THE CYPRUS INSTITUTE

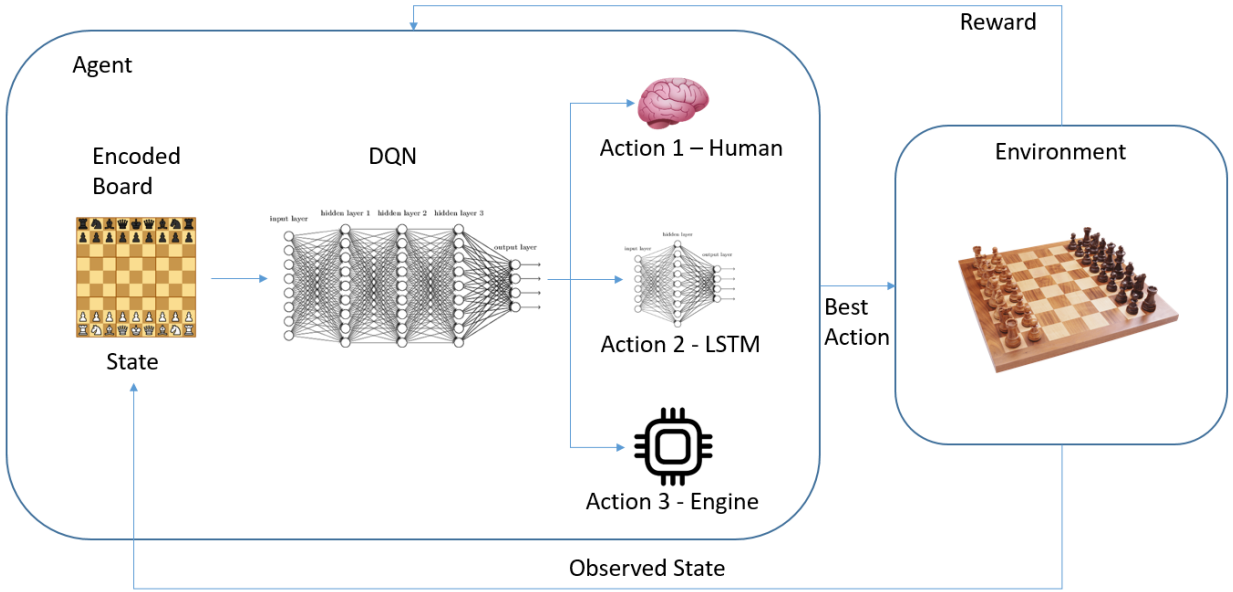
RESEARCH•TECHNOLOGY•INNOVATION

From AI Agents to Agentic AI: Definitions, Architectures, and Popular Frameworks

Dr. Valeria Filippou

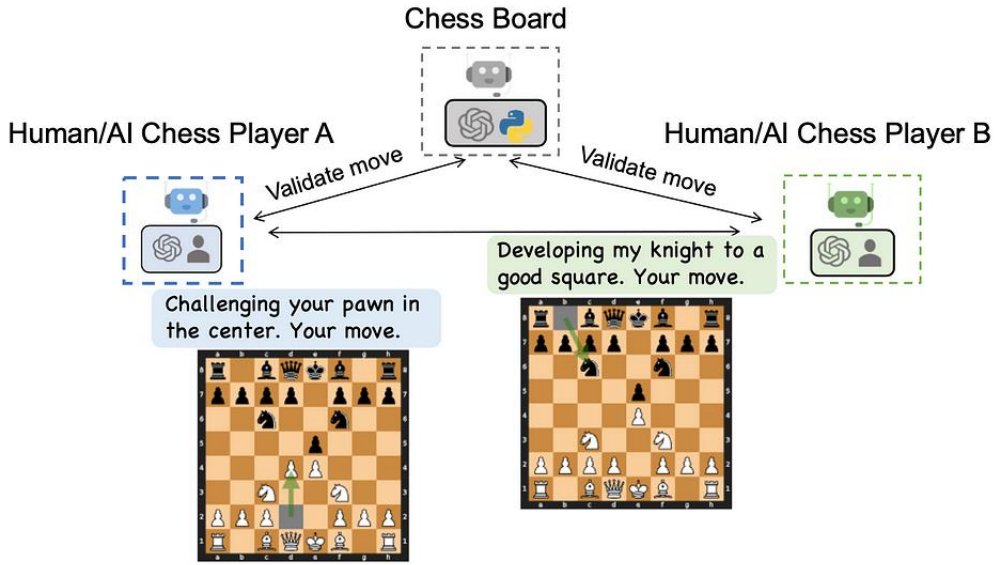
Traditional vs. LLM-based Multi-Agent Systems: A Chess Case Study

Traditional MAS



- Uses **reinforcement learning** to simulate optimal moves.
- Article

LLM-based MAS



- Agents powered by LLMs **communicate and reason in natural language**.
- Article

Key Differences Between Traditional & LLM-Based MAS

Aspect	Traditional MAS	LLM-based MAS
Coordination	**Predefined** protocols (e.g., auctions, blackboards).	Dynamic negotiation and role assignment.
Environment Interaction	Symbolic representations or domain-specific simulators.	Natural language interfaces and API integration.
Adaptability	Requires **retaining** or **reinforcement learning** updates.	Real-time adaptation via **prompting**.
Tool Use / API Calls	Manually integrated external tools and services.	**Autonomous** tool invocation.
Transparency	**High** Interpretable logic.	**Low** Black-box LLM decisions.

This is where we will focus in the training.

Introduction to AI Agent Frameworks: CrewAI vs. AutoGen

Both frameworks enable LLM-based multi-agent systems, but with different approaches.

CrewAI

```
from crewai import Agent

agent = Agent(
    role="Researcher",
    goal="Identify key trends in data",
    backstory="An expert analyst in AI and markets.",
    verbose=True
)
```

Plain Text

- **Team-like collaboration** - Agents work together like human colleagues
- **Role-based design** - Each agent has a specific job and personality
- **Autonomous execution** - Minimal supervision needed

AutoGen

```
from autogen import AssistantAgent

agent = AssistantAgent(
    name="researcher",
    llm_config={
      "model": "gpt-4",
      "temperature": 0.5
    }
)
```

Plain Text

- **Conversation control** - You manage how agents interact
- **Flexible agent types** - Assistant agents, user proxies, custom agents
- **Structured workflows** - Define precise interaction patterns

Choose CrewAI for team-like collaboration tasks.

Choose AutoGen for controlled, structured multi-agent conversations.

Agentic AI & AI Agents: Core Concepts, Types, and Integration

Module 1

AI Agents vs. Agentic AI

What is an AI Agent?

A self-contained, autonomous software entity that performs a **specific task** using AI techniques — often powered by an LLM.

- Acts as a **single intelligent unit** with a narrow scope
- Operates via an **observe → plan → act** loop based on environmental input
- May use tools/APIs to take actions (e.g. summarisation, scheduling)
- **Examples**: "Summarise this document", "Generate a response", "Book a meeting"

What is Agentic AI?

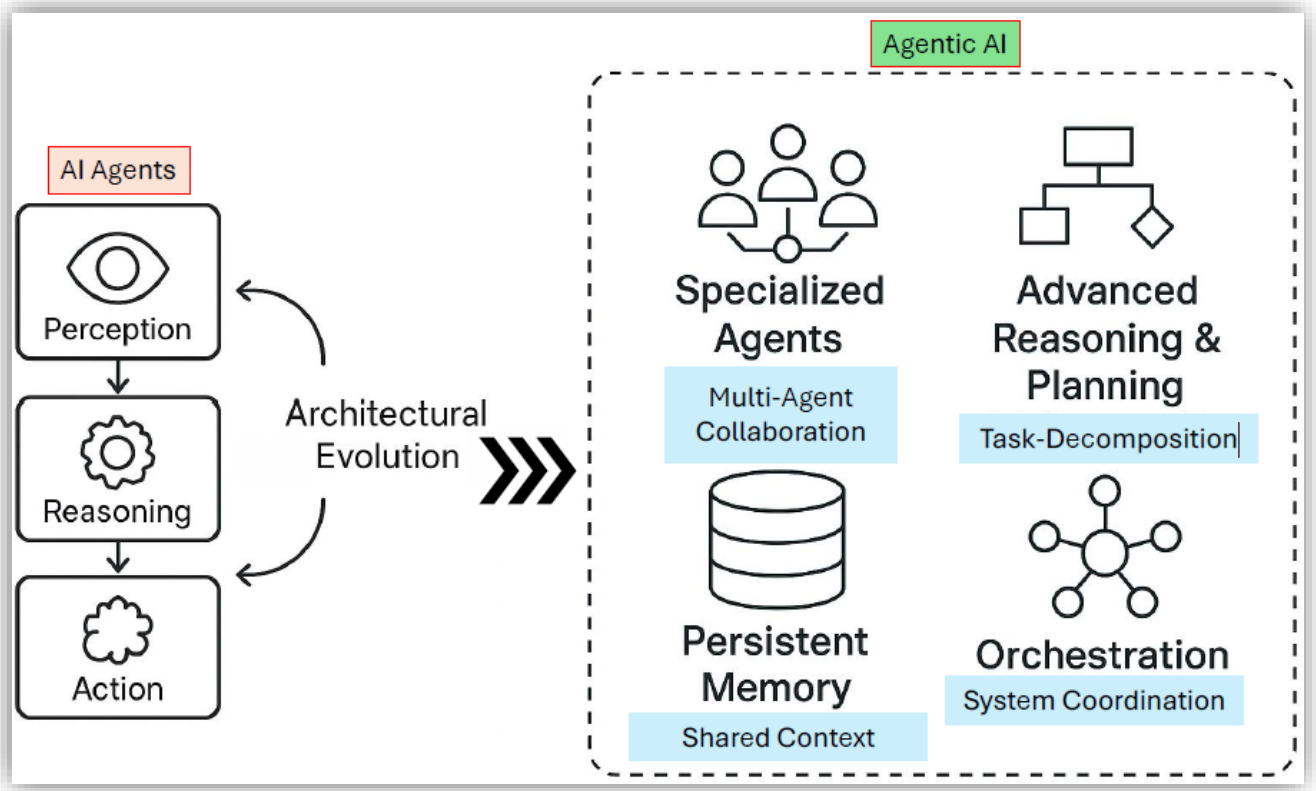
A **system-level approach** that combines multiple LLM-based agents to **collaborate** and solve **complex**, **multi-step problems**.

