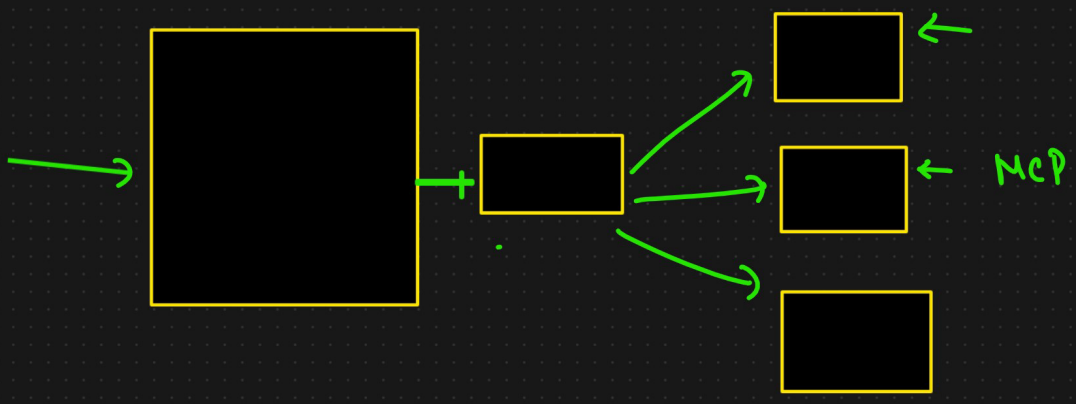
Tavily = External Search Engine

wiki = General Encyclopedia.

