What are agents anyway?

Generative AI = great at understanding and generating content
Agentic AI = understands , generates and performs actions

Why does performing action matter?

Initial genAI used to do

- Summarize a document
- Suggest a product to a user
- Sentiment analysis
- Research / Q&A

Agentic :

- Summarize a document + put it in the right folder and create downstream tasks(do further research on a topic)
- Suggest a product to a user + create landing pages web/apps
- Dont give marketing leads + email

How Agents take action ?

The magic lies in **Tools**

Most agents are paired with APIs, function calls, plugins that let them interact with external systems . The LLM just does not respond with text - it outputs commands like structured outputs

- Call the send_email() function with following inputs…
- Fetch records from the CRM system using this query

Tool use(function calling). The agents are told what tools / functions are available and agents automatically figures out when and how to use these tools through some planning mechanism

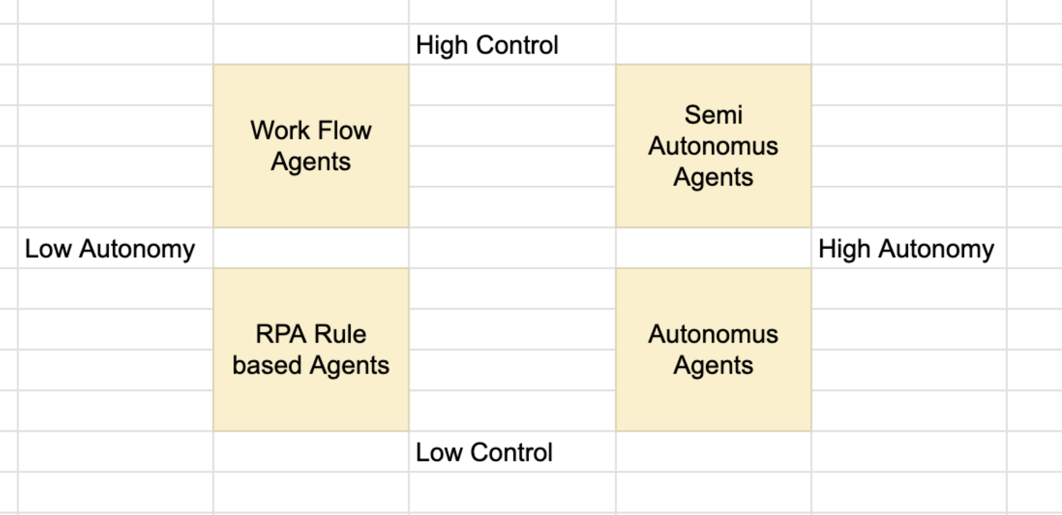
In more advanced systems:

- **Memory** = remembers past steps or context
- **Planning Modules =** deciding what to do next especially if there are multiple tasks
- **State management** = tracking progress and avoid looping and failures

Agents = LLM + Tools + Planning Modules + Memory + State management (technically)
Agents = Systems that complete tasks end to end (business)

Useful Mental Model : Autonomy vs Control :

Given a business problem , how autonomous you want your agents to be



		High Control		
	Work Flow Agents		Semi Autonomus Agents	
Low Autonomy				High Autonomy
	RPA Rule based Agents		Autonomus Agents	
		Low Control		

Low Autonomy , Low Control = RPA/ Rule based agents
These systems are not using LLM at all. They are built on traditional if this then that logic. Every decision path is manually scripted . No reasoning / no learning

1. Approve reimbursements under a particular thresholds from a set of vendor
2. Rename folder name based on filenames inside it
3. Copy data from excel sheet and paste in different fields of a form

Low Autonomy and High Control (Workflow Agents)
First step for different enterprises introducing LLMs in their workflows. Here LLM enhances the productivity of the existing workflow , bit does not execute actions at all. A human will be responsible for executing action.