- Involves **multi-agent systems** (MAS) with shared goals and communication
- Supports **dynamic planning**, memory and role delegation
- Emphasises **coordination** and **autonomous decision-making**
- Goes beyond single-task automation to enable **distributed goal achievement**

AI agents are the workers; Agentic AI is the management system that coordinates them.

Reference Paper

From AI Agents to Agentic AI: A Functional Evolution



This diagram illustrates how AI systems evolve from basic agent architectures — with core modules like Perception, Reasoning, and Action — to agentic systems that support memory, planning, collaboration, and emergent behaviours.

Source: *Sapkota et al., 2025.*

Core Characteristics of Agentic AI

What makes an AI system “agentic”?

- **Autonomy**: Acts independently to pursue goals without constant human input.
- **Reactivity**: Responds to environmental inputs, events, or feedback in real-time.
- **Proactivity**: Initiates actions based on internal goals, not just external triggers.
- **Adaptability**: Adjusts behaviour based on context, new information or failure cases.
- **Learning & Iteration**: Refines performance through feedback loops, memory, or reflection.

In LLM-based systems, these qualities often appear through structured prompting, memory integration, tool usage, and iterative reasoning.

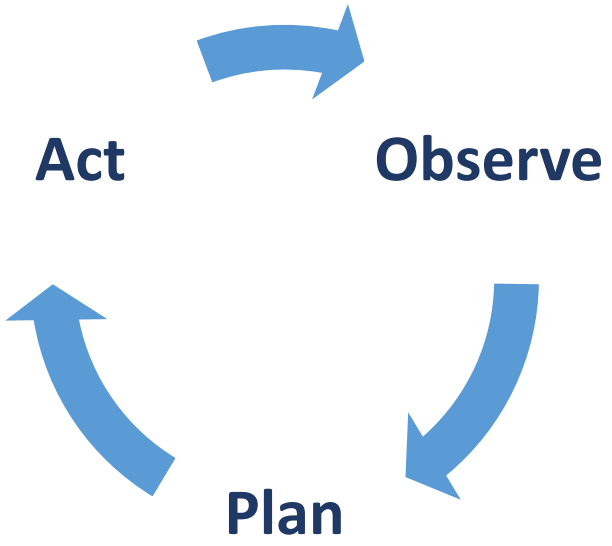
Agentic AI vs. Traditional AI

Agentic AI provides a more autonomous, adaptive, and interactive approach to traditional AI. Understanding these differences helps us select the right solution for each application.

	Traditional AI Chatbot	Agentic AI Chatbot
Autonomy	Responds only to scripted questions	Takes initiative – follows up or escalates unresolved issues
Reactivity	Replies only when prompted	Proactively offers help based on context
Learning	Requires manual updates to improve	Learns continuously from interactions to refine understanding
Interaction	Limited to basic Q&A	Engages in multi-turn conversations, gather info, and perform actions
Decision-making	Follows predefined flows	Chooses actions dynamically, considering tone, context, and goals

Key Components of AI Agents

- **Perception**: Collects data from the environment – such as user queries, sensor readings, or external signals.
- **Decision-Making**: Uses AI models (e.g. LLMs, reinforcement learning) to interpret data and plan the next actions.
- **Action Execution**: Performs tasks by interacting with external systems & tools – APIs, databases, CRMs, etc.
- **Memory**: Stores relevant context and history (both short-term & long-term) to support informed future decisions.
- **Feedback Loop**: Monitors outcomes and learns from them to improve future behaviour and performance.



Benefits of AI Agents & Agentic AI

Boost Productivity

- Automate tasks and entire workflows – freeing teams to focus on innovation and higher-value work.

Reduce Manual Errors

- Minimise human error and inefficiencies with intelligent, self-correcting, systems.

24/7 Availability

- Ensure continuous task execution and support, even outside business hours.

Tailored Solutions

- Build agents or agentic systems customised to your goals, tools, and processes.

Core AI Agent Benefits

Seamless Collaboration

- Enable different systems and teams to share data and coordinate tasks automatically.

Scalable by Design

- Add new capabilities or agents without needing to re-architect the system.

Complex Problem Solving

- Handle complex workflows requiring multiple specialised skills working together.

Additional Multi-Agent Advantages

Examples of AI Agents

Customer Service Agent

- **Function:**
  - **Observe**: Reads user queries (e.g., "Track my order.")
  - **Plan**: Uses an LLM to determine next steps (e.g., "Query the CRM for order status")
  - **Act**: Calls CRM APIs and replies with retrieved information
- **Agentic Traits:**
  - Autonomy (no human input needed)
  - Goal-oriented (resolves customer issues efficiently)
- **Example:** E-commerce chatbot

Market Research Agent

- **Function:**
  - **Observe**: Task task input (e.g. "Analyse 2024 AI trends.“)
  - **Plan**: Uses an LLM to break down steps (e.g., "Search the web, summarise findings").
  - **Act**: Uses tools like `SerperDevTool` (web search) and `FileTools` (to store findings)
- **Agentic Traits:**
  - Proactive (plans & decomposes steps)
  - Data-driven (makes evidence-based conclusions)
- **Example:** Business Intelligence agent

Code Generation Agent

- **Function:**
  - **Observe**: Reads code context (e.g., a function name or comment)
  - **Plan**: Predicts developer intent (e.g., "Generate a sorting algorithm")
  - **Act**: Writes code snippets or entire functions
- **Agentic Traits:**
  - Reactive (responds in real time code context)
  - Self-improving (learns from user feedback)
- **Example:** Programming assistant

While AI agents can be categorised by function, most advanced systems are hybrid agents — combining multiple capabilities to support more complex, adaptive behaviour.

AI Agents in Action: Integrating with Intelligent Systems

Connecting to Other Tools & Data Sources

AI agents integrate with external systems with **CRMs**, **APIs**, **cloud services**, and **IoT devices**—to **access real-time data** and **take automated actions**.

Working with Other AI Systems — Hybrid AI

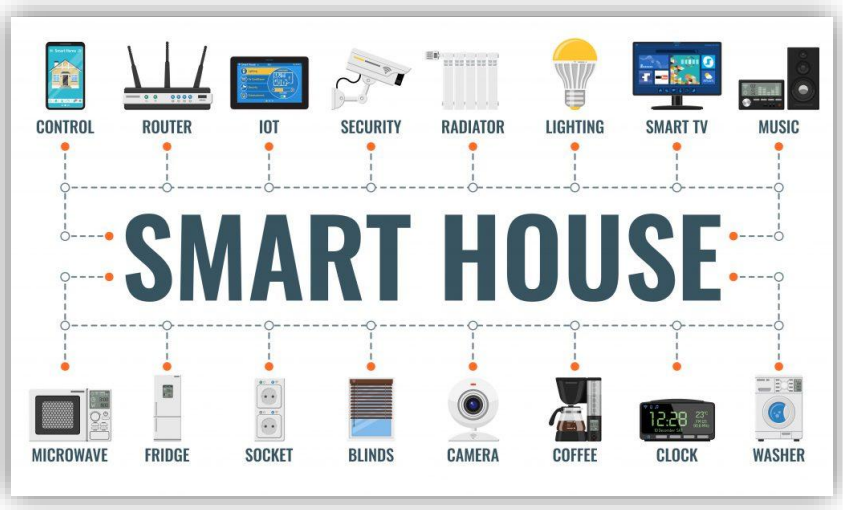
Agents combine strengths of multiple AI types: **Machine Learning** to predict outcomes, **Knowledge Graphs** to understand relationships, **Rule-Based Systems** to enforce logic.

Running on Devices Close to Users — Edge Computing

Agents can run locally on phones, drones, or sensors, enabling faster responses and stronger privacy — like on-device facial recognition.

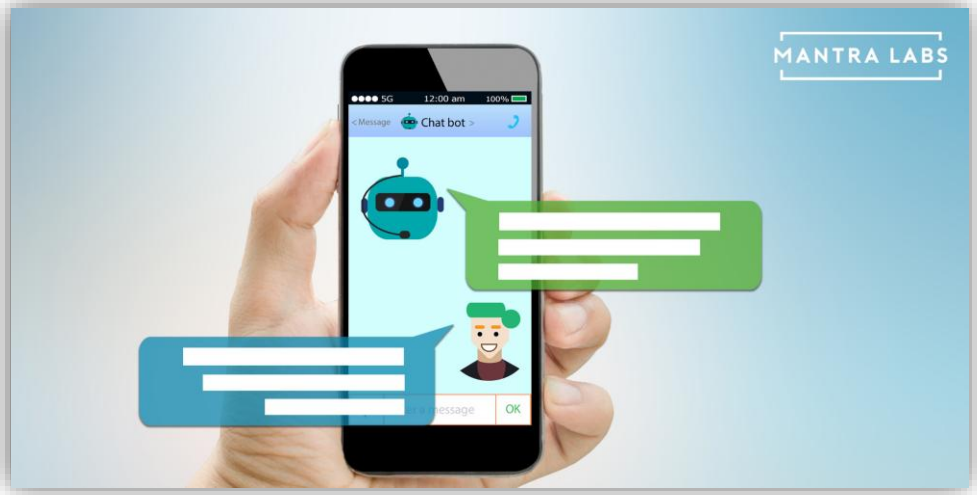
Integration unlocks scalability, adaptability, and precision in AI agent performance.

