Bluescarf

Agentic Architectures and Frameworks

Agentic team

August 2024

Introduction 3
List of Single Agent Frameworks 3
ReAct 3
Use case scenario: 4
LATs: 4
Use case scenario: 7
BDI: 7
Use scenario: 8
OODA: 8
RAISE: 9
Use case scenario: 10
ReWOO: 10
Use case scenario: 11
Advantages of these frameworks 11
List of Multi Agent Frameworks 13
DyLAN: 13
AgentVerse 13
Use cases: 15
Development Framework: 15
LangChain 15
LangGraph: 18
CrewAI: 20
Microsoft Semantic Kernel: 21
Potential Use Cases: 23
Job Screening 23
Potential agent-based solution: 23
Network Optimization: 24
Potential agent-based solution: 24
Future Prospects 25
Appendix 27
Architecture 27
Framework 27
Hallucinations 27

Introduction

In the rapidly evolving field of artificial intelligence, the design and development of autonomous agents is a key area of focus. These agents, capable of performing complex tasks with minimal human intervention, rely on well-defined architectures and frameworks to function effectively.

When examining agent architectures within the context of an entire agentic system, these architectures can be seen as frameworks. The overall blueprint or plan that defines the interaction of various components constitutes the architecture.

AI agent architectures typically fall into two types of frameworks: single-agent and multi-agent.

The single-agent framework relies on a single LLM-based agent equipped with a system prompt and, potentially, additional tools for performing tasks. The agent operates independently, without feedback or guidance from other AI agents, although it may include mechanisms for human feedback or guidance.

In contrast, multi-agent frameworks involve two or more agents, each utilizing different sets of large language models, tools, and plugins. In this approach, a robust feedback mechanism among the agents is usually in place to enhance coordination and performance.

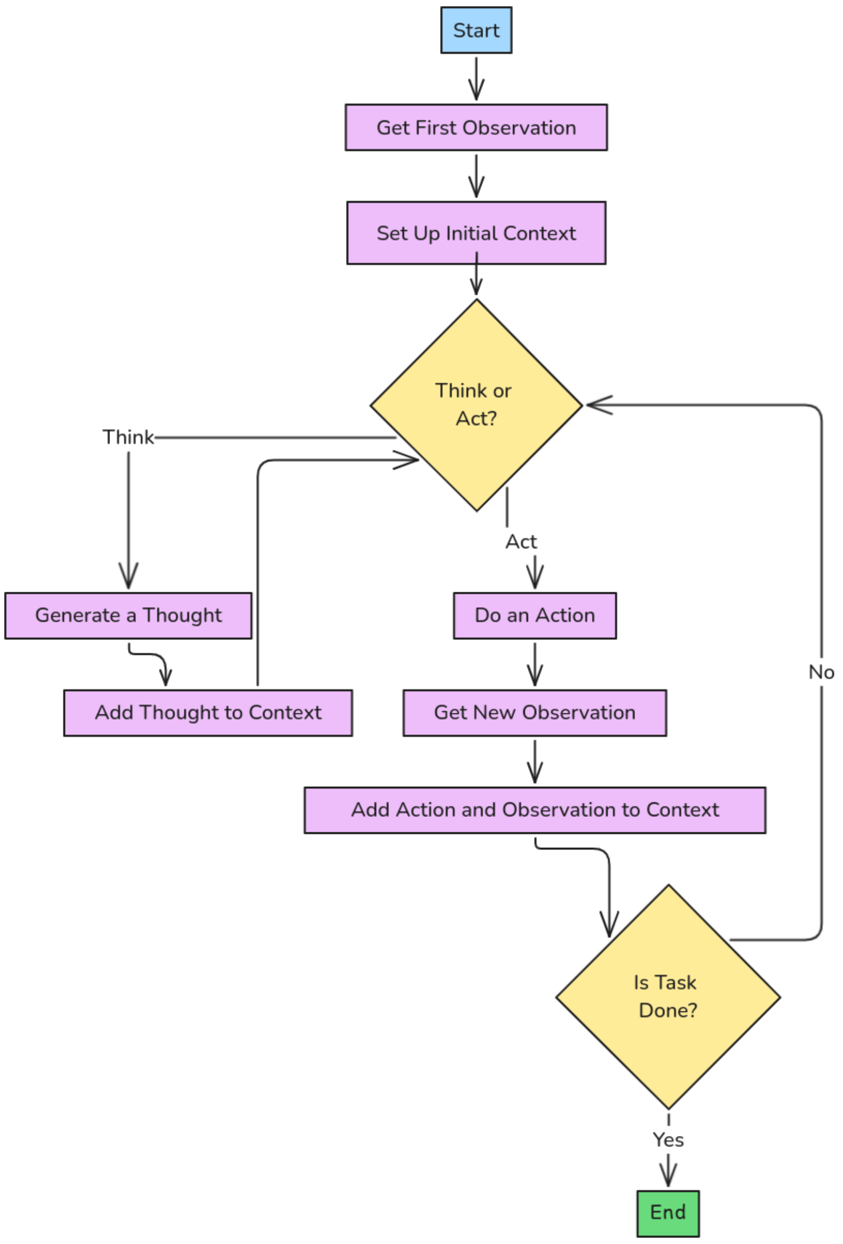
List of Single Agent Frameworks

The following is a list of various frameworks used to build AI agents:

- ReAct
- LATs
- BDI
- OODA
- RAISE
- ReWOO

ReAct

ReAct, short for "Reasoning and Acting," is a framework designed for constructing AI agents. The ReAct process begins with the agent receiving an observation from the environment, which establishes the context. Based on this context, the agent generates a thought, which then informs its decision on the next action. If the task is not yet complete, the agent repeats this cycle of observing, thinking, and acting until the task is successfully accomplished. The figure below visually represents this process.



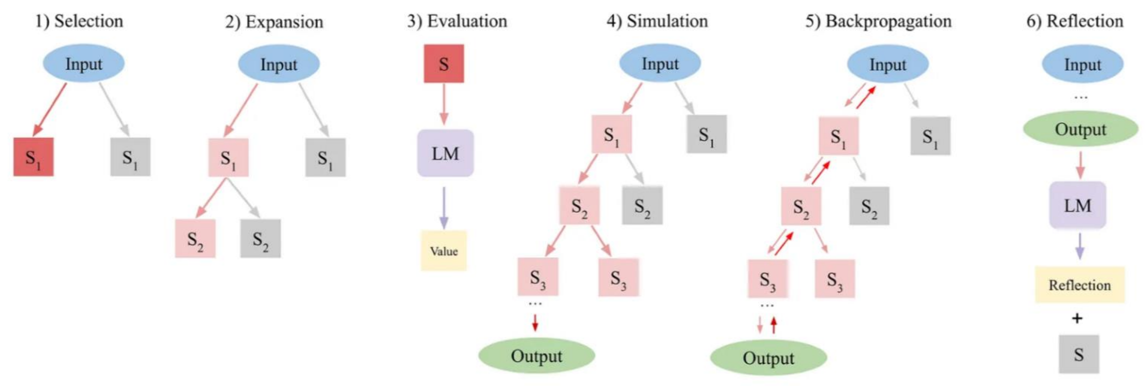
Use case scenario:

Ali wants to know if his horse is healthy. He asks the agent, "My horse weighs 400 kgs and is 2 years old. Tell me if it is healthy or not."