1. Generating summary of meeting transcripts (Agents) + Human will send the email with AI / summary
2. User asks natural language query in a BI dashboard , LLM converts it into structured search query and then human clicks to finally load the dashboard

High AUtonomy , Moderate / high control (Semi Autonomous Agents)
These are thought to be true agentic systems . They can not only understand task but plan , execute , invoke tool and complete business goal. However,they often work under some sort of human monitoring and constraints

1. Anomoly identification agent that has ability to identify the anomaly, find root cause and generate jira tasks to fix it . (human will review the task)

High Autonomy , Low control Agents (Autonomous agents):
These agents are fully autonomous without human intervention anywhere in the flow

1. An agent that crawls competitor websites / apps , compares its features or products with ours and then sends a gap identification emailer - what features we should build , what products we should keep

- Is it repetitive
- Does it involve language understanding or generation
- Is it a multi step decision making process
- Do you trust an AI system 100% to execute the entire task ?

All these approaches are not mutually exclusive for a given business goal. A single system can use mix of them . Some parts might require high control , while others can benefit from high autonomy. Think of it as options you can apply to differents parts of the business problem

Agentic AI vs AI Agents :

What are tools in AI ?

In the context of Agentic AI , tools are external capabilities that LLMs can invoke things like -

- APIs
- Database queries
- Internal services
- Third party systems
- Internal Codebase

This turns LLms from something that just talks to sth that acts.

Example : End to End Agent works using tools :
Problem Statement : Hey Agent , can you inform all the customers that their orders are delayed and offer a new delivery time. (all the customers who were supposed to get a delivery today , check their order status , wherever order status is delayed from expected, draft an email with an alternate slot, alternate slot decide based on teh current order status)

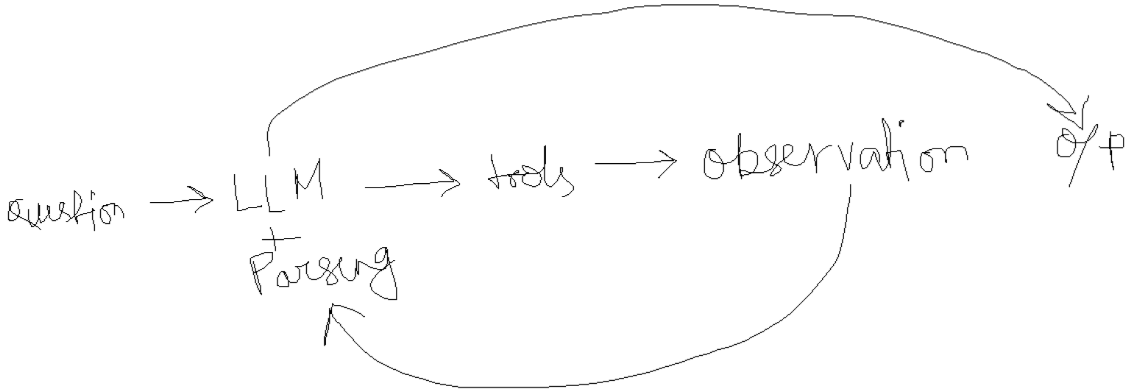
Input : Human question : send email to john on XYZ topic

1. **Input** : Human Question : “ Hey Agent , can you inform all the customers that their orders are delayed and offer a new delivery time.”
2. **Planning :** The LLM breaks it down
    a. Find customers who are supposed to get their order delivery today
    b. Check the order status from the table t1 for those
    c. If the order status shows delayed , the check the revised delivery time (ETA) from T2
    d. Based on the ETA received , generate couple of more delivery slots
    e. Draft emails to all teh customers with order details , the status and revised ETA
    f. Send the email
    g. Log interaction
3. **Tool Calls :**
    a. get_todays_customers/orders(date = X) ->this tool tells us which are teh customers who have a pending order on a given date
    b. get_order_status(order_id = Y)

c. get_available_slots(order_id = Z which is delayed)
d. send_email()
e. log_event()

While developing agents, you will register the tools with the Agentic system in a way telling LLMs what tools you can specifically use . LLM is constrained by using those tools only which are registered. LLMs job is match planned task with the tool description in the runtime to figure out for a given task which tool to invoke when. with certain details - **name**, tool **description (tells the model when to use it)** , tool parameters , output structure,

4. Text generation : LLM composes the email . Input of the email will come from teh output of the tool calls
5. Execution : Send_email()



VISUAL MODEL

1. User asks a question
2. LLM understands what needs to be done and plans its next step (understand intent + planning)
3. A parser which is often integrated within LLS converts what needs to be done to a structured format get_order_Status()
4. Agent calls the right tool depending on what task it has to do and what tool it has available with itself
5. Tool returns a result - Observation
6. LLM looks at the result , decides what is missing - what comes next
7. This loop continues till the time LLM realizes all the steps are done and it has enough information to generate final output

LOOP = **REASONING + ACTIONING + REFLECTION (agent)**

CrewAI , Autogen, Langchain - frameworks use exactly this structure and give us wrappers

How planning is done?

