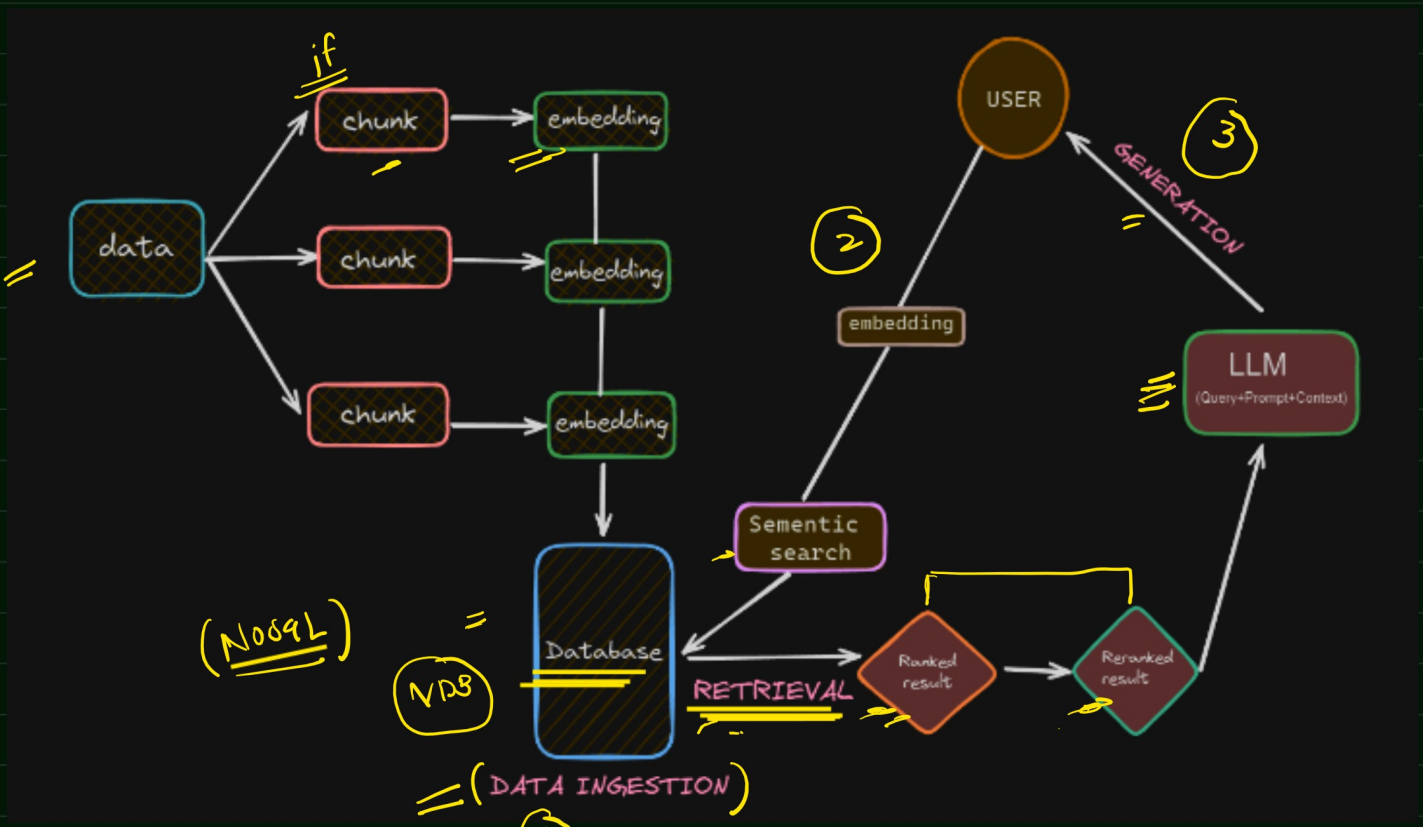
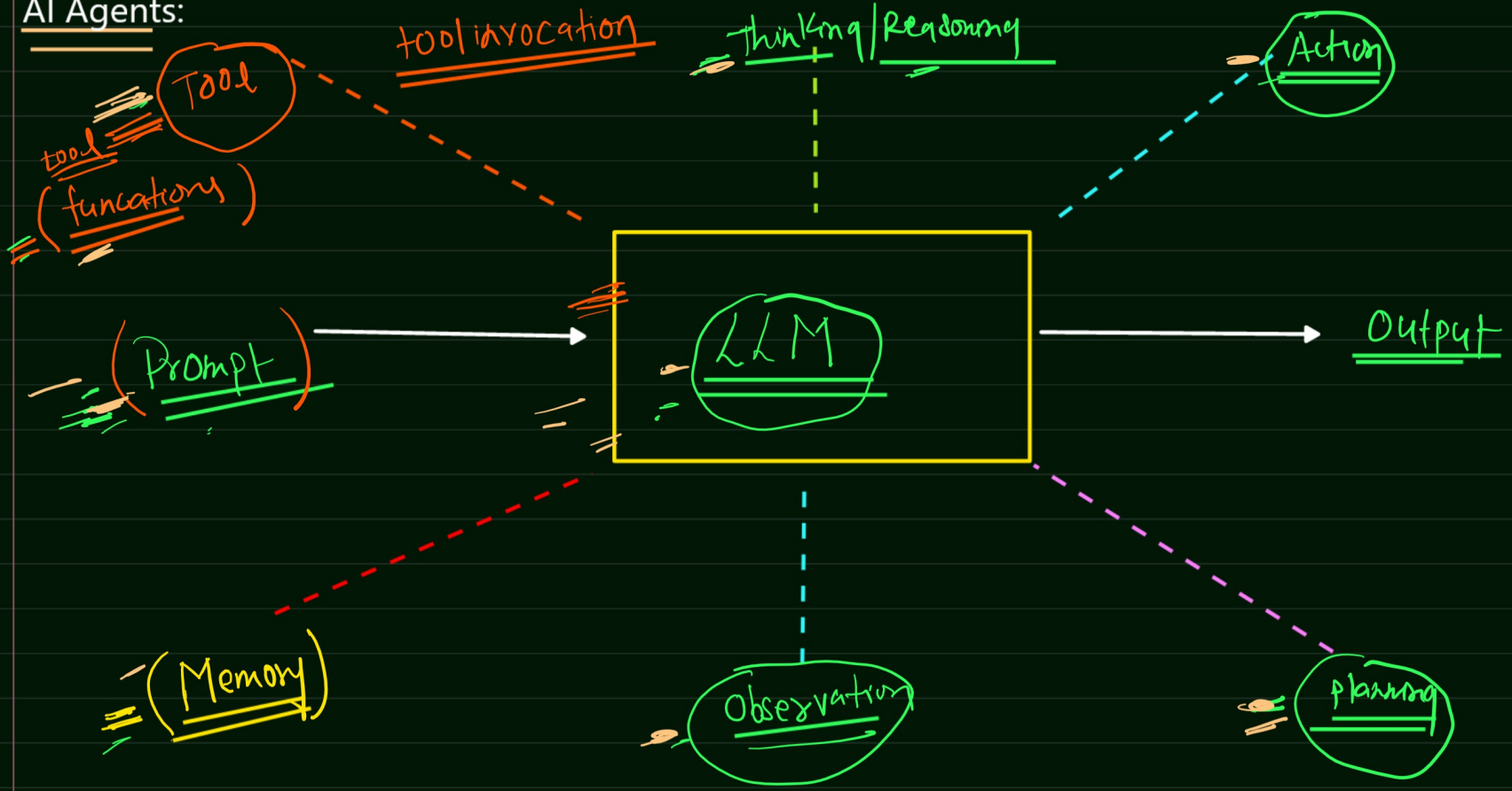
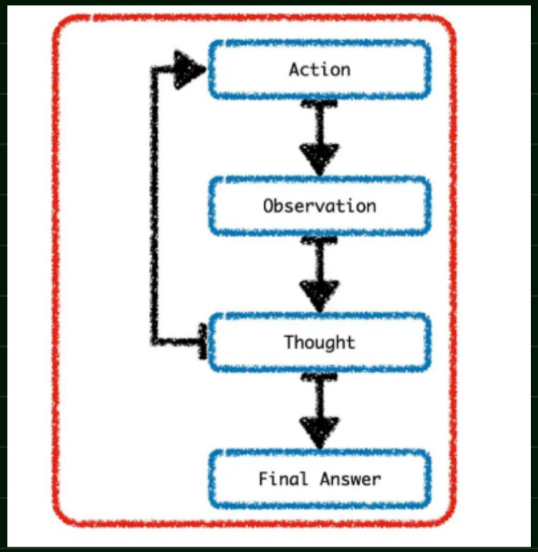
1. Finetuning
2. Rertieval-Augmented Generation