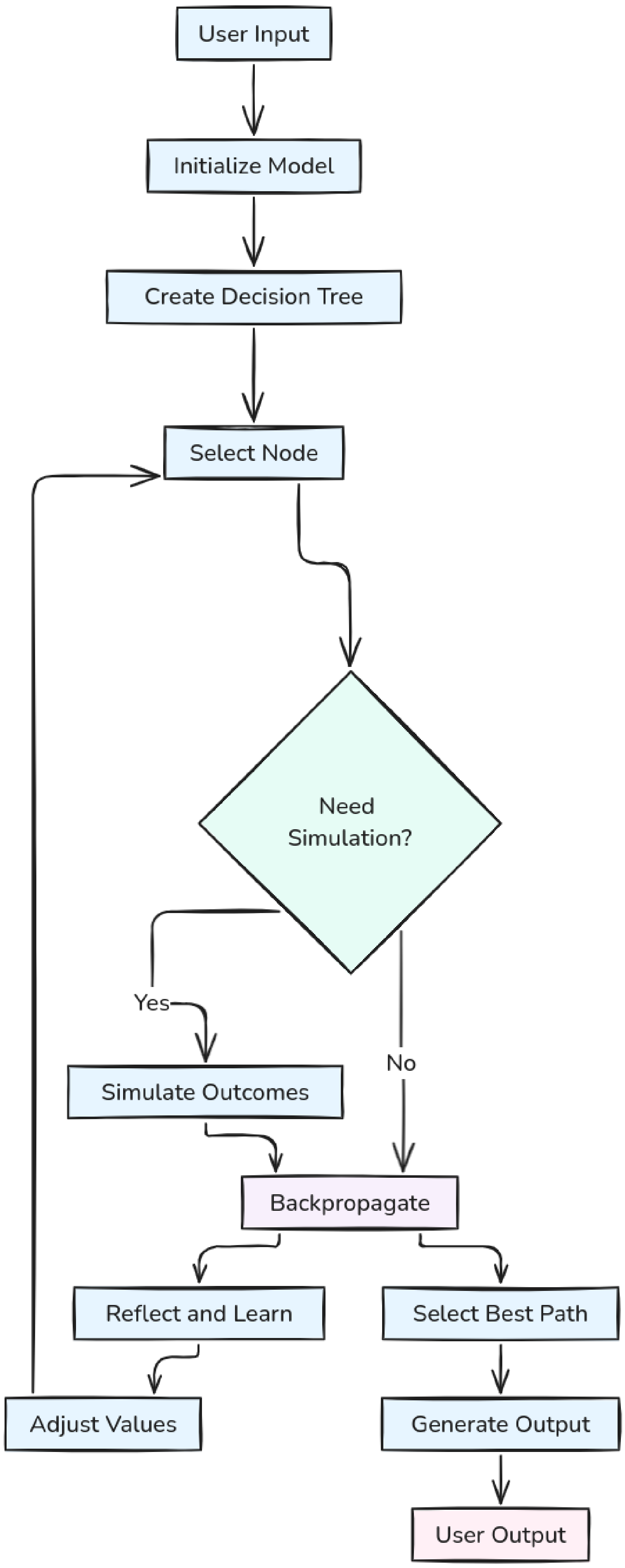
An agent built on the ReACT (Reasoning and Acting) framework is well-suited for this scenario. The agent first searches the web for average horse healthy horse weight and then reasons through the gathered data. It compares this information with the specifics of Ali's horse and provides a conclusion of its health.

LATs:

LATs, short for “Language Agent Tree Search” is framework that combines reasoning, acting and planning in a single framework as methods like CoT and ReAct lack the ability to plan and create multiple paths to a solution of the problem. LATs achieves this by making use of the Monte Carlo Tree search algorithm.



https://arxiv.org/pdf/2310.04406


User Input
Initialize Model
Create Decision Tree
Select Node
Need Simulation?
Yes
No
Simulate Outcomes
Backpropagate
Reflect and Learn
Select Best Path
Adjust Values
Generate Output
User Output

Use case scenario:

XYZShop needs to assist customers in efficiently finding and purchasing specific products that match their requirements. For example, a customer is looking for a 3-ounce bottle of bright citrus deodorant for sensitive skin, with a price lower than 50rs. The search process involves scanning through various options, comparing them, and making a decision based on the product's details such as price, scent, and size.

An agent built on the LATs (Language Agent Tree Search) architecture can be particularly effective in this scenario. The tree search module of LATs allows the agent to systematically explore multiple product options, while the language processing capabilities of LLM enable it to understand and interpret the customer's requirements accurately. This combination allows the agent to efficiently navigate through the available products, select the most relevant options, and guide the customer to the best purchase decision, thereby enhancing the overall shopping experience.

BDI:

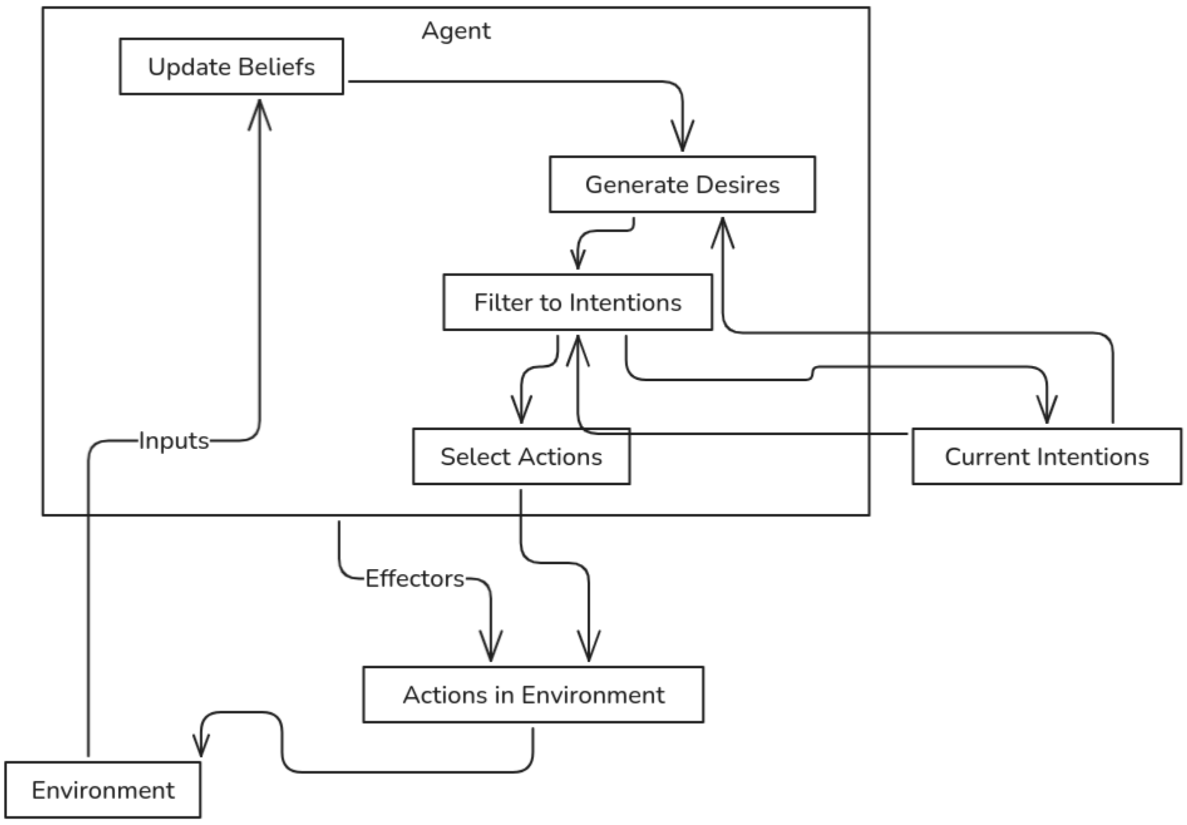
BDI short for Belief, Desire, Intention is a framework for building intelligent agents that can reason. The idea is based on Michael Bratman theory which states that reason is based on three things beliefs, desires and intentions.