1. Latest LLMs are starting to behave more like LRMs (training has been moved to planbased from token based partially).
2. Memory (Seed memory , iterative memory enhancement via interactions where human adds information to the memory on what planned step(s) missing or added)

============

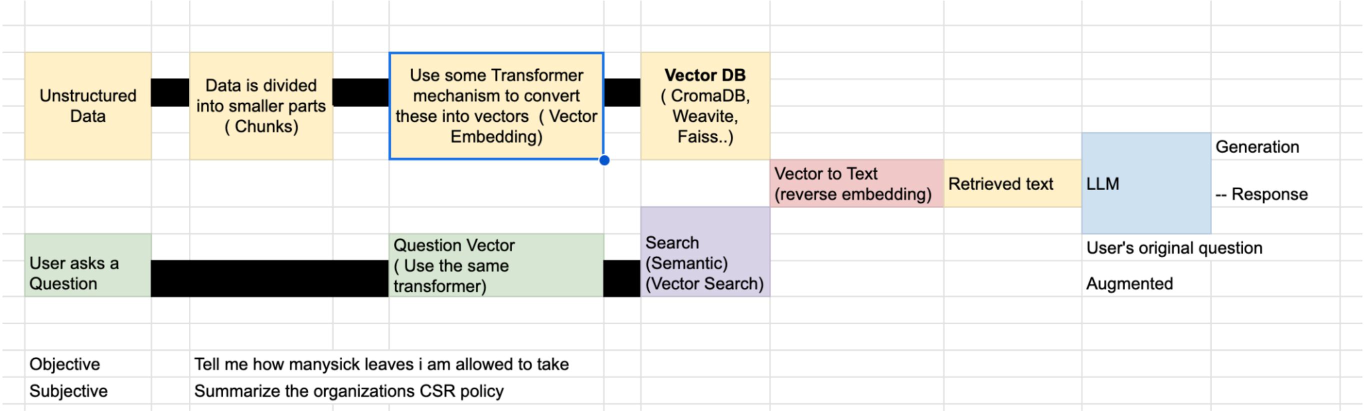
We looked at how tools help AI Agents interact with real world systems - send emails, file jira tickets, trigger APIs

What if the model does need to act ? It just needs to access the right information . That is the case in many of our organizational usecases:

1. Internal documents spread across teams
2. Policy PDFs
3. Research articles / journals
4. Customer interactions in CRM notes
5. Email dump

Tools will not help if you want to find the relevant information from these information dumps . That is where **RAG** (Retrieval Augmented Generation) comes in.

Why not just give all the organization's data to the model directly ? . Technically you can. Only problem is , that LLMs can only process limited amount of text at a time (input token limit) . Even within that limit , they struggle to isolate noise vs signal - its responses start to hallucinate.



Unstructured Data

Data is divided into smaller parts (Chunks)

Use some Transformer mechanism to convert these into vectors (Vector Embedding)

Vector DB (CromaDB, Weavite, Faiss..)

Vector to Text (reverse embedding)

Retrieved text

LLM

Generation

-- Response

User asks a Question

Question Vector (Use the same transformer)

Search (Semantic) (Vector Search)

User's original question

Augmented

Objective	Tell me how manysick leaves i am allowed to take
Subjective	Summarize the organizations CSR policy

Give me all the count of IP clauses in contract docs I have

Advantages of RAG :

1. Fine tuning vs RAG :
   - Fine tuning will always have better performance (A healthcare org writes its emails or policies in a specific format - tonality /lingo.Knowledge becomes stale
   - Org data changes very dynamically . And u need fact based answers - RAG
2. It narrows the scope on which LLM operates . Ability to get the answer wrong is limited