RAG.
MMRAG

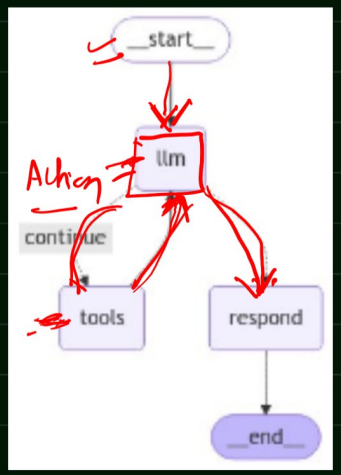


AI Agents:

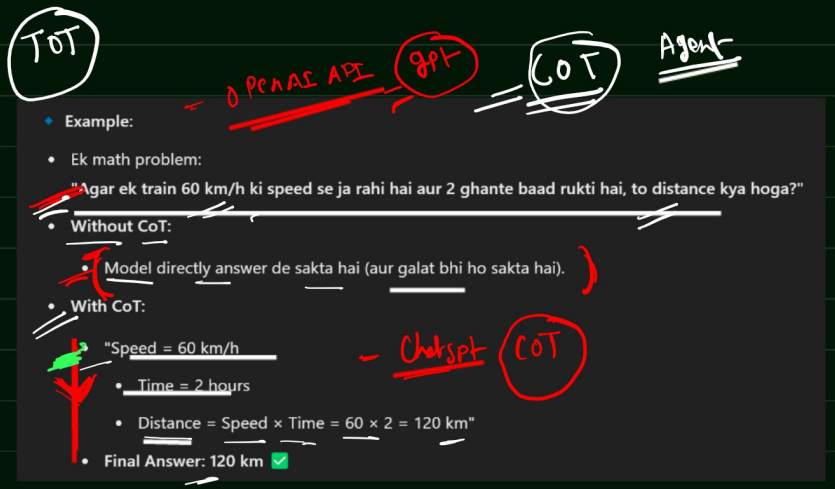




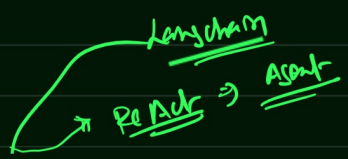
Agent
LLM
prompt
tool calling
(planning)

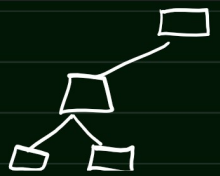


Feature	Retrieval-Augmented Generation (RAG)	AI Agents
Definition	Enhances LLMs with external knowledge retrieval (e.g., from a vector database).	Autonomous systems that use LLMs and tools to make decisions and take actions.
Core Mechanism	Fetches relevant documents from an external source and uses them for response generation.	Uses reasoning, decision-making, and tool execution to complete complex tasks.
Dependency on External Data	Strongly relies on retrieved documents for accurate responses.	Can retrieve, analyze, and act on data dynamically.
Interactivity	Mostly passive—fetches documents and generates answers based on them.	Interactive—can call APIs, query databases, and refine responses iteratively.
Memory	Limited to retrieved documents.	Can maintain memory across multiple interactions and refine strategy dynamically.
Autonomy	Not autonomous; executes a single retrieval and generation cycle.	Autonomous; can call different tools, modify goals, and make iterative improvements.
Example Use Case	Searching for technical documentation and answering questions.	Booking a flight, writing reports, executing API calls, automating workflows.



Step	Component	Description
1	**Thought**	Identifies possible moves
2	**Expand Thought Tree**	Generates all possible legal moves
3	**Thought**	Evaluates opponent's best responses
4	**Expand Thought Tree**	Thinks ahead for multiple moves
5	**Evaluation & Pruning**	Scores moves and removes bad branches
6	**Final Thought**	Chooses the best move based on analysis
7	**Action**	Executes the best move