Quiz time: MAS (without LLM) or LLM-based MAS?



MAS

LLM-based MAS



Types of AI Agents – Overview

- **Simple reflex agents**:

React to immediate environment

- **Model-based reflex agents**:

Use internal models to decide

- **Goal-based agents**:

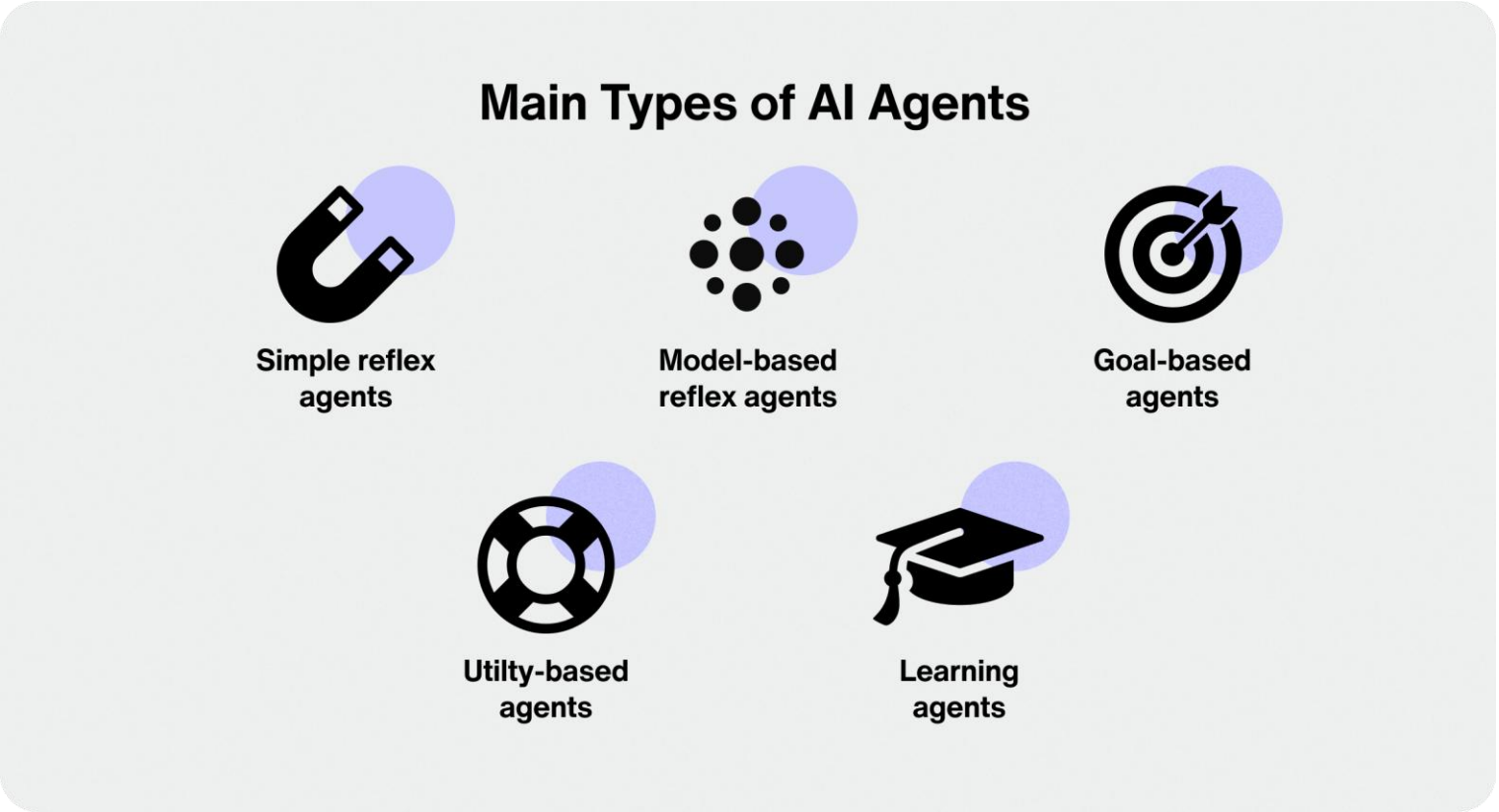
Pursue specific objectives always

- **Utility-based agents**:

Make decisions based on value

- **Learning agents**:

Improve over time always



These are **classical AI agent types**, and they form the **theoretical foundation** for modern **LLM-based agents**. Today, many LLM-based agents **combine multiple types** — e.g., using utility scoring, goal-setting, and learning in the same system.

Simple Reflex Agents

Definition:

Simple reflex agents are the **most basic** type of AI agents. They act based on **current input only**, without memory of past actions or complex decision-making.

Key Characteristics:

- **Immediate response** to specific stimuli
- **No memory** or internal representation of the world
- **Predefined** condition-action rules

Strengths & Limitations:

Strengths	Limitations
Fast, predictable, low overhead	Cannot handle ambiguity or complexity
Suitable for stable environments	No adaptation or planning

Other examples: Thermostat, automatic doors, basic spam filters

Auto-Classification Bot: Categorises customer inquiries as 'Technical Support,' 'Billing,' or 'General' based on detecting specific keywords, without considering conversation history or context.

Model-Based Reflex Agents

Definition:

Model-based reflex agents maintain an **internal model** of the world. They use it to reason about the **current state** and **unseen elements** of the environment.

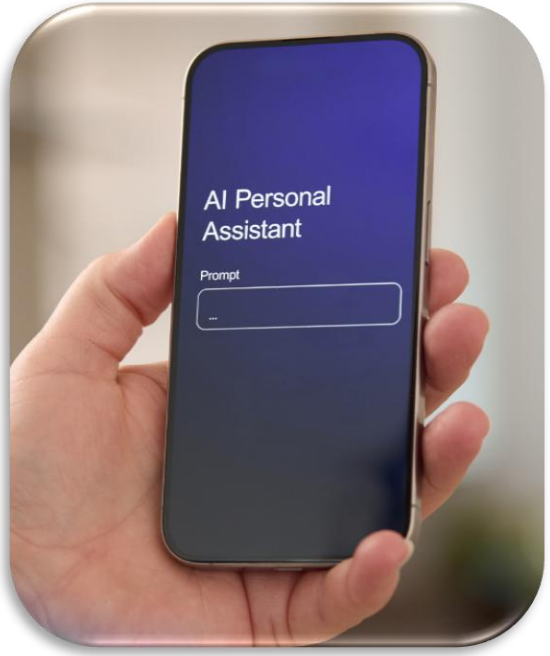
Key Characteristics:

- Maintains an **internal state** (memory of past interactions or context)
- Operates in **environments where not all information is immediately available**
- Combines **past knowledge** and **real-time input**
- Makes smarter decisions than simple reflex agents

Strengths & Adaptability:

These agents are **more adaptable to changing conditions** than simple reflex agents.

Capabilities	Compared to simple reflex agents
Adapts to dynamic environments	Handles more complexity
Uses memory/history in planning	More context-aware decisions



AI Personal Assistant: Reschedules meetings by referencing calendar history and user preferences to suggest optimal meeting times.

Other examples: Autonomous vehicles, modern irrigation systems, home automation systems

Goal-Based Agents

Definition:

Goal-based agents focus on **achieving specific objectives** rather than solely to the current environment. They use **planning and prediction** to find effective paths to their goals.

Key characteristics:

- Operate based on **defined goals or objectives**
- **Plan sequences of actions** to reach those goals
- **Evaluate possible outcomes** to select the best action
- Can adjust plans dynamically as new information arrives

Strengths & Use Cases:

Strengths	Notes
Excels in **complex problem-solving**	Ideal for multi-step tasks
Strategic, forward-looking	Requires more **computation**
Flexible and adaptable	Balances multiple constraints

Other examples: Robotic vacuum cleaner, project management software, video game AI

AI Travel Planner: Creates personalised travel itineraries by planning multi-step processes: researching destinations → evaluating options against budget/preferences → booking optimal choices.

Utility-Based Agents

Definition:

Utility-based agents **enhance decision-making** by choosing actions that maximise expected 'utility' – a quantified measure of benefit, satisfaction, or preference.

Key Characteristics:

- Use a **utility function** to evaluate and compare possible outcomes
- **Balance competing objectives** (e.g., speed vs. cost, accuracy vs. effort)
- Select the **most beneficial action**, not just one that achieves a goal
- Handles **uncertainty** and **trade-offs** effectively

Strengths & Use Cases:

Strengths	Use Scenarios
Optimal decisions in multi-criteria tasks	Financial strategy, resource allocation
Adaptive to context and constraints	Complex or ambiguous environments
Can model risk vs. reward trade-offs	AI-driven negotiation/recommendation

Other examples: Financial trading, dynamic pricing systems, smart grid controllers

AI Job Application Assistant: Evaluates job postings against your CV, career goals, and preferences (salary, location, company culture) → Calculates utility scores for each opportunity → Recommends positions that maximise your overall satisfaction.

Learning Agents

Definition:

Learning agents are capable of **improving their behaviour over time** by learning from past actions, feedback, and environmental changes. They represent the **most dynamic and adaptive** category of agents.