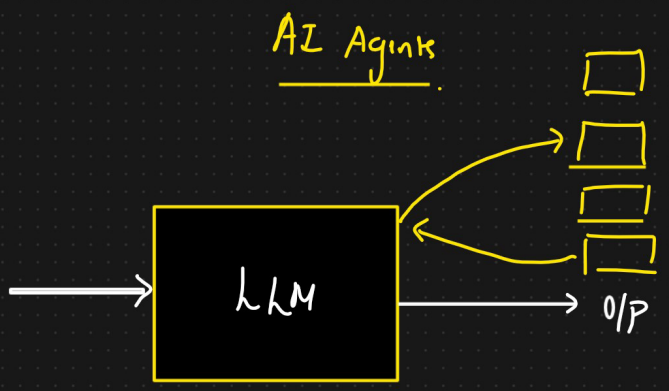
Arxiv = open source Research papers


ARXIV
LLM's
API
WIKI
TAVILY
MCP
Model Context
Protocol
Service Provide


MCP
Protocol


MCP

Generative AI Vs Agentic AI



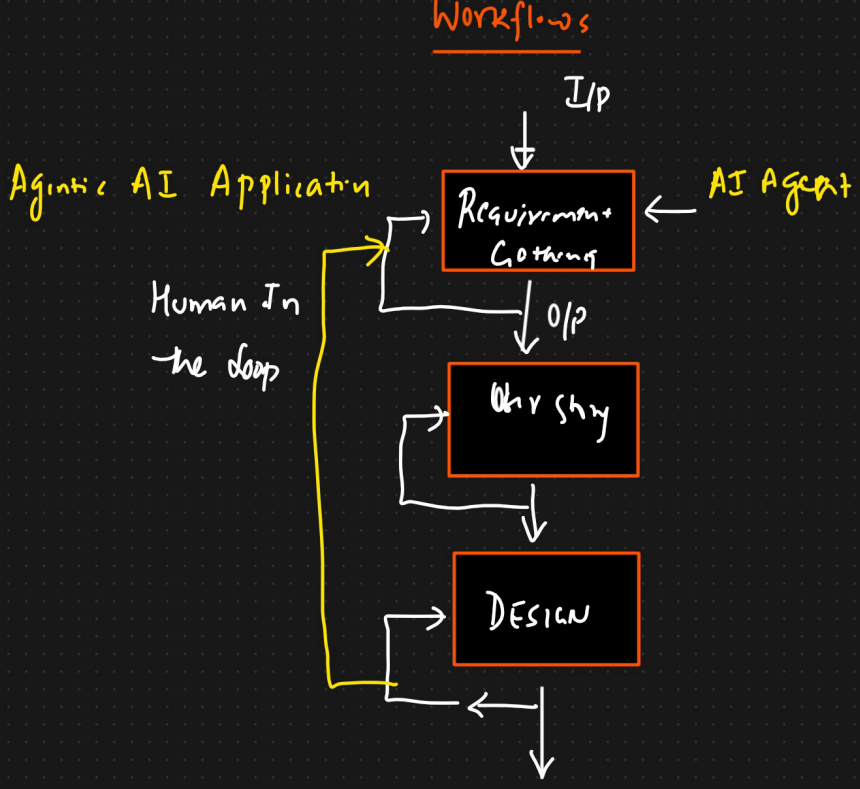
Agentic AI → Complex Workflows → SDLC

V.V. imp

Software Development life cycle.

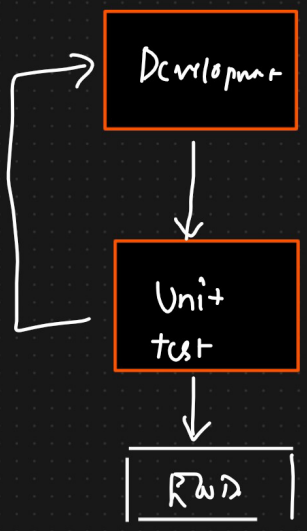
LLM + Human Feedback

1) Requirement Gathering : Business Analyst, Product Manager
2) Documentation And User story : BA
3) DESIGN ←
4) Development ←
5) Testing ←



Industry Process

AI Agents Vs Agentic AI



7740.

Automate Ticket Support

Email → Support / Billing / — ⇒ CRM

Mock Interview ÷ 15 minutes → B2B

B2C

Face Detection

Resume
Job Description

2 or 3 minutes ← 0

Tool

Technical Questions
Project Explanation
} Resume Around Mistake

Reports → Feedback

Pydantic → FAST API ←

DATA Model

Class

HTTP Post/get

API's

/login

/get

Rust

Very very fast

JSON

{ "name": "Krish" Age : 35 }

DATA VALIDATION

=> DATA Type { Type hinting }

DATA CLASSES => DATA VALIDATION

Yt to Blog

1) YT video → TRANSCRIPT

2) TRANSCRPT

LLM

Blog Headline : String

Blog Content : String

Conclusion : String

Class (BaseModel)

{ headline : str, content : str, Conclusion : str }

LLM

Journey to Gen AI

1. Statistical Techniques
   - Descriptive 2. Inferential
2. ML Algo's
3. Big Data → with Big in picture, ML algo's were not sufficient & accurate
4. Deep Learning → MLP (Multi level perceptron)
   1. ANN — Tabular Data
      - Classify
      - predict
   2. CNN — Images / Video
      - Image classify
        - Resnet (Residual)
        - Alexnet
      - Object Detection
        - Yolo (You only look once)
        - Retina Net
   3. RNN — Sequential Data, Text Data.
      - RNNS
      - LSTM / GRU
      - Encoder / Decoder
        ↓
        with attention

        Transformers

        Gen AI.

        LLMs

Note: If not interested in Computer Vision, skip

Q. How is Gen AI different from ML?

Gen AI → focuses on Generation (images, text)

ML → focuses on prediction / Classification

Q. What are LLM hallucinations

When model gives non sensical output.

- unclear prompts
- No RAG
- No specific Data

RAG; helps LLM + External Data + Query — Traditional RAG

- Makes LLM upto date and company specific without retraining entire LLM (basically adds external data as extra content) tokens

- RAG reduces hallucinations

👉 **How it works (in short):**

1. **Retrieve**: Pulls relevant info from external sources (docs, DBs, web).
2. **Augment**: Feeds that info into the LLM as extra context.
3. **Generate**: LLM creates the final answer using both its training + retrieved data.