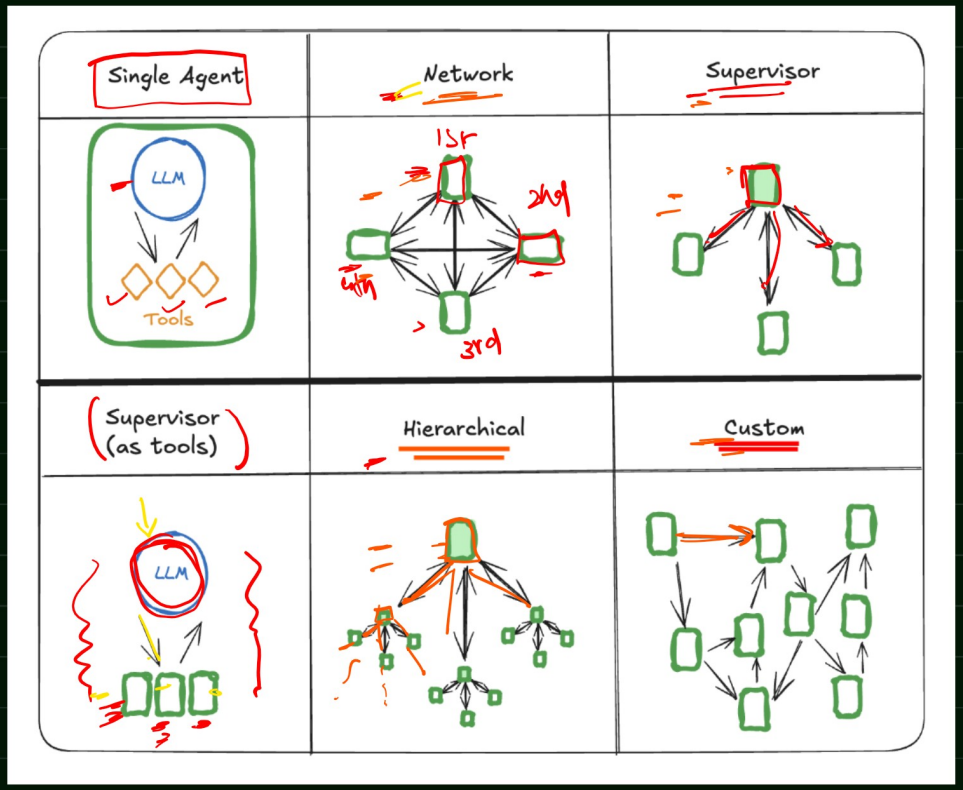


Example: Web Scraping Agent using ReAct

Task: Fetch the latest stock price of Apple (AAPL) from a financial website.

Step	Component	Description
1	Thought	Determines where to find stock price
2	Action	Checks if a function exists to get stock price
3	Observation	No function available, so another approach is needed
4	Thought	Decides to use web scraping
5	Action	Sends a request to scrape Yahoo Finance
6	Observation	Extracts relevant stock price information
7	Thought	Confirms extracted price is ready to be returned
8	Action	Returns the final stock price

Feature	Tree of Thought (ToT)	Multi-Agent System
Definition	A structured reasoning approach exploring multiple paths	Multiple agents working together on tasks
Execution	Can be done by a single AI agent	Requires multiple interacting agents
Parallelism	Usually sequential or guided search	Can be parallel or independent
Decision Making	Evaluates multiple thought paths	Different agents contribute to the decision



Type	Structure	Best For
Single Agent	A single LLM with direct access to tools.	Simple automation tasks where one AI handles everything.
Network	Multiple agents connected in a decentralized way, communicating equally.	Collaborative AI, where agents share information and make collective decisions.
Supervisor	A central agent (supervisor) controls and directs other agents.	Task delegation and centralized decision-making.
Supervisor as Tools	The LLM treats multiple agents as tools, calling them when needed.	Workflow automation where different AI tools handle specific functions.
Hierarchical	Multi-level AI control, where a top-level supervisor manages sub-supervisors.	Large AI-driven systems with structured control (e.g., enterprise AI).
Custom	A dynamic, flexible agent structure with complex interactions.	Adaptive AI systems that adjust based on changing needs (e.g., LangGraph).

Multi-Agent Type	Use Case	Why This Structure?
Hierarchical	**Self-Driving Cars**	Top-down control from planning to execution.
Supervisor	**AI Customer Support**	A central AI directs specialized agents.
Network	**Stock Market Trading**	Decentralized knowledge-sharing agents.
Custom	**Fraud Detection**	Adaptive & evolving multi-agent collaboration.

Hierarchical Multi-Agent System
Use Case: Autonomous Vehicles (Self-Driving Cars)

How It Works:
Top-Level Supervisor (Central AI Brain) → Plans high-level driving strategy.
Mid-Level Agents (Perception, Planning, Navigation) → Handle specific tasks like object detection, route planning, and speed control.
Low-Level Agents (Sensor Data Processing & Actuators) → Work with cameras, LiDAR, and control braking, acceleration, and steering.

Example Workflow:

Car approaches an intersection
Perception Agent detects a red light
Navigation Agent decides to stop
Actuator Agent applies brakes smoothly
Why Hierarchical? Because decision-making is structured, from high-level planning to low-level execution.