An agent which uses these concepts have three components.

Belief: which is the model of its environment basically what it believes to be true and as it is not knowledge so doesn't have to be necessarily true.

Desire: is the ideal state for the agent to be in. It can be realistic or not as it totally based on wishes and desires. These are the things we would like to see, whether achievable or not, to be accomplished in the future.

Intentions: the subset or desires that are based on its beliefs. The intentions are the goals which the agent wants to achieve soon.



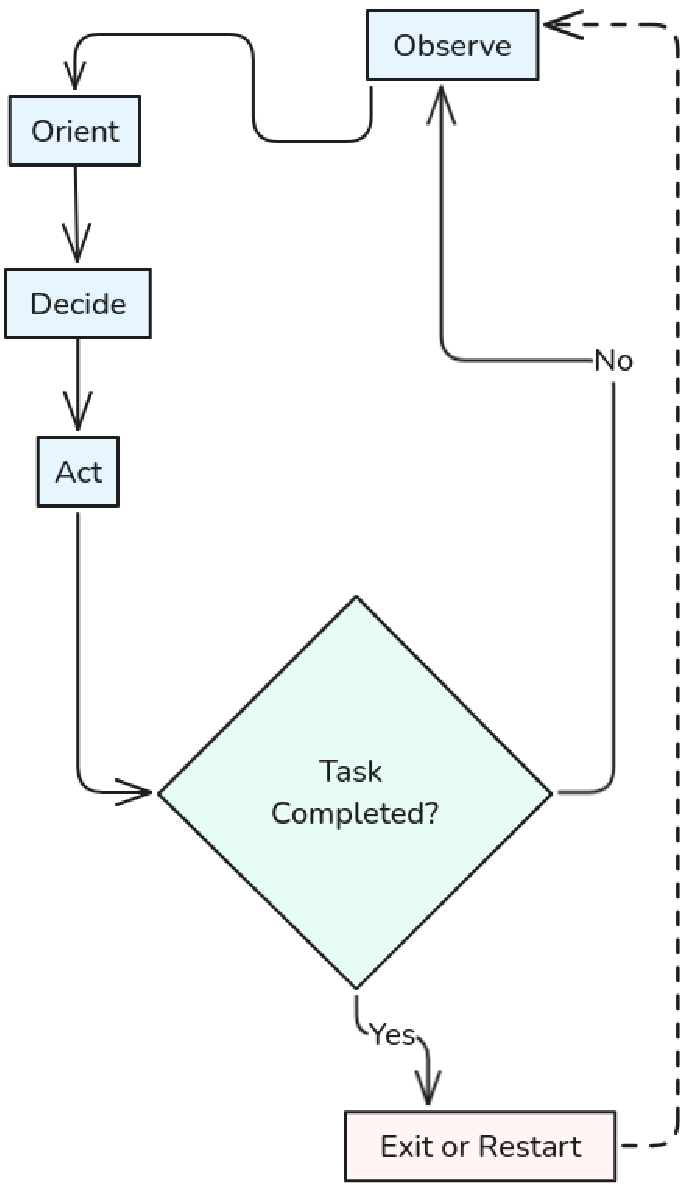
Use scenario:

Ali is planning to train his horse for an upcoming competition and wants to create a training schedule. He needs an agent to help him design a regimen that aligns with his goals.

An agent built on the BDI (Belief, Desire, Intention) framework is ideal for this task. The agent first forms beliefs about the horse's current condition and training environment, such as its weight, age, and stamina. Then, it considers Ali’s desire, which is to prepare the horse for competition. Finally, the agent translates these desires into actionable intentions by creating a realistic training schedule based on its beliefs about what is achievable. This approach allows the agent to balance Ali's goals with the horse's current capabilities, ensuring a well-rounded training plan.

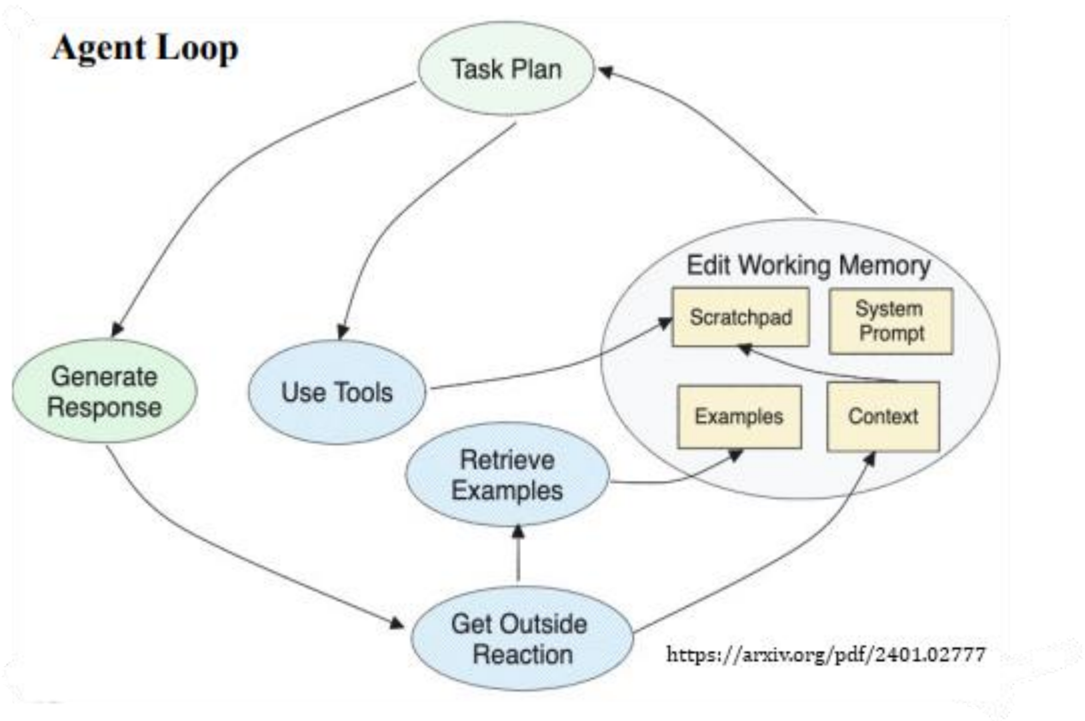
OODA:

OODA (Observe, Orient, Decide, Act) was a model developed by John Boyd, a military strategist. The observe stage involves collecting data from the environment, in case of AI agents it is collecting user query. Then the agent Orients itself with the information by collecting the context and interpreting the information. In the decide stage the agent decides what actions should it take to solve the task at hand by utilizing the information at hand. In the acting stage the agent simple executes or performs the task.



RAISE:

Inspired by ReACT, RAISE (Reasoning, and Acting through Scratchpad and Examples) tackles the issue of memory management in an agent. It integrates five key modules to enhance user-agent interactions. The Dialogue module serves as the communication interface, while the LLMs act as the agent's brain, handling tasks like perception, planning, tool usage, and summarization. The Memory module stores vital information, including conversation history and task trajectories, to inform future actions. The Tool module enriches the agent's capabilities by incorporating external knowledge sources. Lastly, the Controller orchestrates these modules through a loop of perception, planning, and tool execution, ultimately guiding the agent's decision-making and response generation processes.