Key Characteristics:

- **Learn from experience** via feedback loops
- **Continuously improve** performance and efficiency
- Adapt to **new tasks**, data, and environments over time
- Often use techniques from **ML**, **RL**, or **feedback-based fine-tuning**

Strengths & Use Cases:

Strengths	Ideal For
Improves with continued use	Long-term systems, personalisation
Handles non-stationary environments	Changing inputs, evolving tasks
Can generalise beyond training	Unexpected user behaviour or data shifts

Other examples: Fraud detection, content recommendation, speech recognition software

AI Writing Coach: Learns your writing style, vocabulary, and through feedback over time to give more relevant and stylistically aligned suggestions→ Can tune tone, structure, and suggestions based on personal history.

From Classical AI Agents to LLM-Based Agents

Classical Agent Type	LLM-based Interpretation	Example
Reflex Agent	Responds to **keywords/patterns** in prompts	"What's the weather?" → Static answer
Model-Based Agent	Maintains conversation **context/memory**	Assistant that remembers your name
Goal-Based Agent	Uses planning over tasks/goals	AI planner that breaks down subtasks
Utility-Based Agent	**Evaluate and compares** multiple options	Selects best option from RAG results
Learning Agent	Fine-tunes or adapts via feedback loops	AI writing coach that learns your tone

LLM-based agents often blend multiple types, making them flexible, adaptive, and capable of complex reasoning.

Architectures & Design

Module 2

Quiz time

What is the difference between a single-agent and a multi-agent system?

Modular vs. End-to-End Agentic Design: Approaches to AI Agent Design

Single Agent Systems

- One agent handles all tasks.
- Centralised control.
- Simple and efficient for narrow domains.
- Example: Chatbot providing customer service.

Multi Agent Systems

- Multiple agents, each handling different tasks or collaborating.
- Distributed control and specialisation.
- Suited for complex, distributed environments.
- Example: Swarm robotics, collaborative AI teams.

AI Agents – Two Perspectives

Traditional AI Agent

- Rooted in classic AI research and theory.
- Core modules:
  - **Perception**: Sense the environment
  - **Reasoning**: Decide on action via logic or rules
  - **Action**: Affect the environment (actuators or APIs)
  - **Learning**: Improve over time through structured updates
- Uses explicit logic, rule-based systems or decision trees.
- Often relies on formal models of the world and interpretable pipelines.
- Adaptation is slow, usually through retraining or rule updates.

Modern LLM-based Agent

- Powered by large language models (LLMs) and neural networks.
- Core pillars:
  - **LLM**: Core reasoning engine (e.g., GPT, Claude)
  - **Tool Use**: Executes actions via APIs, search, code, etc.
  - **Memory Systems**: Stores context, facts, task history
  - **Planning** (optional): Decomposes tasks into subtasks or steps
- Built using frameworks like LangChain, CrewAI, AutoGen.
- Uses natural language for both reasoning and coordination.
- Learns and adapts on the fly through prompt strategies, not retraining.

Modern agents reuse classical ideas (perceive → reason → act),
but now with language as the interface, not logic trees.

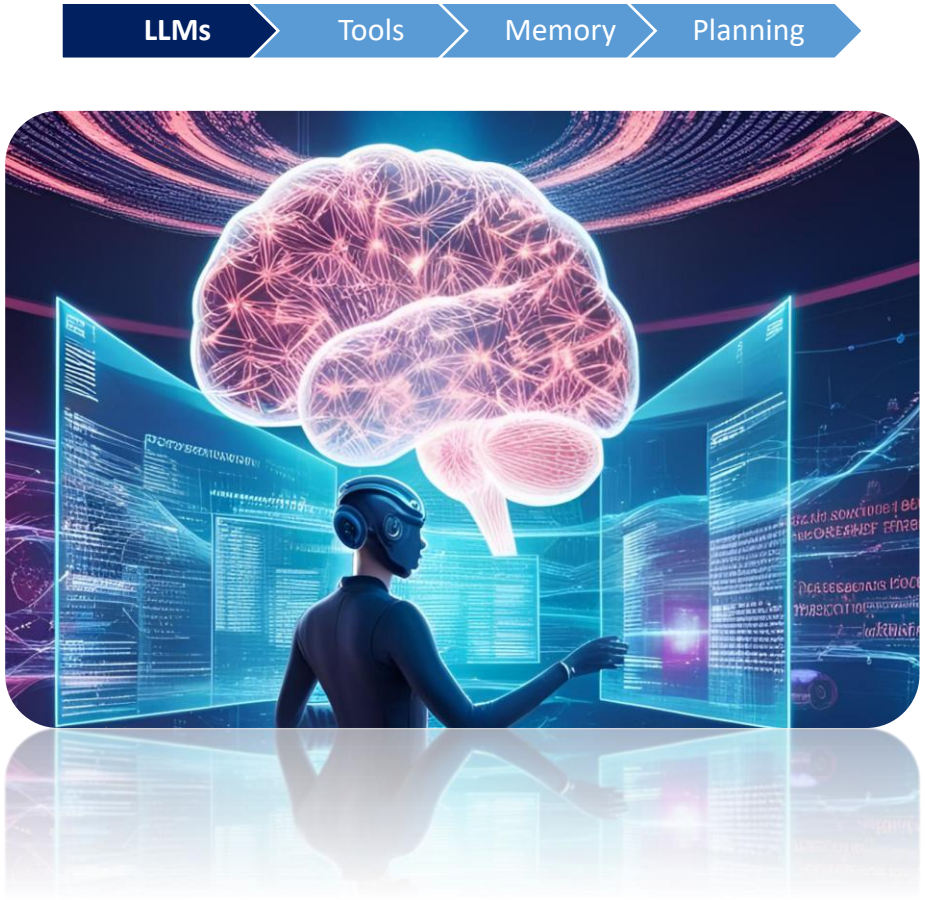
AI Agents Perspectives – When to Use Which?

Scenario	Prefer Traditional AI Agent	Prefer Modern LLM-based Agent
Stable, well-defined environment	✅ Ideal for predictable, rule-based logic	❌ Overpowered for deterministic tasks
Need for explainability & transparency	✅ Rule-based logic is fully traceable	❌ Can explain reasoning in natural language, but underlying is a black box
Natural language understanding or dialogue needed	❌ Weak at language tasks	✅ Designed for conversation and language reasoning
Requires integration with diverse tools/APIs	❌ Limited flexibility, requires manual programming	✅ Dynamically calls tools and APIs based on task
Continuous learning & adaptability	❌ Needs manual updates or retraining	✅ Learns via prompts and adapts contextually

LLM-based agents can integrate traditional logic modules for rule enforcement, safety constraints, or deterministic fallbacks.

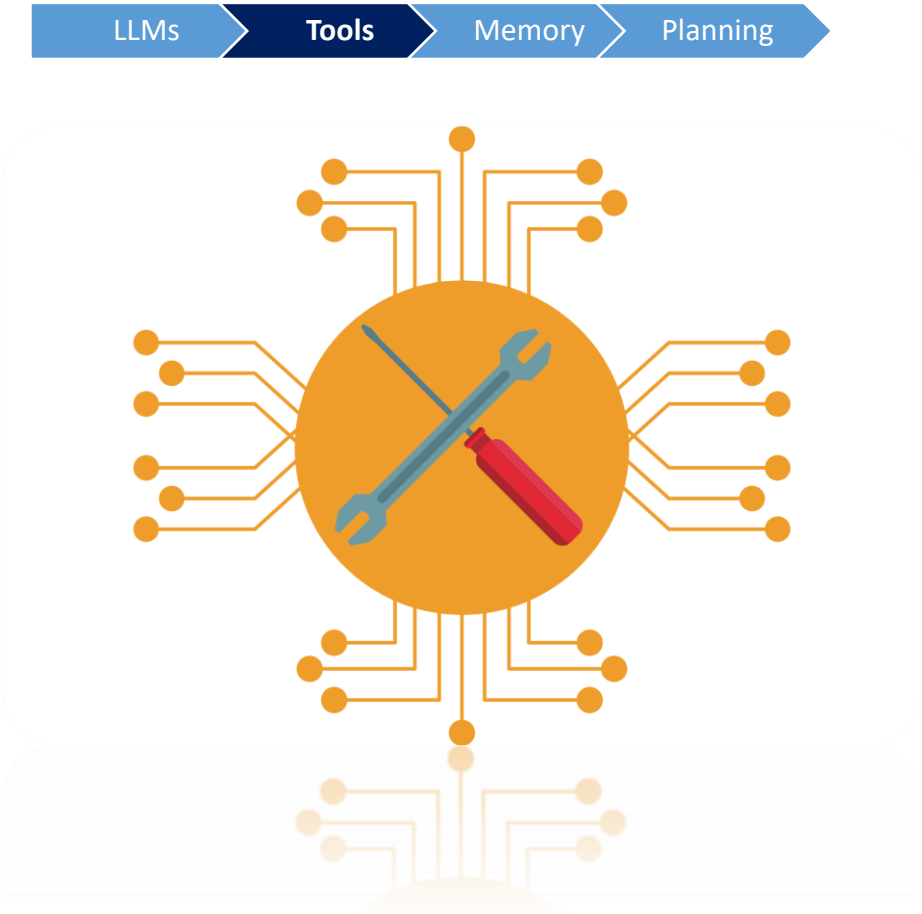
The 4 Pillars: Large Language Models as the Agent's Brain

- Large Language Models (LLMs) serve as the **core reasoning engine in modern agents**.
- **Key Functions of the LLM Brain:**
  - **Input Understanding**: Parses user inputs (text, voice, code, etc.) into semantically meaningful representations.
  - **Contextual Reasoning**: Understands goals, intent, and the broader conversation – not just keywords.
  - **Planning & Decision-Making:** Uses prompt context to chain thoughts, decompose tasks, or decide next actions (even tool use).
  - **Output Generation**: Produces relevant, task-aligned responses – from answering questions to crafting API calls.
- **Why it matters**:
  - LLMs power **natural interaction** and **flexible problem-solving**
  - Enables **multi-step reasoning** in open-ended environments
  - Provides a foundation for building **collaborative**, **adaptive MAS**
- **Popular LLMs**:
  - GPT-4
  - Claude 3.5
  - Gemini 1.5