Supervisor Multi-Agent System
Use Case: AI Customer Support in a Large Company (Chatbot & Agents)

How It Works:

Supervisor Agent (Main AI Bot) → Decides which sub-agent should handle the user request.
Sub-Agents (Finance, Tech Support, HR, Sales, etc.) → Specialize in different areas.
Action Agents → Perform tasks like retrieving order details, resetting passwords, or scheduling meetings.
Example Workflow:

Customer asks, "Can I get a refund for my order?"
Supervisor Agent detects it's a finance query.
Finance Agent retrieves order info.
Refund is processed, and confirmation is sent to the customer.
Why Supervisor? Because a single AI directs sub-agents to handle specific tasks efficiently.

Network Multi-Agent System
Use Case: AI-Powered Stock Market Trading

How It Works:

Multiple agents monitor different markets (Crypto, Forex, Stocks, Commodities).
Each agent analyzes trends, predicts movements, and shares insights with others.
Decisions are made collectively, based on insights from all agents.
Example Workflow:

Crypto Agent detects a Bitcoin price surge.
Stock Market Agent checks correlations with tech stocks.
Decision Agent predicts a market shift and suggests buying Google shares.
Why Network? Because all agents are connected and share knowledge without a single supervisor controlling them.

1. AutoGen (Microsoft)

Class	Purpose
`Agent`	Base class for all agents.
`UserProxyAgent`	Acts as an interface between the user and the AI agents.
`AssistantAgent`	Standard AI agent that executes tasks (calls LLM, APIs, etc.).
`GroupChatManager`	Manages multiple agents communicating together.
`ConversableAgent`	An agent that can chat and reason with others.
`CodeInterpreterAgent`	Executes code and evaluates outputs.
`LLMAgent`	Specialized LLM agent for language tasks.

2. CrewAI (Role-Based Multi-Agent System)

Class	Purpose
`Agent`	Base class for an agent with a defined role.
`Task`	Defines what an agent should do.
`Crew`	A team of agents that work together.
`Process`	Handles task execution within a crew.
`Memory`	Stores past interactions.

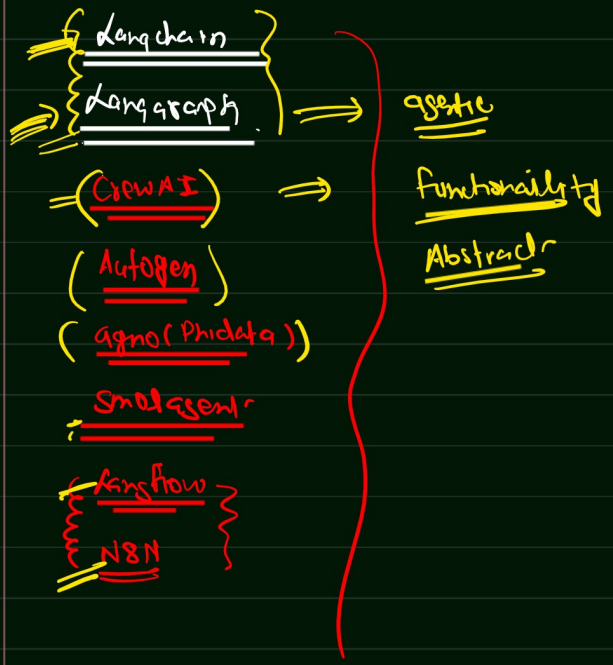
3. LangGraph (Graph-Based Agent Workflows)

Class	Purpose
`Graph`	Represents the overall workflow of agents.
`Node`	Represents a single step in the workflow.
`State`	Stores agent memory and intermediate results.
`Condition`	Enables branching logic based on agent decisions.

4. PhiData (Agentic Data Framework)

Class	Purpose
`DataAgent`	Base class for data agents.
`SQLAgent`	Converts natural language to SQL queries.
`ReportAgent`	Summarizes data into readable reports.
`Pipeline`	Manages the execution of multiple data agents.
`DataStore`	Connects to a database (SQL, Snowflake, etc.).

- If you want **LLM agents that collaborate dynamically** → **AutoGen**
- If you want **structured, goal-driven agents** → **CrewAI**
- If you want **AI-powered data workflows** → **PhiData**


Langchain
Langgraph
CrewAI
Autogen
agno (Phidata)
Smolagent
Langflow
N8N
agentic
functionality
Abstract