Challenges :

1. **Chunking mechanism :**
2. Transformer usage (transformer that are trained on big data will be better) - non issue
3. What **search algorithm** are you using and what vector Db are u using

RAG:

1. **Text to SQL** [Ask a question english → data from database]
   1. LLM : Which has ability to write SQL already
   2. Unstructured data / Vector DB : Schema information (table is all about , column name, column types) , User questions and queries
   3. Action - SQL Connector to hit the query
2. Enterprise Knowledge Search

<u>Agentic RAG :</u>

- RAG - 1 query , 1 retrieval , 1 response
- Example: Say you are developing an agent which is a **deal assistant** b2B dsetup:
  - **Pull the customer cRM history**
  - Retrieve current pricing for similar customers in this market segment
  - **Look up the regional sales teams field inputs**
  - **Reference to past contracts**
  - Generate a custom proposal
  - **Double check facts with companies contract policies**
  - Create the proposal and send a version for human to review

Makes the decision making very efficient

<u>MCP (Model COntext Protocol):</u>

LLMs - can call tools , retrieve context from org knowledge base with RAG.

MCP is a standardized way to give LLM everything it needs to reason and respond.

It is a protocol - TPC/IP, HTTP, — standardization

Anthropic - Network effect kicked in

<u>Agentic Design pattern : Single Agent vs Multi Agent Framework:</u>

Single Agent :

One agent handles everything taking user input, reasoning, planning, action ,

When to use : very focussed task , narrow scope , it cannot have parallel execution - u should use single agent

Benefits : faster response time, no coordination required

MultiAgent System:

Multiple agents collaborate and communicate to complete a task. Remember every agent in a multi agent system specialize on doing a very focussed task

When to use: Complex workflow and parallelism is possible

Benefits: Scalable model , improves the agent performance , supports distributed system

In a multi agent system there is a orchestrating agent that is there …

Not there in case of single agent system agent

Always any multiagent system can be configured as single agent system :

- Time taken (lesser of reason) [parallel work can be done in parallel ..]
- **Narrow focussed specific roles** are given to agents in a multiagent system and that turns out to improve the performance significantly

Usecases:

1. Personalized Email response agent : Respond to customer service emails based on templates, tone and history . **(Single Agent)**
2. Travel Planning assistant : Plan a multi country itinerary within a given budget including visa, flights , hotels and sightseeing **(Multi Agent)**
3. Ai Health assistant : Monitors patient health statistics - bp, hrv,spo2…, suggest actions , alert doctors and track medication . **(Multi Agent)**
4. Code generation and execution agent : Write an execute code snippets based on prompts . **(Single Agent)**
5. E Commerce Customer Assistant : Answers user query for a given product and it refers to product catalog , customer reviews and gives pros and cons for to the customer (Multi Agent) RAG

Agentic AI Memory :

- **Duration based** (Short term memory , Long term memory)
- **Content based** (Episodic Memory , Procedural Memory , Semantic memory)

Episodic Memory - User - Agent interaction stored in memory (session level data)

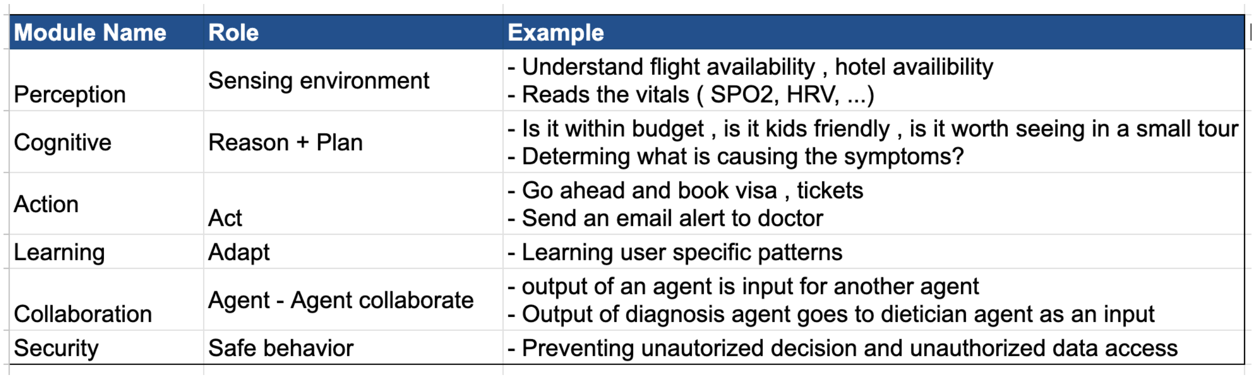
Procedural Memory : procedures are also stored (Travel Planning assistant flowchart