Imp: Without RAG

Average height of human = 5 feet

With RAG

Avg height + Vector db (baseball team) ↳ 6 ft +

In **RAG**:

- Your **company rules / revenue docs** are **pre-converted into embeddings** and stored in a **vector DB** (done once, not every query).
- When you ask: *"What is this company's revenue?"* →
  1. Your **query is converted into an embedding**.
  2. RAG **retrieves the most relevant chunks** (e.g., revenue section from the doc).
  3. That text is **added to your prompt** → LLM processes it with its transformer layers.
  4. LLM generates the final answer with both **its knowledge + retrieved doc context**.

Imp

👉 So yes: each time you run a query, RAG fetches relevant **pieces of your docs** and **temporarily injects them** into the LLM to guide the response.

VECTOR DB where data is stored in the form of vectors.

↳ Embeddings ; for Augmenting

🔷 **What is a Vector DB?**

A **Vector Database** is a database designed to **store and search embeddings (vectors)** efficiently.

- Each document, image, or query is turned into an **embedding** (high-dimensional vector).
- The DB then allows **similarity search** (e.g., "find me vectors most similar to this query").
- It's the backbone of **RAG**, semantic search, recommendation systems, etc.

- **Pinecone** → Managed, cloud-native, easy for RAG.
- **Weaviate** → Open-source, hybrid search, scalable.
- **Milvus** → Open-source, enterprise-grade, highly scalable.
- **FAISS** → Fast similarity search library, widely used for prototypes.
- **Chroma** → Lightweight, popular for local RAG setups.
- **Postgres + pgvector** → SQL DB with vector search support.

🔷 Workflow: PDF → RAG → LLM Answer

1. **Ingest the PDF**
   - Load your company PDF (e.g., policies, financials).
   - Split it into **chunks** (small passages, like 300–1000 words each).
2. **Create Embeddings**
   - Use an **embedding model** (e.g., OpenAI `text-embedding-ada`, Hugging Face models, etc.) to convert each chunk into a **vector**.
3. **Store in Vector Database**
   - Save these vectors + original text in a **Vector DB** (Pinecone, Weaviate, Milvus, FAISS, Chroma, etc.).
4. **Query Processing**
   - User asks: *"What was the company's revenue last year?"*
   - Query is also converted into an **embedding**.
5. **Retrieve Relevant Chunks** (cosine similarity) → similar words similar vectors
   - Vector DB does **similarity search** (e.g., cosine similarity).
   - Fetches top N most relevant chunks (e.g., financial statement from PDF).
6. **Prompt Augmentation**
   - Retrieved text is **added to the LLM prompt**.
   - Example:

```sql
Context: [Extracted revenue data from PDF]
Question: What was the company's revenue last year?
```

7. **LLM Generation**
   - LLM (like GPT, LLaMA, Claude) runs transformer operations on this **augmented prompt**.
   - Produces a final answer, grounded in your company PDF data.

⚡ **In short:**
PDF → split → embeddings → vector DB → query → retrieve relevant chunks → LLM with context → answer.

additional context

MCP = Advanced API's with Conversational Layer.

Standard protocol to provide context (data) to LLMs in a standardized manner.

🔷 Key Components of MCP → Model Context Protocol

- **MCP Server** → The **data provider** (docs, APIs, DBs, tools).
- **MCP Client** → The **LLM (or app using the LLM)** that needs that data.
- **MCP Protocol** → The **standard rules** for how the server and client talk, share, and structure context.

API = Generic way to send data (need maintenance)

MCP = Standard way. ↳ protocolized API no maintenance needed.