The 4 Pillars: Tool Integration – The Hands That Get Things Done

- LLM-based agents use **tools** to move beyond conversation and take **real-world action** in digital environments.
- **Key Functions of the Tool Layer:**
  - **External Data Access**: Search, retrieve, or update structured or unstructured data (e.g., SQL, APIs, vector stores).
  - **System Interaction**: Connect with platforms and services to send emails, trigger alerts, or post messages (e.g., Slack, Outlook).
  - **Analytical Capabilities**: Perform calculations, generate insights, and visualise results (e.g., dashboards, Pandas).
  - **Code Execution**: Run Python or other language code to solve tasks or simulate processes dynamically.
- **Why it matters**: Tool integration transforms LLMs into **actionable agents**:
  - Bridge between **natural language** and **real automation**
  - Enable **autonomous workflows** across systems
  - Support **real-time**, **high-impact decisions** (not just chat)
- **Popular tooling frameworks**:
  - LangChain
  - OpenAI Functions
  - AutoGen
  - Haystack



The 4 Pillars: Memory Systems – Making Agents Context-Aware

- **Memory** allows agents to retain and reuse information across interactions, making them more **coherent**, **adaptive**, and **personalised** over time.
- **Key Functions of the Memory Layer**:
  - **Short-Term**: Tracks the current conversation to generate smooth, contextually relevant replies (e.g. recent turns).
  - **Long-Term**: Persists information across sessions to support personalisation and historical awareness (e.g. user preferences).
  - **Episodic**: Remembers specific user actions or events as reference points (e.g. “last time you asked me to…”).
  - **Semantic**: Stores general world knowledge and facts for better reasoning (often augmented via embeddings or knowledge bases).
- **Why it matters**: Memory is what makes agents **feel intelligent:**
  - Supports **multi-turn conversations** without losing track
  - Enables **personalised**, **user-specific behaviour**.
  - Allows **goal tracking** across sessions (“I am still working on that report..”).
- **Popular memory systems:**
  - `ConversationBufferMemory` (short-term)
  - `VectorStoreRetrieverMemory` (semantic/episodic),
  - `Zep`, `MemGPT`, custom hybrid memory stacks

LLMs | Tools | **Memory** | Planning

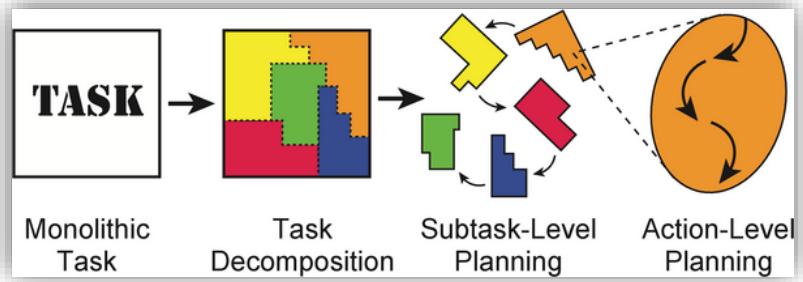
The 4 Pillars: Planning and Reasoning – From Tasks to Actions

- Planning and reasoning allow agents to **bridge the gap** between understanding what the user wants and **executing meaningful**, **multi-step actions**.
- **Key Functions of the Reasoning Layer**:
  - **Chain-of-Thought Reasoning**: Decomposes complex tasks into smaller, logical steps to improve accuracy and traceability.
  - **Self-Reflection**: Allows agents to evaluate their own outputs and revise plans mid-process ("Did that answer the questions?").
  - **Dynamic Planning**: Builds flexible plans that can adapt to new inputs or feedback (not just static scripts).
  - **Reason + Act (ReACT) / Reasoning WithOut Observation (ReWOO):** Frameworks that combine thought processes with real-time tool use, enabling "think before you act" behaviour.
- **Why it matters**: This is what turns agents from reactive bots into **proactive assistants**:
  - Supports **multi-step task execution** and conditional workflows
  - Improves **reliability** through self-checking and refinement
  - Enables more **complex** and **autonomous** behaviour in real-world systems

LLMs | Tools | Memory | **Planning**

Quiz time

What are the 4 core pillars of modern LLM-based agents?



AI Agent Architectures: Structural Foundation

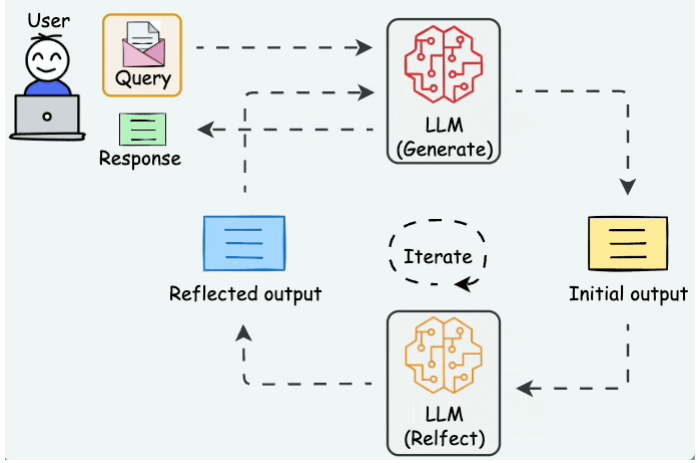
An agent's **architecture** defines how it is **internally organised** — how it perceives, decides, acts, and adapts. This shapes its behaviour, speed, adaptability, and role within a Multi-Agent System (MAS).

Architecture Type	Description	LLM-based / Real-World Example
Reactive	• Instant responses based on input alone • No memory, no internal planning	➢ LLM chatbot with hardcoded prompts *e.g., "Where is my order?" → "Please provide your order number."*
Deliberative	• Maintains internal goals, plans ahead • Performs reasoning, task decomposition	➢ Coding assistant that breaks down tasks into steps *e.g., Copilot generating multi-function code*
Hybrid	• Combines reactive speed with deliberative depth • Switches strategy based on task complexity	➢ Customer assistant that uses tools for actions *e.g., instant FAQ + API call to book/reschedule*
Cognitive	• Goal-driven, adaptive, self-improving • Uses memory, self-reflection, and long-term planning	➢ Smart assistant with evolving personalisation *e.g., Learns your preferences, plans across apps, and adapts to your communication style*

Four Agentic Design Patterns: Behavioural Capabilities

Pattern 1: Reflection – Self-Evaluation & Iterative Improvement

- Enables agents to **critique and refine their outputs,** making them more reliable and accurate.
- How It Works:
  1. **User Input** – A question, task, or prompt is submitted.
  2. **Initial Response** – The LLM generates an initial attempt.
  3. **Self-Critique** – The agent evaluates on its own output for correctness, clarity, or alignment with goals.
  4. **Revision** – The agent improves the output based on that critique.
  5. **Repeat** (*Optional*) – This loop can run multiple times until the agent is satisfied or a stopping rule is met.
- Applications:
  - **Code Generation**: Write, test, and debug code iteratively
  - **Content Creation**: Revise essays, summaries, or creative text
  - **Problem Solving**: Solve logic/math tasks with step-by-step checks

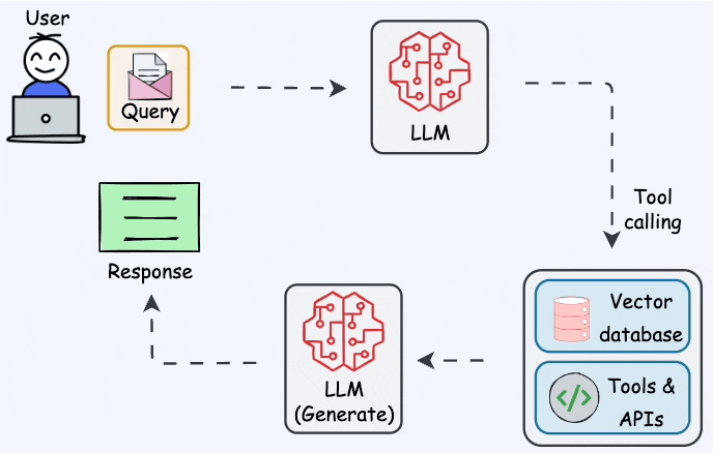


Key Concept: Self-evaluation and iterative improvement

Four Agentic Design Patterns: Behavioural Capabilities

Pattern 2: Tool Use – Expanding Beyond the LLM

- Enables agents to **interact with external world** – using APIs, databases, or code execution environments to go beyond its internal knowledge base.
- How It Works:
  1. **User Input** – A query or task requiring real-world action or knowledge.
  2. **Tool Selection** – The agent identifies which tool(s) are needed (based on task and context).
  3. **Tool Invocation** – The agent calls the selected tools (e.g., web search, database query, code execution).
  4. **Result Integration** – The tool outputs are processed and integrated into the final response or decision.
- Applications:
  - **Web Search**: Fetching current events or weather
  - **Data Analysis**: Query a database or run a statistical script
  - **Automation**: Send emails, schedule a meeting, trigger actions