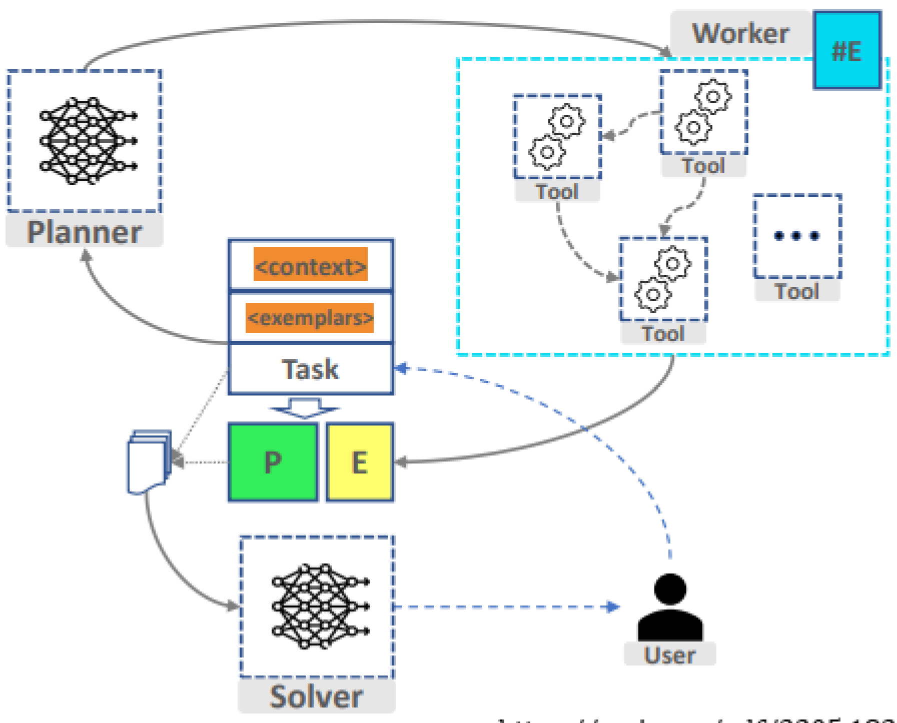
Use case scenario:

ABC needs a chartered accountant (CA) to manage their financial operations. The primary responsibilities of a CA include providing strategic financial advice, conducting thorough analysis of financial records, and auditing accounts to ensure accuracy and compliance.

An agent built on the RAISE architecture would be particularly beneficial in this scenario. Leveraging the context and scratchpad modules, the agent can recall relevant financial data from previous records, establish a comprehensive context, and generate insightful analysis. This approach ensures that the agent not only provides accurate financial advice but also adapts its analysis based on historical data, enhancing the quality and relevance of its recommendations.

ReWOO:

ReWOO short for Reasoning WithOut Observation consists of three modules Planner, Worker, and Solver. This framework basically tackles the issue of prompt redundancy in frameworks like RAISE, causing in high costs. The idea behind ReWOO is quite simple. Planner breaks down the task in simpler tasks it is basically a blueprint or the bigger plan. Then those tasks for forwarded to Worker which consists of worker agent. Each agent is given their specific task. The Solver process all the evidence and plans to formulate a solution to the original problem.



https://arxiv.org/pdf/2305.1832

Use case scenario:

Ali needs to find out who created the 1989 comic book that was later adapted into a film featuring Jon Raymond Polito. The ReWOO architecture is ideal for this scenario as it can systematically break down the problem into smaller tasks, gather relevant information at each step, and synthesize the findings into a final, accurate answer.

An agent built on the ReWOO architecture can efficiently address this query. The architecture's step-by-step planning allows the agent to first gather information on Jon Raymond Polito's filmography, identify the 1989 comic book in question, and then track down details about the comic book's creator. By focusing on each sub-task sequentially, the ReWOO architecture ensures that the agent compiles the necessary evidence before arriving at the conclusion that Dave Stevens is the creator of "The Rocketeer," the 1989 comic book adapted into a film featuring Jon Raymond Polito.

Advantages of these frameworks

Systematic Exploration	Less Resource Intensive	Memory Management	Improved Context	Planned output
• LATs	ReWOO	RAISE	• RAISE	• ReWOO

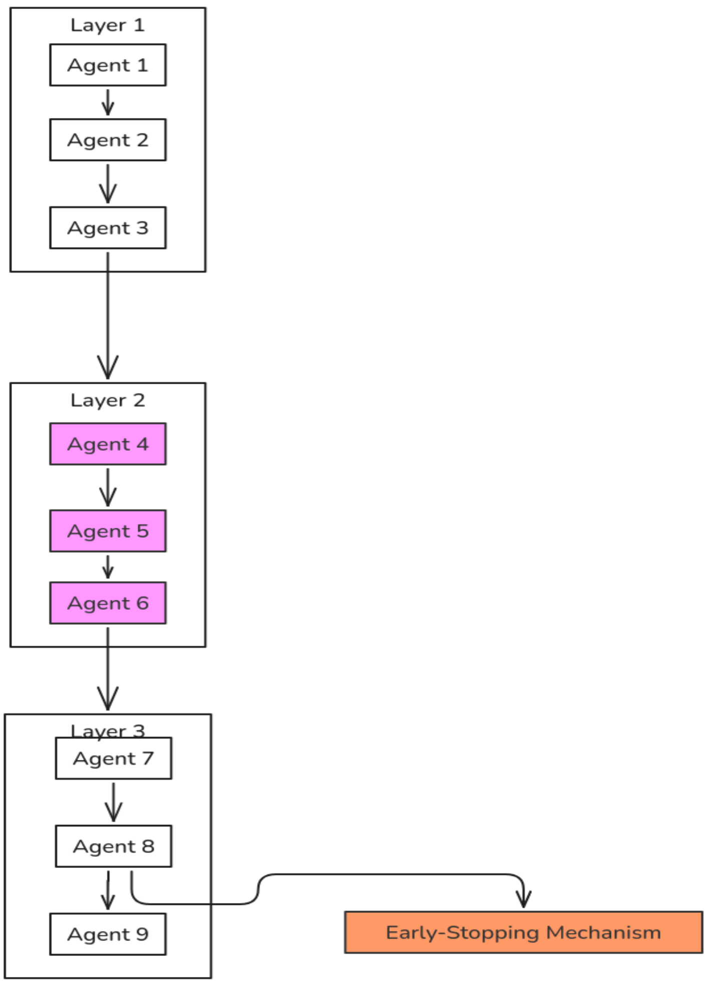
List of Multi Agent Frameworks

The following is a list of various frameworks used to build AI agents:

- DyLAN
- AgentVerse

DyLAN:

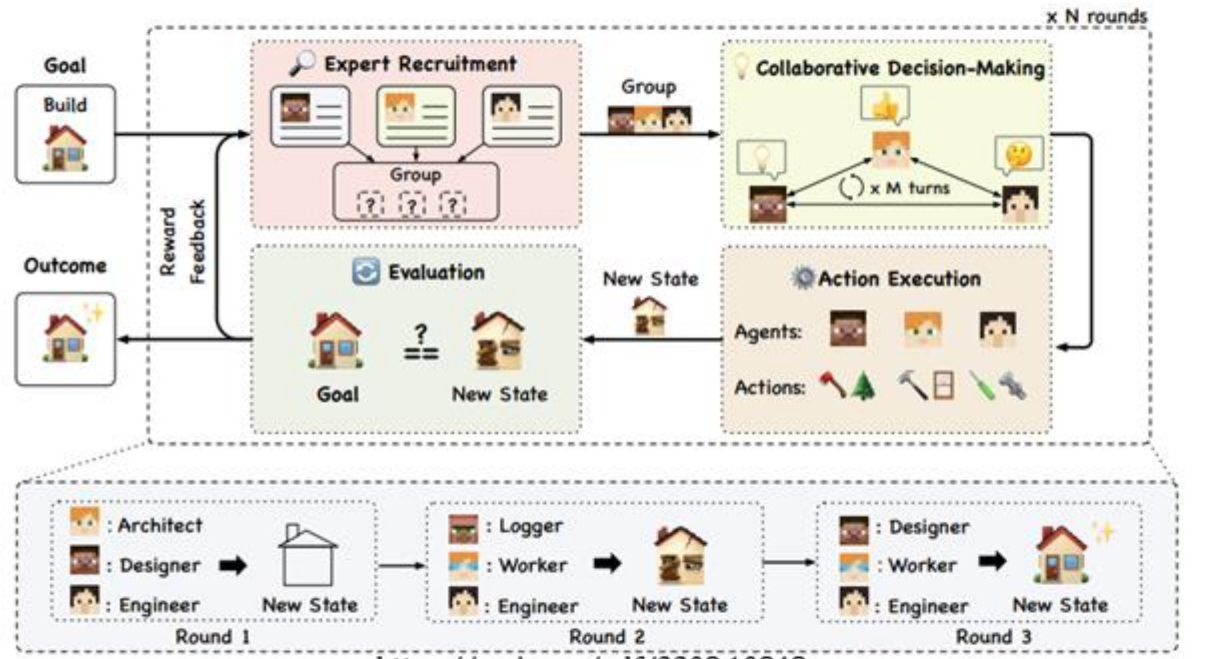
DyLAN, Dynamic LLM-Agent Network is a system designed to help multiple AI agents work together more effectively by allowing them to communicate and collaborate dynamically. Think of it as a series of layers with agents (nodes) connected by lines (edges). Information flows from one layer to the next, with agents sharing and rating ideas. The best ideas are selected to move forward, and the process stops when a consensus is reached.



AgentVerse

Inspired by human group dynamics this framework aims to improve the overall performance of an agentic workflow by utilizing the concept of teamwork. The way human beings work in groups to solve a complex task. Agentverse splits the problem-solving process into four stages.

1. Expert Recruitment: Making the groups according to the problem at hand
2. Collaborative Decision-Making: The agents collaborate with each other to devise a plan or strategy to solve the problem.
3. Action Execution: In this step the agents execute the plan to perform the task at hand.
4. Evaluation: It compares the current output with the desired output and generates feedback based on that.



https://arxiv.org/pdf/2308.10848

All multi-agent frameworks share a common approach to problem-solving: when a problem arises, the task is distributed across the entire pipeline, with different agents handling specific parts of the process until a solution is achieved. So due to this reason I have extracted the unique features of each framework and composed them in a tabular form as show below.

Architecture Name	Unique Feature	Primary Application/Focus Area
MetaGPT	Uses a hierarchical system of prompts for layered and sophisticated interactions with the LLM.	Complex layered AI interactions requiring deep reasoning.
HuggingGPT	Leverages the Hugging Face ecosystem, enabling easy integration of multiple AI models.	Multi-model integration and application in diverse tasks.
Auto-GPT	Automates task execution by decomposing goals into subtasks, managed by AI agents autonomously.	Autonomous task management and execution.
Auto-Agent	Utilizes self-improving capabilities, allowing agents to enhance performance autonomously.	Autonomous improvement and self-optimization in agents.
CAMEL	Focuses on enabling collaborative learning between AI agents, enhancing collective intelligence. Role playing.	Collaborative tasks requiring shared learning and adaptation.

Use cases:

These architectures are good at performing tasks which need a feedback mechanism are not straightforward and need specialized agents or workers. For example, developing a software, project consulting, and account management. For all the tasks mentioned we need to have specialized agents which can work as a team to perform the tasks at hand.

Development Framework:

LangChain

1. It is an open-source framework which allows AI developers to combine LLMs like GPT-4 with external sources of computation and data.
2. It came from a concept known as chain of thoughts, which means the series of ideas that lead to a specific conclusion.
3. Langserver: A package to deploy LangChain chains as REST APIs. Makes it easy to get a production ready API up and running.
4. Langsmith: A developer platform that lets you debug, test, evaluate, and monitor LLM applications.
5. **Components:**

1. **Prompt Templates:** Prompt templates help to translate user input and parameters into instructions for a language model. This can be used to guide a model's response, helping it understand the context and generate relevant and coherent language-based output. There are a few different types of prompt templates:
    i. **Prompt Templates:** These prompt templates are used to format a single string, and generally are used for simpler inputs.
    ii. **Chat prompt templates:** These prompt templates are used to format a list of messages and consist of a collection of templates.
    iii. **Messages placeholder:** The Messages placeholder prompt template allows to insert a list of messages into a specific location within a format

2. **Chat models:** Language models that use a sequence of messages as inputs and return chat messages as outputs are known as chat models. We have some standardized parameters when constructing chat models:

    - *model:* the name of the model
    - *temperature:* the sampling temperature
    - *timeout:* request timeout
    - *max_tokens:* max tokens to generate
    - *stop*: default stop sequences
    - *max_retries*: max number of times to retry requests
    - *api_key:* API key for the model provider
    - *base_url:* endpoint to send requests to

    NOTE: Multimodal chat models can handle different types of inputs like images, audio, and video

3. **LLMs:** Large Language Models (LLMs) are a core component of LangChain. LangChain does not serve its own LLMs, but rather provides a standard interface for interacting with many different LLMs.

4. **Messages:** Language models process a list of messages and return a message, each with a role, content, and optional response metadata. The roles are:
    - **Human message:** Role "user".
    - **AI message:** Role "assistant", includes *response_metadata* (model-specific metadata like log-probs and token usage) and *tool_calls* (decisions to call tools, detailed in ToolCall dictionaries with name, args, and id).
    - **System message:** Role "system", guides model behaviour (not supported by all models).

- **Tool message:** Role "tool", includes *tool_call_id* (ID of the tool call) and artifact (additional useful data from tool execution).
- **Function message (legacy):** Previously used for function calls, now replaced by tool message, includes a name parameter for the function called.

5. **Example Selectors:** Few-shot prompting is a technique where examples are included in the prompt to guide the language model's behaviour. Example Selectors are classes used to choose and format these examples into the prompt for more advanced scenarios.

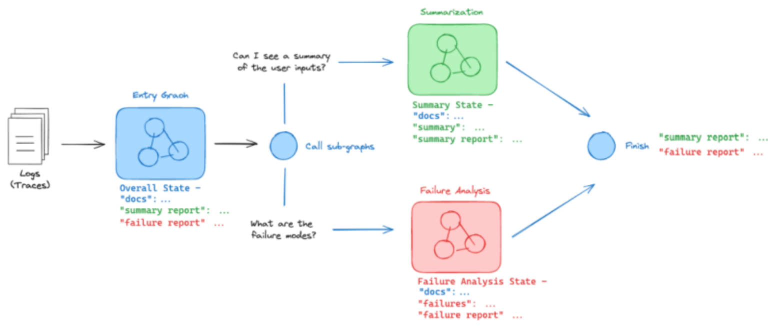
6. **Chat History:** In LLM applications with conversational interfaces, the ability to reference earlier information is crucial for ensuring continuity in the conversation. The *ChatHistory* class in LangChain helps with this by tracking and storing the inputs and outputs of a chain.