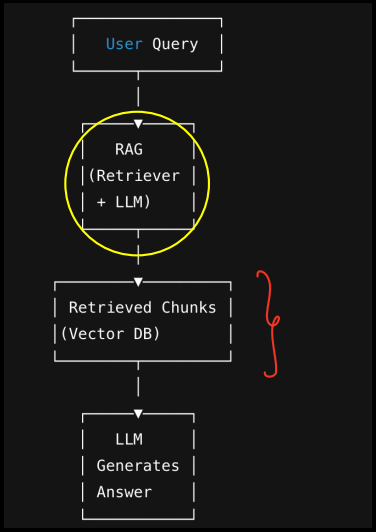
Q11 Difference b/w RAG & MCP

⚡ **Key Difference (Short)**

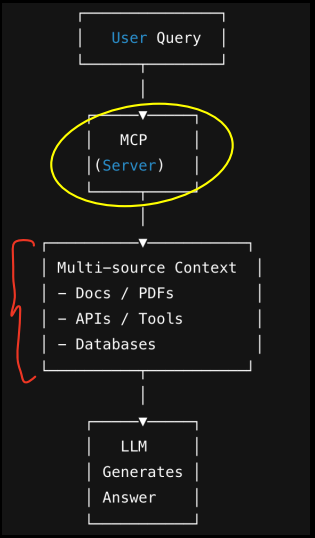
Aspect	RAG	MCP
Purpose	Inject external data into LLM	Provide structured, multi-source context
Integration Scope	Mainly documents / vector DB	Documents + APIs + tools + enterprise systems
Standardization	No formal protocol	Standardized protocol for LLM context
Use Case	Fact-based answer generation	Enterprise LLM apps, multi-tool orchestration

Imp.

RAG =



MCP =



Imp.

- **MCP provides external context** to the LLM.
- Example flow:
  1. You ask: *"What is the average rent in Paris?"*
  2. MCP **adds Airbnb-specific context** (listings, pricing, policies).
  3. LLM processes **query + context** using its transformer layers.
  4. You get an answer that is **both accurate and grounded in Airbnb data**.

👉 So think of MCP as a **"context injector"** for LLMs, making outputs **domain-specific, up-to-date, and reliable**.

AI AGENTS; Reason, plan & take actions

- AI that works as an autonomous agent.
- It can Reason, plan & take actions.

- Uses:
  1. **LLM** (brain for reasoning)
  2. **Tools/APIs** (to act, fetch, or compute)
  3. **Memory/feedback** (to adapt and improve)

AGENTIC AI; when multiple agents work together.

🔷 Example

- Normal LLM → *"How do I book an Airbnb under $200?"* → gives you steps.
- **Agentic AI** → actually searches Airbnb, compares listings, and suggests/book one.

✅ In short:

LLM = talks.

Agentic AI = thinks + decides + acts.

Q. How is Agentic AI different than Power Automate?

🔷 Traditional Automation ⟶ Rule Based

- Rule-based: you design fixed steps.
- Deterministic: it does the **same thing every time**.
- Needs humans to design workflows.

Example:

In **Power Automate**, you build a flow:

1. When an email arrives with subject "Invoice" →
2. Save attachment to OneDrive →
3. Notify on Teams.

👉 If anything unexpected happens (e.g., email format changes), the automation **fails**.

imp

✅ Core Difference:

- **Traditional Automation** = "If X happens, do Y" (rigid).
- **Agentic AI** = "Get me to outcome Z" (flexible + reasoning).

🔷 Agentic AI ⟶ Goal oriented.

- Goal-oriented: you give it an **outcome**, not steps.
- Adaptive: it can **reason, plan, and change its approach**.
- Autonomous: can combine tools, APIs, and logic dynamically.

Example:

You tell an **AI agent**:

"Whenever I get a new invoice, extract vendor name, due date, and amount, then update my finance tracker."

👉 The AI agent will:

1. Detect invoice emails—even if format changes.
2. Use OCR/LLM to parse the PDF.
3. If the tracker API is down, it might retry or export to Excel instead.
4. Notify you only if something looks suspicious.