- 1) airbnb_search 2) if available , then calculate cost and compare with budget compare_budget() 3) if budget exceeds, ask user for another date ..

Semantic Memory : It is not built over time. It is hardcoded information you provide. In an agent scenario [you are part of XYZ organization , in this market segments, your competitors are …..]]Fact based memory (user is 38 yr old male , he works in XX, he loves to do etc).

Short term memory : Within a session , the conversation is stored . Get better context of the conversation . Helps in maintaining agent state.
Long term memory : Memory that are generated over the episodic memory . (personalize ur answer)

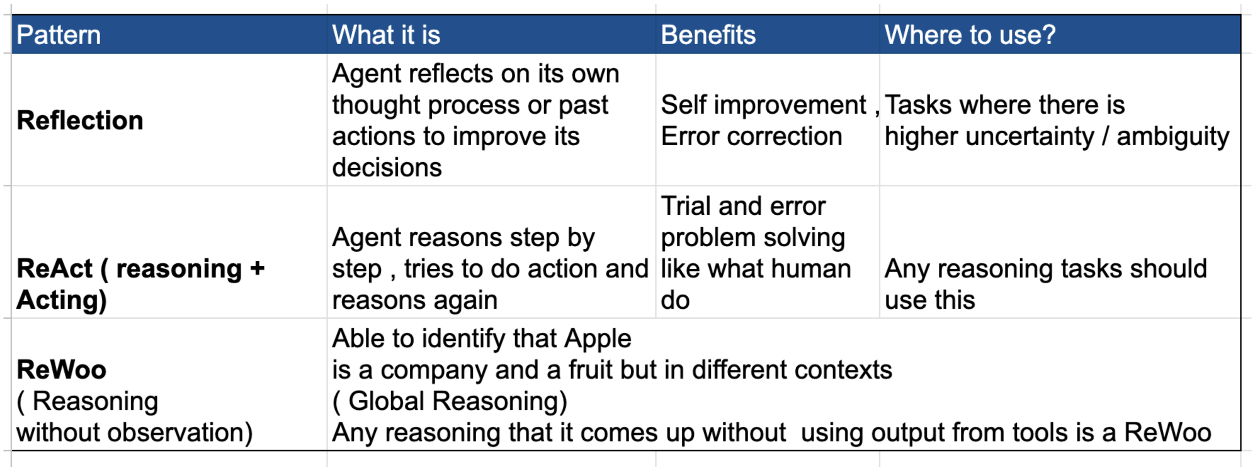
Different Modules of Agentic AI :



Module Name	Role	Example
Perception	Sensing environment	- Understand flight availability , hotel availibility - Reads the vitals (SPO2, HRV, ...)
Cognitive	Reason + Plan	- Is it within budget , is it kids friendly , is it worth seeing in a small tour - Determing what is causing the symptoms?
Action	Act	- Go ahead and book visa , tickets - Send an email alert to doctor
Learning	Adapt	- Learning user specific patterns
Collaboration	Agent - Agent collaborate	- output of an agent is input for another agent - Output of diagnosis agent goes to dietician agent as an input
Security	Safe behavior	- Preventing unautorized decision and unauthorized data access

Important Concept :

Example :
Can you tell me why I am sweating ?
Reasoning : -> I will read ur vitals and see if BP is high (reasoning with observation / evidence)
Reasoning : -> generally people sweat when it is very hot outside or people do some exercise