7. **Embedding model**: Embedding models convert text into vector representations, which are arrays of numbers capturing the text's semantic meaning. These embeddings capture semantic meaning and relationships between words or phrases. The Embeddings class in LangChain provides a standard interface for various embedding model providers. Some of the embeddings are:
   - AzureOpenAI
   - Ollama
   - OpenAI
8. **Tools/Toolkit:** Tools are utilities designed for interaction with models, enabling models to control code or call external APIs and a toolkit is a collection of tools that are designed to be used together. There are many tools/toolkits but some of them are:
   - **Search:** Like Bing Search, Brave search, DuckDuckgoSearch etc.
   - **Code Interpreter:** Like Bearly code interpreter, E2B data Analysis, Riza code interpreter etc.
   - **Productivity:** Used to automate tasks in productivity tools. Like GitHub Toolkit, Gitlab toolkit, Gmail Toolkit etc.
   - **Web Browsing:** Used to automate tasks in web browsers. Like *MultiOn* toolkit, *PlayWright* Browser, and Requests Toolkit.
   - **Database:** Used to automate tasks in database. Like Cassandra Database toolkit, SQLDatabase Toolkit, and Spark SQL Toolkit.

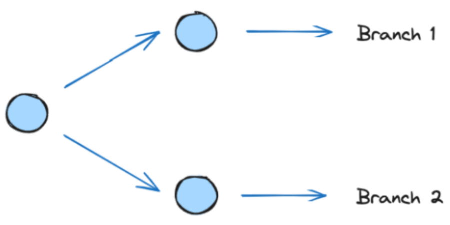
LangGraph:

1. LangGraph is a framework designed to facilitate the development of AI applications using graph-based state machines.
2. The key concepts of LangGraph are:

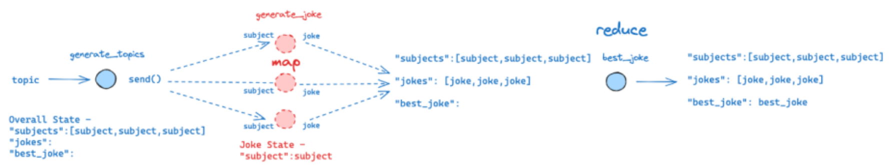
    1. **Nodes:** Nodes are the basic units of work within a LangGraph. They typically represent functions or operations that perform specific tasks
    2. **Edges:** It defines the flow of information and the order of execution between nodes. Conditional edges can also be created to allow for decision-making based on node outputs
    3. **State:** It is a central object that is updated over time by the nodes in the graph. It manages the internal state of the application and can include conversation history, contextual data and variables.
3. LangGraph has several core principles that we believe make it the most suitable framework for building agentic applications:
    1. **Controllability**: LangGraph is known for being a highly controllable agent framework.
        i. **Sub-graphs:** Sub-graphs allow you to create and manage different states in different parts of your graph. This allows to build things like multi-agent teams, where each team can track its own separate state.



        ii. **Parallel node execution:** Parallel execution of nodes is crucial for accelerating graph operations, which boosts the performance of graph-based workflows.



iii. **Map-reduce:** It is used for efficient task decomposition and parallel processing. The approach involves:
   1. Map: Breaking a task into smaller sub-tasks that can be processed independently and in parallel.
   2. Reduce: Aggregating the results from all the sub-tasks to produce the final output.



iv. **Recursion limit:** The recursion limit sets the number of super steps that the graph is allowed to execute before it raises an error.

2. **Human-in-the-Loop**: These interactions are essential for managing and guiding agentic systems. Breakpoints pause execution to seek human approval on critical actions, built on a checkpoint system that saves the graph's state after each node. They allow dynamic interruptions and state updates, supporting human-in-the-loop interactions. This includes reviewing or modifying actions, managing tool calls (like executing SQL or generating summaries), and updating the agent’s state to enhance control and integration within the system.
3. **Streaming First**: LangGraph supports robust streaming for real-time updates in agentic applications, including event streaming (e.g., tool call executions) and token streaming (e.g., LLM outputs). This functionality keeps users informed about ongoing processes and enhances their experience. LangGraph supports two main streaming modes:
   i. **Values:** Streams the entire state of the graph after each node is processed, showing the complete status of the graph.
   ii. **Updates:** Streams only the changes or updates to the graph state after each node is processed, showing what has changed since the last update.

4. **Tool calling:** Tool calling is the process of choosing and specifying tools and their inputs based on context. The LLM decides which tools to use and invokes them accordingly.
5. **Memory:** Memory management is crucial for agentic applications, as users often expect the system to recall previous interactions. LangGraph provides comprehensive control over memory management:
   i. **User-Defined State:** Allows to define the exact schema for the memory you want to retain.
   ii. **Checkpointers:** Enable to store checkpoints of previous interactions and resume from those points in future interactions.

CrewAI:

1. CrewAI is an open-source framework for multi-agent collaboration built on Langchain. It enables various artificial intelligence agents to work together as a team, efficiently accomplishing complex tasks.
2. Core-concepts of CrewAI:

    1. **Agents:** An agent is an autonomous unit that Perform tasks, make decisions and communicate with other agents. In defining the agent, we need to provide them a role, goal and backstory

    2. **Tasks:** These are specific assignments completed by agents. They provide all necessary details for execution. This expects at least 3 attributes: *description*, *expected_output* and *agent*.
        i. **description***:* The description of why we expect the task to be or what we expect the agent to do.
        ii. **expected_output***:* What do we expect in the output.
        iii. **agent***:* Assign an agent to this task.

    3. **Tool:** It is a skill or function that agents can utilize to perform various actions. Some tools are:
        i. **BrowserbaseLoadTool**: For extracting data from web browsers.
        ii. **CodeInterpreterTool:** A tool for interpreting python code.
        iii. **DirectorySearchTool:** A RAG tool for searching within directories, useful for navigating through file systems.
        iv. **FileReadTool**: Enables reading and extracting data from files, supporting various file formats.
        v. **LlamaIndexTool:** Enables the use of LlamaIndex tools.
        vi. **YoutubeChannelSearchTool:** A RAG tool for searching within YouTube channels, useful for video content analysis.

    4. **Processes:** It ensures tasks are distributed and executed efficiently, in alignment with a predefined strategy. The processes are:
        a. **Sequential:** It progresses through tasks in a systematic manner. Task execution follows the predefined order in the task list, with the output of one task serving as context for the next.
        b. **Hierarchical Process:** Emulates a corporate hierarchy, CrewAI allows specifying a custom manager agent or automatically creates one. This agent oversees task execution, including planning, delegation, and validation.
        c. **Consensual Process:** Future feature for collaborative, democratic decision-making among agents.

5. **Crew:** It represents a collaborative group of agents working together to achieve a set of tasks. Its basic 3 attributes are:
    a. **Tasks:** A list of tasks assigned to the crew.
    b. **Agent:** A list of agents that are part of the crew.
    c. **Verbose:** The verbosity level for logging during execution.
6. **Collaboration:** It helps agents in:
    a. **Information Sharing:** Keeps all agents informed for better collaboration.
    b. **Task Assistance:** Allows agents to get help from peers with specific expertise.
    c. **Resource Allocation:** Distributes resources efficiently among agents for optimal task execution.
7. **Pipeline:** A pipeline in CrewAI organizes and runs multiple crews in sequence or parallel, where each stage's output can be used as input for the next stage
8. **Memory:** It offers sophisticated memory system which includes:
    a. **Short-Term Memory:** Temporarily holds recent interactions for use during current tasks.
    b. **Long-Term Memory:** Stores insights from past tasks, helping agents learn and improve over time.
    c. **Entity Memory:** Organizes information about key entities (like people or places) for better understanding.
    d. **Contextual Memory:** Combines all memory types to maintain context, ensuring coherent and relevant responses across tasks.
9. **Planning:** It allows to add planning capability in the crew. When enabled, before each Crew iteration, all Crew information is sent to an AgentPlanner that will plan the tasks step by step, and this plan will be added to each task description.
10. **Testing:** It allows us to test the crew and evaluate its performance using the built-in testing capabilities.