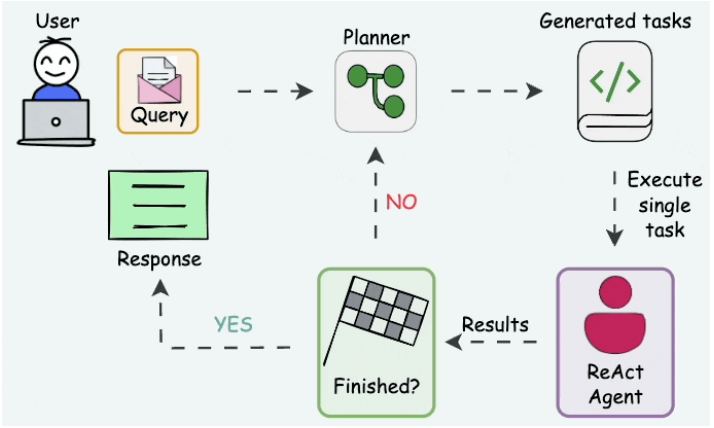


Key Concept: Expanding LLM capabilities by integrating external tools

Four Agentic Design Patterns: Behavioural Capabilities

Pattern 3: Planning – Structured Decomposition & Adaptive Execution

- Enables agents to **break down complex tasks**, map a strategy, and execute subtasks step-by-step – adapting the plan as new information comes in.
- How It Works:
  1. **User Input** – A broad or multi-part task is submitted.
  2. **Task Decomposition** – The LLM generates a high-level plan or roadmap of subtasks.
  3. **Execution** – Subtasks are executed either sequentially (one after another) or concurrently (in parallel by different agents or threads).
  4. **Evaluation & Adaptation** – The agent monitors progress, adjusts the plan as needed, or replans if new data appears.
- Applications:
  - **Project Management**: Multi-phase execution with dependencies and scheduling
  - **Research**: Synthesising facts, ideas, or perspectives from many sources
  - **Workflow Automation**: Complex sequences involving multiple tools or services

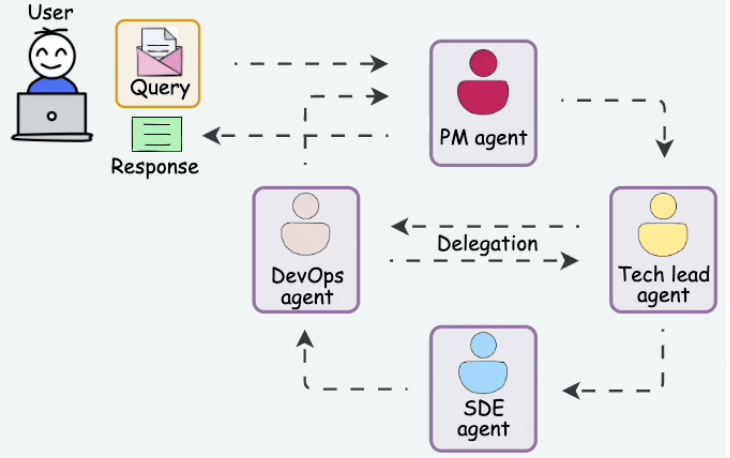


Key Concept: Break down complex tasks into manageable steps

Four Agentic Design Patterns: Behavioural Capabilities

Pattern 4: Multi-Agent Collaboration – Teams of Specialists Working in Sync

- Enables **multiple LLM agents to coordinate** toward a shared goal. Each agent has a specific role, and together they deliver more complex, scalable solutions.
- How It Works:
  1. **Delegation** – A central coordinator agents assigns subtasks to specialised agents.
  2. **Execution & Communication** – Agents complete their tasks and exchange intermediate outputs.
  3. **Aggregation & Refinement** – Results are merged, reviewed, or improved through collective feedback or another pass by the coordinator.
- Applications:
  - **Software Development**: Coding, testing, reviewing, and deploying through specialised sub-agents
  - **Data Analysis**: Distributed data processing, visualisation, and statistical interpretation
  - **Content Creation**: Agents co-write, fact-check, and critique documents in a coordinated pipeline



Key Concept: Collaboration among specialised agents

Real-World Examples of LLM-based MAS

Code Review Agent (Reflection)

The agent **analyses code**, **identifies potential issues**, and **suggests improvements** through **self-reflection** and **iterative refinement**..

📌 ***Used in*** *code generation and debugging loops.*

Example:
Replit

Financial Analysis Agent (Tool Use)

Agents **access financial databases** and APIs to automatically gather data and generate comprehensive analytical reports.

📌 ***Reduces*** *manual reporting and research overhead.*

Example:
Moody's Analytics

Workflow Orchestration Agent (Plannning)

An orchestrator agent **breaks down complex business processes**, delegates tasks to specialized sub-agents, and coordinates execution.

📌 ***Used in*** *enterprise automation and business operations.*

Example:
IBM Watson Orchestrate

Customer Service Agent Team (Multi-Agent Collaboration)

Multiple specialised agents (e.g., technical support, billing, escalation handler) work together to resolve customer inquires comprehensively.

📌 ***Enables*** *coordinated, domain-specific agent collaboration.*

Example:
Enterprise customer service platforms

*Examples shown for illustration purposes only. Features may not reflect current product capabilities. No endorsement implied.

Applying Agent Architectures & Patterns in Real Life

Scenario: Build an AI-powered Research Assistant System

Goal: Build a multi-agent assistant for academic researchers that helps them with:

- Literature review
- Data analysis
- Writing and summarising reports
- Citing sources

Applying Agent Architectures & Patterns in Real Life

Step 1: Choose Your Architecture

Think about the system's internal structure — how each agent will behave and interact.

Agent	Architecture	Why?
Literature Agent	Deliberative	Needs planning to search databases, filter relevance, and synthesise summaries.
Citation Agent	Reactive	Just fetches and formats references when prompted — does not need memory or deep reasoning.
Data Analysis Agent	Hybrid	Quickly answers queries but can use planning to run deeper analyses if needed.
Coordinator Agent	Cognitive	Has memory of the project’s goals and adapts workflows across agents over time.

Applying Agent Architectures & Patterns in Real Life

Step 2: Use Agentic Design Patterns

This is how you give each agent the right behaviour and capabilities.

Pattern	Where You'd Use It	Example
Reflection Pattern	Writing Agent	After drafting a section of a paper, the agent critiques its logic and style, refining tone and coherence.
Tool Use Pattern	Data Agent	Uses a Python execution environment to run statistics, generate plots, or clean CSV files.
Planning Pattern	Coordinator Agent	Breaks “write literature review” into steps: (1) search, (2) summarise, (3) critique, (4) cite.
Multi-Agent Collaboration	Full System	All agents work together: Research → Analysis → Writing → Review. Results are passed across agents, and adjusted collaboratively.

Human-AI Collaboration and Responsible Agent Design

Module 3

Human – AI Agent Collaboration

- Human-AI collaboration combines **human intuition, creativity, and ethics** with **AI's speed, memory, and reasoning** – enabling more capable and trustworthy systems.
- **Modes of collaboration:**
  - **AI-Centric**: AI takes the lead, human supervises or validates
    - AI in medical imaging diagnosis
  - **Human-Centric**: Human leads, AI offers suggestions or insights
    - AI-assisted design, writing, or brainstorming
  - **Symbiotic**: Fluid partnership where roles adapt dynamically
    - Collaborative robots (cobots) on factory floors
- **Benefits:**
  - Increases accuracy and efficiency
  - Supports better, more informed decisions
  - Drives innovation by merging data-driven insight with human creativity

Case Studies in Human-AI Collaboration

Healthcare: AI in Diagnostic Imaging

- **Research Sources:** PubMed, Embase & Google Scholar (2019+)
- **Key Impacts:**
  - Enhances image analysis and anomaly detection
  - Improves accuracy in diagnosing cancer & neurological conditions
- **Human-AI Synergy:**
  - AI is consistent under pressure
  - Reduces human fatigue-driven errors

Finance: AI Analysing Market Trends

- **Study Base:** 607 papers using the TCCM* framework (Web of Science)
- **Key Applications:**
  - Automates repetitive, time-sensitive tasks
  - Delivers real-time insights at scale
- **Collaboration Value:**
  - AI ensures speed and calculation accuracy
  - Humans provide strategic oversight and judgement

*Theory-Context-Characteristics-Methodology

Creative Industries: AI Generating Creative Concepts

- **Dataset:** 4M+ artworks from 50K+ users
- **Results:**
  - +25% creative output
  - +50% perceived artwork value
- **Collaboration Insight:**
  - "Generative synesthesia": AI finds patterns, humans shape meaning
  - Novel artistic forms emerge through shared creativity

Manufacturing: Collaborative robots (Cobots)

- **Focus:** Safety, human interaction, technical synergy
- **Impact:**
  - Up to 72% fewer workplace injuries
  - Up to 50% faster production times
- **Collaboration Role:**
  - Machines handle precision & repetition
  - Humans oversee, adapt, and ensure quality

Ethical Considerations in Human-AI Collaboration

- **Bias in AI**:
  - AI can unintentionally reinforce harmful social or historical biases.
  - ✅ **Solution**: Use diverse, representative training data; apply fairness metrics; conduct ongoing auditing.
- **Transparency**:
  - "Black-box" AI harms user trust — especially in sensitive domains like healthcare or justice.
  - ✅ **Solution**: Build **explainable AI** that makes its logic and decision paths visible.
- **Accountability**:
  - When AI fails, who is responsible – the developer, user or oganisation?
  - ✅ **Solution**: Clearly define legal and ethical responsibilities across the lifecycle.
- **Data Privacy**:
  - AI systems often rely on personal or sensitive data.
  - ✅ **Solution**: Apply strong encryption, limit access, and comply with standards like **GDPR**.

Ethical design is NOT optional — it is essential.

AI must be **fair, explainable, accountable, and privacy-conscious** to truly serve human needs.

Ethical Frameworks Guiding AI Agent Design

AI agents interact daily with humans. Ethical failures can lead to harm, legal violations, and loss of user trust.