It **figures out the workflow itself** instead of you hardcoding every rule.

Agentic AI example (in Software Dev.)

1. Requirement Gathering → agent 1
2. User story → agent 2
3. Development → agent 3
4. Test → agent 4
5. production → agent 5

} Agentic AI Application

REST API; Representational State Transfer

is an API that exposes Resources (like users, products, orders) through unique URLs and lets applications interact with each other.

FAST API; Web Framework.

FastAPI is a **modern, high-performance web framework in Python** designed for building APIs. It's built on **Starlette** (for web handling) and **Pydantic** (for data validation). It's known for being fast, developer-friendly, and production-ready.

✅ In short:

- **Starlette** → Handles the **web stuff** (requests, routing, middleware).
- **Pydantic** → Handles **data validation and serialization**.
- **FastAPI** = combines both to make building REST APIs super fast.

PYDANTIC; → Data validation, constraints
is a python library for data validation (using python type hints).

It ensures that the data your app receives (e.g., from an API request, a file, or environment variables) matches the expected types and structure.

- very fast as it is using RUST Language

🔑 Key Features

- **Type validation** → If you say `age: int`, Pydantic enforces it.
- **Data parsing** → Automatically converts types (e.g., `"123"` → `123`).
- **Error handling** → Gives clear validation errors.
- **Serialization** → Converts Python objects → JSON easily.

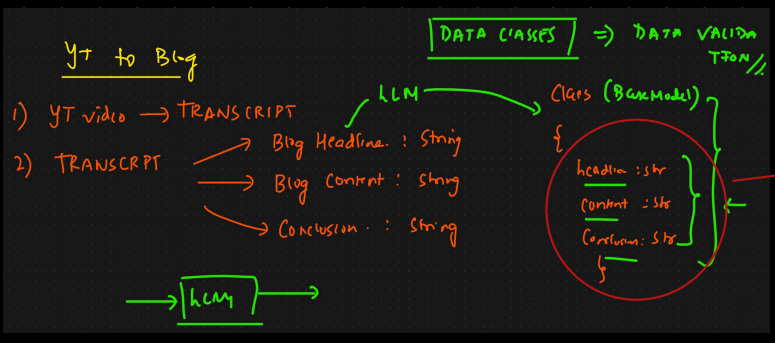
USECASE OF FAST API;
is much popular in AI/ML Domain.

❓ Example Use Case (Machine Learning deployment)

You can train a model in scikit-learn or PyTorch, then expose it via a FastAPI endpoint like `/predict` so external apps/users can send data and receive predictions in real-time.

Q) Why do we use Pydantic

Because it is applicable in LLM. i.e., enforce Data Types in LLM's.



Tools

Framework
- Langchain = Gen Ai Applications
- Langgraph = Agentic Ai Applications

Lang Smith = Monitoring & Testing.

Generative
- **LangChain** → A framework mainly for **building GenAI applications**.
  - It helps connect LLMs with tools, databases, APIs, and custom logic.
  - Commonly used for chatbots, RAG (retrieval augmented generation), and orchestration of prompts.

agent
- **LangGraph** → A newer library built **on top of LangChain**, designed for **agentic AI applications**.
  - It introduces **graph-based workflows** where each node is an agent/tool, and the system dynamically decides the flow (like branching logic, retries, looping).
  - Useful for **multi-agent systems**, autonomous workflows, and stateful reasoning.

👉 In short:
- **LangChain = foundation for GenAI apps.**
- **LangGraph = structure for agentic/multi-agent apps (built using LangChain concepts).**

Q1 Difference b/w Type Casting / Hinting

Type Casting - Change Datatypes

Type Hinting - Validate Datatypes.

Q2 Difference b/w Langchain & RAG ?

🚀 Key Difference
- **LangChain = toolbox/framework** to *implement* AI applications.
- **RAG = technique/architecture** for making LLMs more accurate & up-to-date.