Pattern	What it is	Benefits	Where to use?
Reflection	Agent reflects on its own thought process or past actions to improve its decisions	Self improvement , Error correction	Tasks where there is higher uncertainty / ambiguity
ReAct (reasoning + Acting)	Agent reasons step by step , tries to do action and reasons again	Trial and error problem solving like what human do	Any reasoning tasks should use this
ReWoo (Reasoning without observation)	Able to identify that Apple is a company and a fruit but in different contexts (Global Reasoning) Any reasoning that it comes up without using output from tools is a ReWoo		

Zomato - Customer Support Bot :
LLM - which is trained on food delivery apps (open source apps)
Why is my order delayed :

1. Can you check order status
2. Or it could be raining
3. Or it could be restaurant did not accept

Why is my order delayed :

1. Tool access
2. Rain driver got delayed

<u>Prerequisite for Coding classes:</u>

1. Gemini API key , OpenAI API key ($5 dollar)
2. Successfully run "https://colab.research.google.com/drive/1Iq5gfnQzbbubNuvi0SXNjw7fp1Qz94Dg#scrollTo=nBRsFYIey-vZ" - this is to check key is working
3. Download Visual Studio in ur system / Cursor
4. Learn how to create virtual environment in your system and activate the virtual environment . Try out in terminal to create and activate virtual env

Single agent system using no framework - use tools , use streamlit (complex system)

Request from the class: Call out Agenda before hand

========================== Agenda for 19th Jul=========================

1. Coding prerequisites:
    a. How to create virtual environment + why ?
    b. What all keys will be needing for the course
    c. What packages to install (Langchain + …)
    d. file structure
2. Basics of Langchain :
    a. How using langchain we can call different LLMs
    b. How using langchain we can give the prompts
    c. Observability of langchain applications via Langsmith
    d. Langchain vs Langgraph - when to use which one ?
3. RAG Pipeline / Langchain pipeline :
    a. Data ingestion/ Document loader
    b. Chunking / data transformation
    c. Embedding
    d. Vector Store
    e. Chains to pass on the retrieved context to LLM
    f. RAG end to end workflow
4. Q & A

Groundrules :

1. Do not code simultaneously when I am coding
2. Q & A - hold on to it

Coding Hygiene:

1. **Creating and activating the virtual environment**
   (RAG app1 , RAG app2 , MultiAGent3) - may require different sets of libraries ..if you do not create virtual env for each of your projects there is high chance that the libraries will conflict
   Using conda, uv , python/python3

```
GOOGLE_API_KEY =

GROQ_API_KEY =

OPENAI_API_KEY =

HF_TOKEN =

LANGCHAIN_API_KEY =

LANGCHAIN_TRACING_V2 = "true"

LANGCHAIN_PROJECT = "Agentic Live"
```

====
What are the differences between Langchain and Langgraph ?

Langchain has certain building blocks (objects) - Prompts , LLMs, Chains, Tools, Memory , Retrieval , Output and Agents

All the building blocks can only be stacked unidirectionally and sequentially

Prompt -> LLM -> OutputParser
Userinput -> Retrieval -> LLM-> Answer
Agent -> Tool -> LLM

Single Agent applications (DAG)

Applications : 1) **RAG** 2) Summarization 3) Chatbot / Q&A

Langgraph :
Building block for langgraph - Tasks , Nodes , Edges and Graph

Multi agent applications (conditional if else , iterations , bidirectional flow , information sharing)
Usecase : hey can you schedule a calendar invite based on availability of A.B, C people

=====
Vector DB :

```
# 4. Vector DB
# In Memory  (RAM) - data is lost when the session restarts ( FAISS, CromaDB)
# Local (Hard disk) - here u store teh database as a file ( FAISS, CromaDB)
# Cloud hosted - hosted by vendors - scaliing , multi team access , APIs (Pinecone,
Weaviate, Milvus)
# AWS - Opensearch
# Azure - Ai Search
# GCP vertex AI vector search

# Weaviate has a local support as well as cloud support
# Pinecone has only cloud support
# FAISS, Croma has only local support

# Every vector DB needs to have index

# Semantic Search
# - Exact - FlatIndex
# - Approximate - Grapg based and Inverted index based
# Hierarchical Navigable Small World (HNSW) - Graph based
# and Inverted File with IVF Flat Indexing (IVFFlat)#

# Similarity match
# Cosine Similarityy
# Euclidean similarity ( L2)
# Jaccard similarity
```