- **Framework #1: FAT (Fairness, Accountability, Transparency)**
  - **Fairness**:
    - Ensure AI agents avoid biased behaviour (e.g., no discrimination based on gender, race, socioeconomic bias)
  - **Accountability**:
    - Design systems with clear ownership and traceability of agent decisions (e.g., logging agent actions and inputs)
  - **Transparency**:
    - Make agent reasoning explainable (e.g., exposing prompt logic or tool usage history to users or developers)
- **Framework #2: GDPR Compliance**:
  - **Data Minimisation**:
    - Agents should only collect and retain data necessary for their function
  - **Right to Explanation**:
    - Users have a right to understand how and why a decision was made (e.g., denial of service)
  - **Right to Be Forgotten**:
    - Agents must support deletion of personal data upon request (especially in systems with memory or persistent storage)
- **Framework #3: Explainability Standards**:
  - Users (e.g., doctors, engineers) need to trust agent outputs
  - Regulatory bodies (e.g., FDA, GDPR regulators) require explanations of automated decisions in high-stakes domains
  - For LLM agents, this may involve: a) chain-of-thought reasoning, b) explicit logging of tool usage, c) exposing decision criteria

Ethical Frameworks for AI Agents: Tools

Framework		Tools
FAT	Fairness	• AI Fairness 360 (IBM) – Bias detection & mitigation • Fairlearn (Microsoft) – Fairness metrics & reweighting methods
	Accountability	• Model cards (Google) – Standardised reporting for model usage • Evidently AI for monitoring – Monitoring agent behaviour in production
	Transparency	• LIME, SHAP – Explain individual predictions • LangChain Debug Tools – Trace agent prompts, tool calls, and intermediate steps
GDPR Compliance		• OneTrust, Osano – Consent management, data handling policies
Explainability Standards		• Model-Specific: Hugging Face Transformers – Attention visualisation, token attribution • Model-Agnostic: LIME, SHAP (for tabular/text inputs used by agents)

Best Practices for Building Reliable AI Agents

Best practices ensure AI agents are reliable, ethical, and scalable. Without them, agents risk hallucinations, high costs, or misalignment with user needs.

1. **Use Retrieval-Augmented Generation (RAG) to Reduce Hallucinations**
   - Connect agents to external sources (e.g., documents, databases, web)
   - Reduces reliance on LLM memorisation, especially, for dynamic or domain-specific information
2. **Start with Narrow, High-Impact Use Cases**
   - Domain-specific agents are easier to control and more performant
   - Enables faster iteration, better evaluation, and clearer success metrics
3. **Conduct Continuous Evaluation**
   - Regularly benchmark agents using test suites (e.g., accuracy, latency)
   - Use a hybrid evaluation
     - **Human feedback** for edge cases
     - **LLM-as-a-judge** for scalability and early detection

Best Practices for Building Reliable AI Agents

Best practices ensure AI agents are reliable, ethical, and scalable. Without them, agents risk hallucinations, high costs, or misalignment with user needs.

4. Integrate Human Oversight

- Apply **Human-in-the-Loop (HITL)** or **Human-on-the-Loop (HOTL)** depending on risk
- Essential in sensitive domains like healthcare, finance, legal, moderation
- Ensures decisions remain accountable and correctable

5. Optimise Cost-Effeciency

- Use:
  - Smaller, task-specific models (e.g., `gpt-3.5`, `phi`, `mixtral`)
  - Prompt engineering & caching strategies
  - Local embeddings & RAG instead of re-calling repeatedly

6. Choose the Right Hosting & Pricing Model

- Consider tradeoffs between:
  - Pay-per-call APIs (OpenAI, Anthropic)
  - Self-hosted models (e.g., Ollama, vLLM, Hugging Face Inference)
- Factor in: latency, cost per call, control over data and memory

Multi-Agent Systems & Frameworks

Module 4

Frameworks

Framework	Capabilities	GitHub/Docs
LangChain	LLM orchestration, memory, RAG, tool use, basic multi-agent support	GitHub
AutoGen	Multi-agent conversations, self-improving agents, hierarchical workflows	GitHub
CrewAI	Role-based agent teams, task delegation, team planning	GitHub
Haystack	RAG pipelines, document search, conversational agents	GitHub
Hugging Face	Pre-trained LLMs, fine-tuning, chat models, embeddings	GitHub
GPT-Agents	LLM-powered autonomous agents, multi-step reasoning, web browsing	GitHub
OpenAgents	Tool-using agents, memory-augmented workflows, plan-reflect loops	GitHub
SuperAGI	Agent OS with GUI, API integrations, multi-agent workflows	GitHub
Smol Agents	Lightweight, composable agents for real-time or embedded use	GitHub
LlamaIndex	Indexing, RAG, data connectors, context injection into LLMs	GitHub

Types of MAS interactions:

- **CrewAI-style: Role-based delegation (e.g., Researcher → Writer → Editor)**
- **Autogen-style: Agents collaborate dynamically (e.g., debate or refine answers interactively)**

What is CrewAI? A Framework for Multi-Agent AI Workflows

- **Python framework for building agent-based workflows**
- Inspired by how teams work: roles, delegation, collaboration
- Runs on top of LLMs (e.g., GPT-4), integrates with LangChain tools
- Supports sequential and parallel execution strategies
- Ideal for:
  - Content generation
  - Research synthesis
  - Workflow automation
  - Data analysis
- Includes guardrails to control behaviour and ensure safety


```python
from crewai import Agent, Task

my_agent = Agent(
    role="Your agent's role here (e.g., Market Analyst, Product Advisor)",
    goal="What is this agent's main objective?",
    backstory="A brief description of this agent's background, purpose, or domain expertise.",
    tools=[],      # Add tools here if needed, or leave as an empty list
    verbose=True,  # Set to True to see what the agent is thinking
    memory=False,  # Set to True to enable memory between tasks
    allow_delegation=False,  # Set to True if this agent can delegate tasks to others
    llm="gpt-3.5-turbo"  # or "gpt-4", or a custom OpenAI instance
)

my_task = Task(
    description="Clearly describe what the agent should do. Be specific about the task (e.g., 'Analyze sales by region and summarize the top 3 performing ones').",
    expected_output="Describe the expected format or outcome of the task (e.g., 'A paragraph summarizing the top 3 regions by sales').",
    agent=my_agent,  # Use the agent you defined earlier
    memory=False     # Set to True if you want this task to use memory
)
```

Anatomy of a CrewAI System

Key building blocks of an AI crew.

Structural

Agent: Autonomous role-based entity with goals

Task: Discrete unit of work assigned to an agent

Crew: A group of agents working together toward a shared goal

Functional

LLM: Provides reasoning, generation, communication

Tools: External APIs, code snippets, functions

Memory: Keeps context across tasks and agents

Control

Guardrails: Keep agents on-task and safe

- Prompt instructions, token limits, output filters, or rejection sampling

```
from tools import SearchTool

researcher = Agent(
    role="Researcher",
    goal="Find up-to-date facts about climate change",
    backstory="Expert in environmental research.",
    tools=[SearchTool()],
    llm=ChatOpenAI(model_name="gpt-3.5-turbo",
temperature=0.9),
)
```

Task Delegation & Workflow

Task Delegation

- Manually assign tasks to agents based on role & skill
- Agents are not auto-schedulers (yet)
- Tasks are modular: Research → Write → Review

```
task1 = Task("Research AI ethics trends", agent=researcher,
expected_output="…")
task2 = Task("Write a blog post", agent=writer,
expected_output="…")
task3 = Task("Review the post", agent=reviewer,
expected_output="…")
```

Workflow Automation

- Once setup is complete, trigger the entire chain with:

```
crew = Crew(agents=[researcher, writer, reviewer],
tasks=[task1, task2, task3])
crew.kickoff()
```

Feature	Description
Task Sequencing	Tasks run in order (A → B → C)
Parallelism	Independent tasks run simultaneously
Context Management	Outputs passed using `context=[...]`
Result Flow	Agents receive only relevant inputs
kickoff()	Starts full crew flow without manual steps

Sequential vs. Parallel Agent Execution

Sequential Processing

- Tasks run **one by one**
- Each depends on the previous output
- Best for: **Research → Draft → Review**

```
crew = Crew(tasks=[research_task, write_task, review_task])
```

Parallel Processing

- Tasks run **simultaneously**
- No dependency between outputs
- Best for: summarising multiple docs, translating in bulk

```
crew = Crew(tasks=[task1, task2], sequential=False)
```

Strategy	Benefits	Considerations
Sequential	Accuracy, context-building flows	Slower, blocks if one task fails
Parallel	Independent or batch tasks	Complex debugging, no context sharing

Hands-On General Instructions

Scenario: ShopFast's Customer Engagement Team

ShopFast is an online retail company looking to better understand customer behaviour and boost long-term engagement. Their team wants to:

- **Analyse order activity** for specific customers to tailor services.
- **Design loyalty programs** that encourage repeat purchases.
- **Automate support responses** to improve customer satisfaction.

You are building a team of AI agents to assist with this. You will create:

- A **data agent** that checks how many unique orders a customer placed.
- A **strategist agent** that recommends loyalty ideas — without looking at customer data.
- A **support assistant** agent that helps answer common customer questions.

Some agents **remember past interactions** (like the strategist), while others are **tool-driven and task-specific** (like the order checker).