Microsoft Semantic Kernel:

1. Semantic Kernel is an SDK that integrates Large Language Models (LLMs) like OpenAI, Azure OpenAI, and Hugging Face with conventional programming languages like C#, Python, and Java.
2. **Agent:** It is made up of three core building blocks:
    a. **Plugins:** It allows to give the agent skills via code
    b. **Planner:** It refers to the process where an LLM creates a structure to complete a task. It involves breaking down the task into manageable steps and determining the best way to execute them iteratively
    c. **Persona:** These are the instructions that we provide the agent so they can more effectively perform the role we want them to play. At its simplest, the persona could instruct the AI to be polite.

3. **Kernel:** The kernel is the central component of Semantic Kernel. It manages all the services and plugins needed to run your AI application.
4. **AI services:** Some of them are:
    a. Chat completion
    b. Text generation
    c. Embedding generation
5. **Chat history:** It is essential for maintaining context and continuity in a conversation.
6. Semantic Kernel provides connectors to vector databases, allowing you to easily store and retrieve information. Some of them are:
    a. Vector Database in Azure Cosmos DB for NoSQL
    b. Azure AI Search
7. **AI service connectors:** These are tools that link the system to external AI services. Some of them are:
    a. OpenAI
    b. Azure OpenAI

AutoGen

1. AutoGen is an open-source programming framework for building AI agents and facilitating cooperation among multiple agents to solve tasks.
2. It supports 2 languages python and .NET.
3. The code execution is within a Docker container so that LLM written code does not harm the computer systems.
4. In AutoGen a conversation between agents can be terminated by 2 ways:
    a. **Specify parameters in *initiate_chat*:** When initiating a chat, you can define parameters that determine when the conversation should end.
    b. **Configure an agent to trigger termination:** When defining individual agents, you can specify parameters that allow agents to terminate a conversation based on conditions.
5. AutoGen supports human feedback in agents. The three modes are:
    a. **Never: H**uman input is never requested.
    b. **Terminate (default):** Human input is only requested when a termination condition is met
    c. **Always**: Human input is always requested, and the human can choose to skip trigger an autoreply, provide feedback, or terminate the conversation.
6. **Tools:** These are pre-defined functions that agents can use, such as searching the web, performing calculations, reading files, or calling remote APIs.
7. AutoGen supports 4 types of conversation patterns. They are:
    a. **Two-agent chat:** the simplest form of conversation pattern where two agents chat with each other.
    b. **Sequential chat:** a sequence of chats between two agents, chained together by a carryover mechanism, which brings the summary of the previous chat to the context of the next chat.

c. **Group chat:** It supports multiple agents in a chat, with strategies for selecting the next speaker can be chosen from round robin, random, manual or auto. Custom functions can also be used for the selection of the next speaker.
d. **Nested Chat:** package a workflow into a single agent for reuse in a larger workflow.

Potential Use Cases:

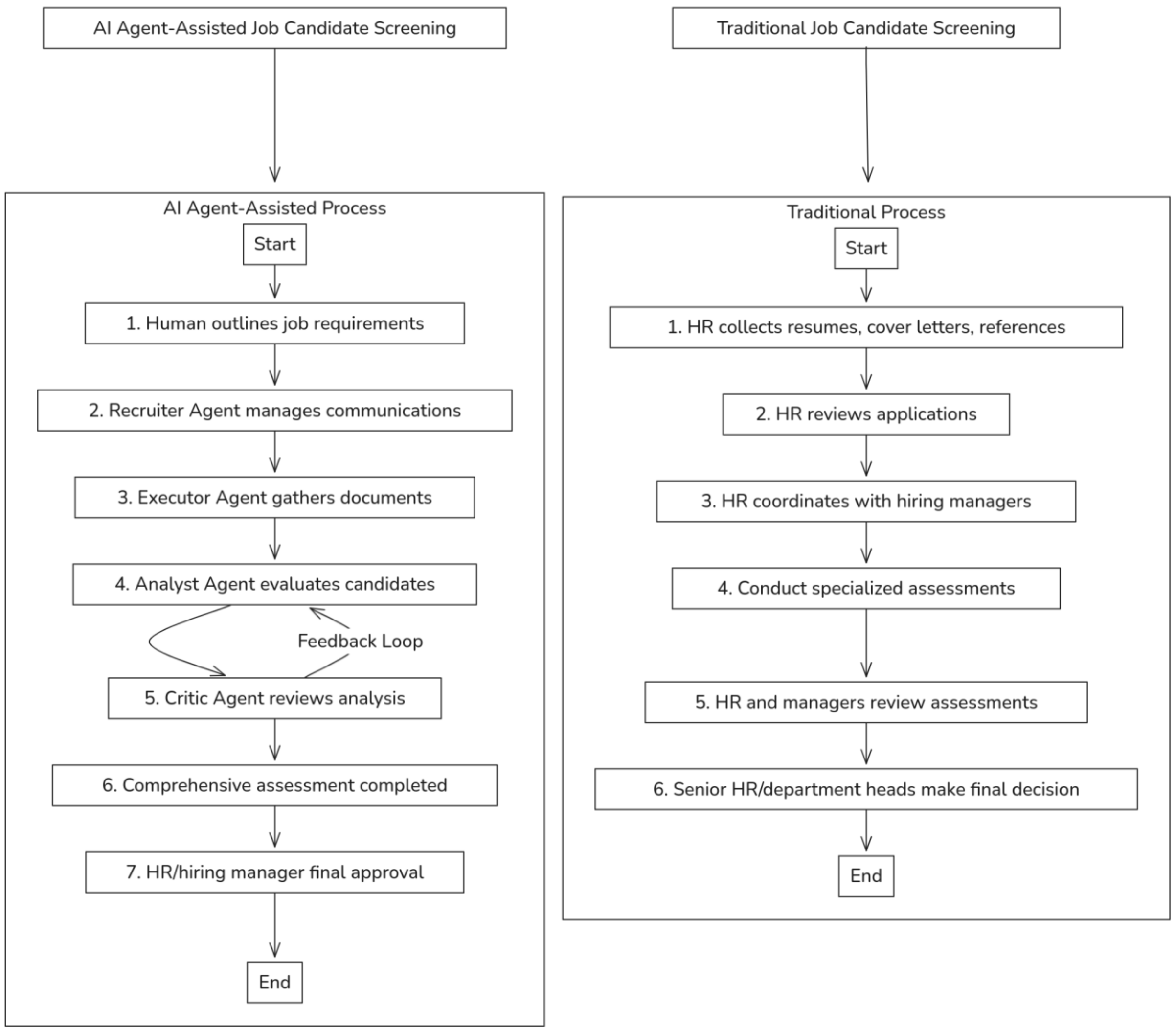
Job Screening

Human resource departments often engage in job candidate screening to evaluate the suitability of applicants for open positions. This process involves collecting, analyzing, and reviewing various pieces of information about candidates, including their resumes, cover letters, references, and other relevant documents. Given the diversity of roles and the depth of evaluation required, this process can be time-consuming and demands a collaborative effort, with recruiters working closely with hiring managers, team leads, and sometimes external consultants to conduct specialized assessments. These assessments are then reviewed by senior HR personnel or department heads to make final hiring decisions.

Potential agent-based solution:

An agentic system—comprising multiple agents, each with a specialized, task-based role—could be designed to manage the job candidate screening process across various positions. A human user could initiate the process by outlining the key qualifications, experience, and competencies required for the role in natural language, specifying any particular standards or conditions.

One agent could act as the recruiter, managing communications between candidates and the company, while another agent could serve as an executor, gathering and organizing candidate documents such as resumes and references. A third agent could function as an analyst, evaluating the candidate's experience, skills, and cultural fit by comparing their credentials against the job requirements, and calculating relevant scores or metrics. A critic agent could then review the analysis, identifying any discrepancies or areas needing further inquiry, and provide feedback for refinement. This iterative process of analysis, refinement, and review would continue until a comprehensive candidate assessment is completed, ready for final approval by HR or the hiring manager.


AI Agent-Assisted Job Candidate Screening
AI Agent-Assisted Process
Start
1. Human outlines job requirements
2. Recruiter Agent manages communications
3. Executor Agent gathers documents
4. Analyst Agent evaluates candidates
Feedback Loop
5. Critic Agent reviews analysis
6. Comprehensive assessment completed
7. HR/hiring manager final approval
End
Traditional Job Candidate Screening
Traditional Process
Start
1. HR collects resumes, cover letters, references
2. HR reviews applications
3. HR coordinates with hiring managers
4. Conduct specialized assessments
5. HR and managers review assessments
6. Senior HR/department heads make final decision
End

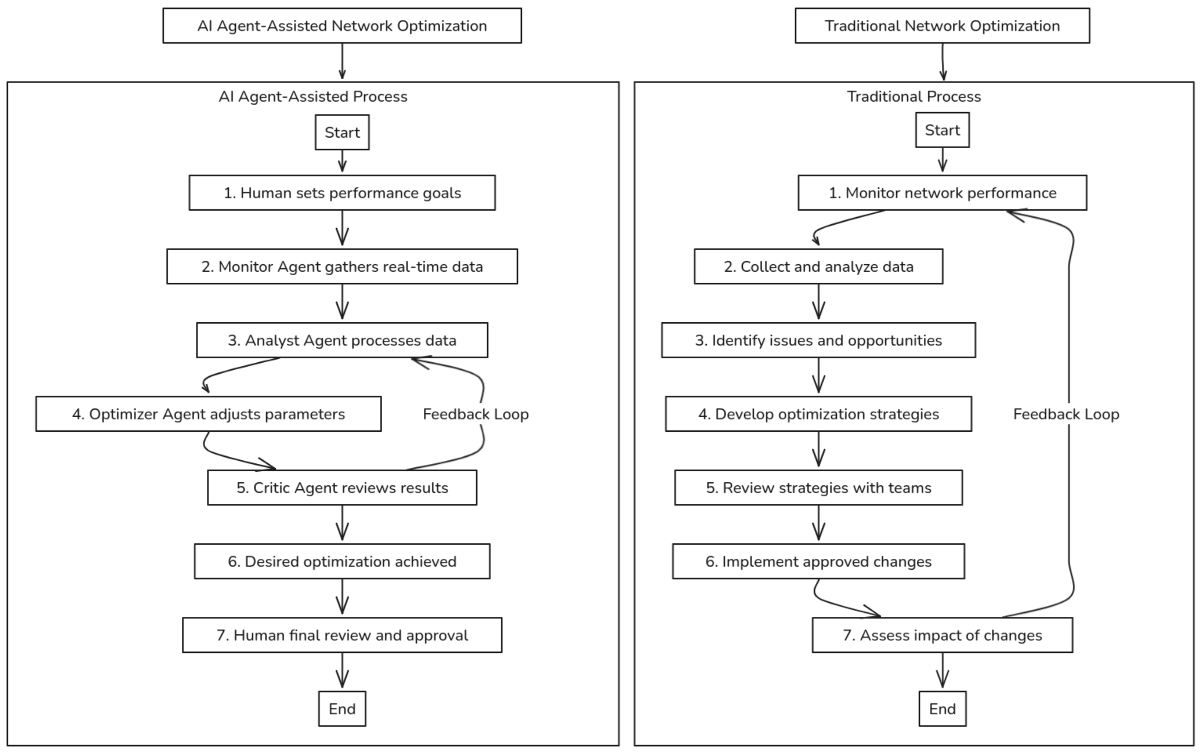
Network Optimization:

Telecommunications companies often undertake the complex process of network optimization to ensure their services meet the highest standards of performance and reliability. This involves the continuous monitoring, analysis, and adjustment of network parameters, such as signal strength, bandwidth allocation, and coverage areas. Given the vast scale of modern telecom networks and the variety of factors influencing performance—such as user demand, geographic conditions, and technological constraints—this process can be highly time-consuming and requires collaboration among network engineers, data analysts, and operations teams. These teams work together to conduct specialized analyses, which are then reviewed and implemented by senior engineers or network managers.

Potential agent-based solution:

An agentic system—comprising multiple agents, each with a specialized, task-based role—could be designed to manage the network optimization process across different regions and technologies. A human user could initiate the process by providing a high-level overview of network performance goals, specific areas of concern, and any operational constraints using natural language.

One agent could act as the network monitor, continuously gathering real-time data on network performance metrics such as latency, throughput, and error rates. Another agent, serving as the data analyst, could process this data to identify trends, bottlenecks, and areas for improvement. A third agent could function as the optimizer, using algorithms to adjust network parameters in real time to enhance performance. A critic agent could then review the results of these adjustments, comparing them against the initial performance goals, and provide feedback for further fine-tuning. This iterative process of monitoring, analysis, adjustment, and review would continue until the network reaches the desired level of optimization, ensuring reliable and efficient service delivery.


AI Agent-Assisted Network Optimization
AI Agent-Assisted Process
Start
1. Human sets performance goals
2. Monitor Agent gathers real-time data
3. Analyst Agent processes data
4. Optimizer Agent adjusts parameters
Feedback Loop
5. Critic Agent reviews results
6. Desired optimization achieved
7. Human final review and approval
End
Traditional Network Optimization
Traditional Process
Start
1. Monitor network performance
2. Collect and analyze data
3. Identify issues and opportunities
4. Develop optimization strategies
Feedback Loop
5. Review strategies with teams
6. Implement approved changes
7. Assess impact of changes
End

Future Prospects

Enhanced Efficiency: In the future, Agentic AI will continue to revolutionize operations by automating increasingly complex and repetitive tasks. This will lead to even faster processing times, minimizing the need for human intervention and streamlining workflows across various industries.

Improved Decision-Making: The growing capabilities of AI agents to analyse vast amounts of data and recognize evolving patterns suggest that future decision-making will become more precise and strategic. As AI systems advance, businesses will be able to rely on even more accurate, data-driven insights to navigate challenges and opportunities.

Increased Productivity: As Agentic AI develops, its ability to offload routine tasks will further enhance human productivity. Future AI systems will allow employees to focus entirely on high-level strategy and innovation, potentially leading to groundbreaking advancements in multiple sectors.

Scalability: The future prospects of Agentic AI include unprecedented scalability, enabling organizations to manage vast volumes of data and complex tasks with ease. This ensures that as businesses expand, they can do so without facing increased operational burdens.

Appendix

Architecture

Architecture refers to the comprehensive blueprint that outlines how a system should be constructed. It encompasses the interaction of various components within the system, detailing how they work together to achieve the desired functionality.

Framework

Framework is a more focused, structured approach applied to specific aspects of the architecture. It provides guidelines or methods to accomplish objectives within the broader architectural structure.

Hallucinations

Hallucinations in LLMs refer to the generation of content that is irrelevant, made-up, or inconsistent with the input data.