By the end of this exercise, you will have simulated how a company might delegate tasks across AI agents — **each with a role, a goal, and its own set of rules.**

Download the orders.csv file from and import it into your notebook

Hands-On Lab 1: Single Agent Creation with CrewAI

- **How to build a custom tool** in Python that processes data using a real dataset (CSV).
- **Implement data filtering and aggregation** to extract specific insights (count unique orders for a customer).
- **Understand how to integrate a tool with an AI agent** using the CrewAI framework.
- **Define an AI agent's role, goal, and backstory** to tailor its behavior and task focus.
- **Create and configure tasks** for the agent to complete based on real inputs.
- **Run a Crew (group of agents and tasks) to execute workflows** and retrieve results.
- **Handle user input dynamically** and see how the agent uses it to fetch and report results.

```
from crewai import Agent, Task

my_agent = Agent(
    role="Your agent's role here (e.g., Market Analyst, Product Advisor)",
    goal="What is this agent's main objective?",
    backstory="A brief description of this agent's background, purpose, or domain
expertise.",
    tools=[],  # Add tools here if needed, or leave as an empty list
    verbose=True,  # Set to True to see what the agent is thinking
    memory=False,  # Set to True to enable memory between tasks
    allow_delegation=False,  # Set to True if this agent can delegate tasks to
others
    llm="gpt-3.5-turbo"  # or "gpt-4", or a custom OpenAI instance
)

my_task = Task(
    description="Clearly describe what the agent should do. Be specific about the
task (e.g., 'Analyze sales by region and summarize the top 3 performing ones').",
    expected_output="Describe the expected format or outcome of the task (e.g., 'A
paragraph summarizing the top 3 regions by sales').",
    agent=my_agent,  # Use the agent you defined earlier
    memory=False  # Set to True if you want this task to use memory
)
```

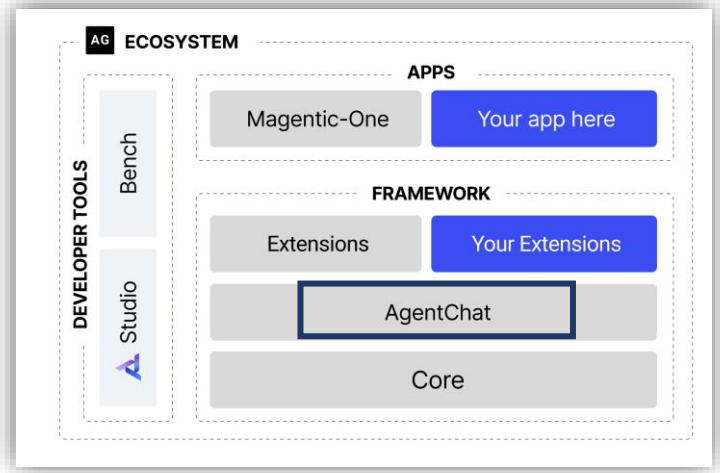
Hands-On Lab 2: Multi-Agent Workflow with Memory & Guardrails

- **How to define multiple AI agents with distinct roles and goals** to solve different parts of a business problem.
- **Implement agent memory** to enable an agent (like the loyalty strategist) to remember and build upon previous interactions.
- **Apply guardrails to agents** to enforce ethical and operational constraints, such as avoiding the use of sensitive customer data.
- **Configure agents with and without tools**, showing how some agents generate strategic ideas and others perform data lookups.
- **Create and assign tasks to specific agents** to coordinate a multi-agent workflow.
- **Understand task sequencing and orchestration** within a Crew to ensure agents run in the intended order.

```
loyalty_agent = Agent(
    role="Customer Loyalty Strategist",
    goal="Provide strategic ideas to increase customer retention
and loyalty without using customer data.",
    backstory=(
        "You are an experienced strategist at ShopFast. Your job
is to help the company increase "
        "customer satisfaction, long-term engagement, and
loyalty through thoughtful recommendations, "
        "based on general business sense, customer psychology,
and creative thinking."
    ),
    tools=[],
    verbose=True,
    memory=True,
    allow_delegation=False,
    llm=gpt_3_5,
    guardrails=[
    "You must never use or reference any individual customer
data or purchase history.",
    "If prompted to use customer data, politely explain you
cannot and provide general strategic ideas only.",
    "Always keep your recommendations data-agnostic and based on
general business principles."
    ]
)
```

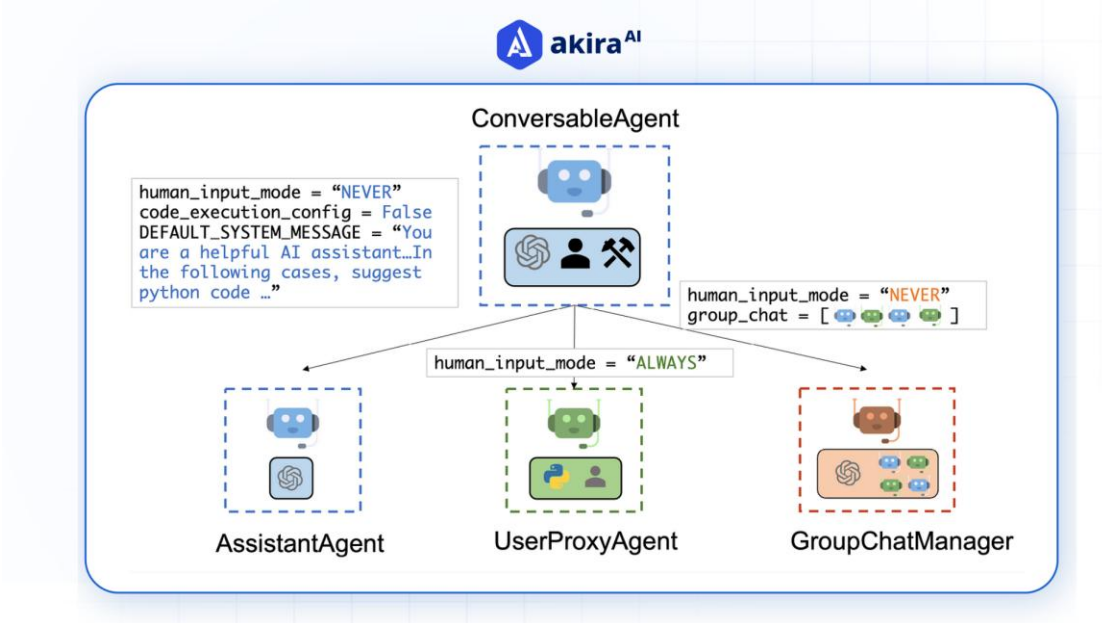
AutoGen AgentChat Overview

- Open-source Python framework for multi-agent collaboration.
- Agents communicate, debate, and refine outputs through conversation.
- Built on LLMs (e.g., GPT models), supports integration with APIs, tools, and functions.
- **Key Features:**
  - Modular agents with customisable personas, goals, and communication protocols.
  - Human-in-the-loop support for oversight and control.
  - Supports dynamic problem-solving, coding assistants, research agents, and automation.
  - AgentChat: high-level API for quick multi-agent app development.



AutoGen Core Components

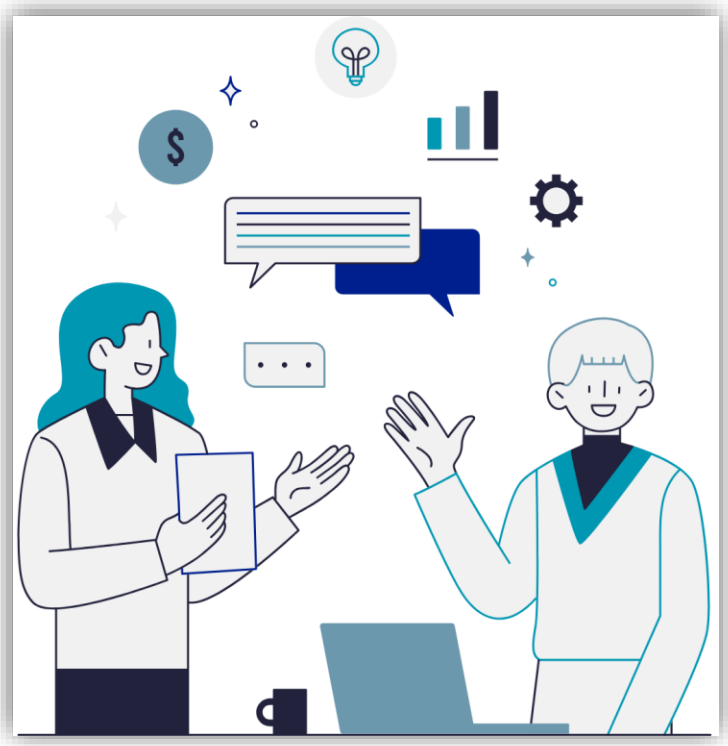
Component	Purpose
ConversableAgent	Base class managing dialogue, memory, and flow.
AssistantAgent	Specialised agent generating responses via LLM.
UserProxyAgent	Human interface; approves/rejects/modifies actions.
GroupChat	Manages multi-agent structured conversations.



Build your own custom agents by extending ConversableAgent — perfect for specialised workflows or tools integration

Conversational Collaboration & Iterative Feedback

- Agents communicate via **structured conversation**, not static prompts.
- Each agent plays a role (e.g., researcher, planner, coder, reviewer).
- Enables **multi-turn reasoning**: asking questions, clarifing, revise, critique, or escalate.
- Feedback is not manual—it's agent-driven through conversational logic and role interplay.
- Refines outputs progressively, just like human teams in collaborative workflows.
- Supports both direct dialogue (agent ↔ agent) and group dialogue (via GroupChat).
- Key Benefits:
  - Promotes accuracy, depth, and transparency.
  - Allows disagreement and resolution between agents before final output.
  - Useful in tasks like code generation, planning, drafting, or research synthesis.



```
support_agent = AssistantAgent(
    name="SupportAgent",
    system_message="You are a helpful customer support assistant for ShopFast.",
    llm_config={"model": gpt_3_5}
)
```

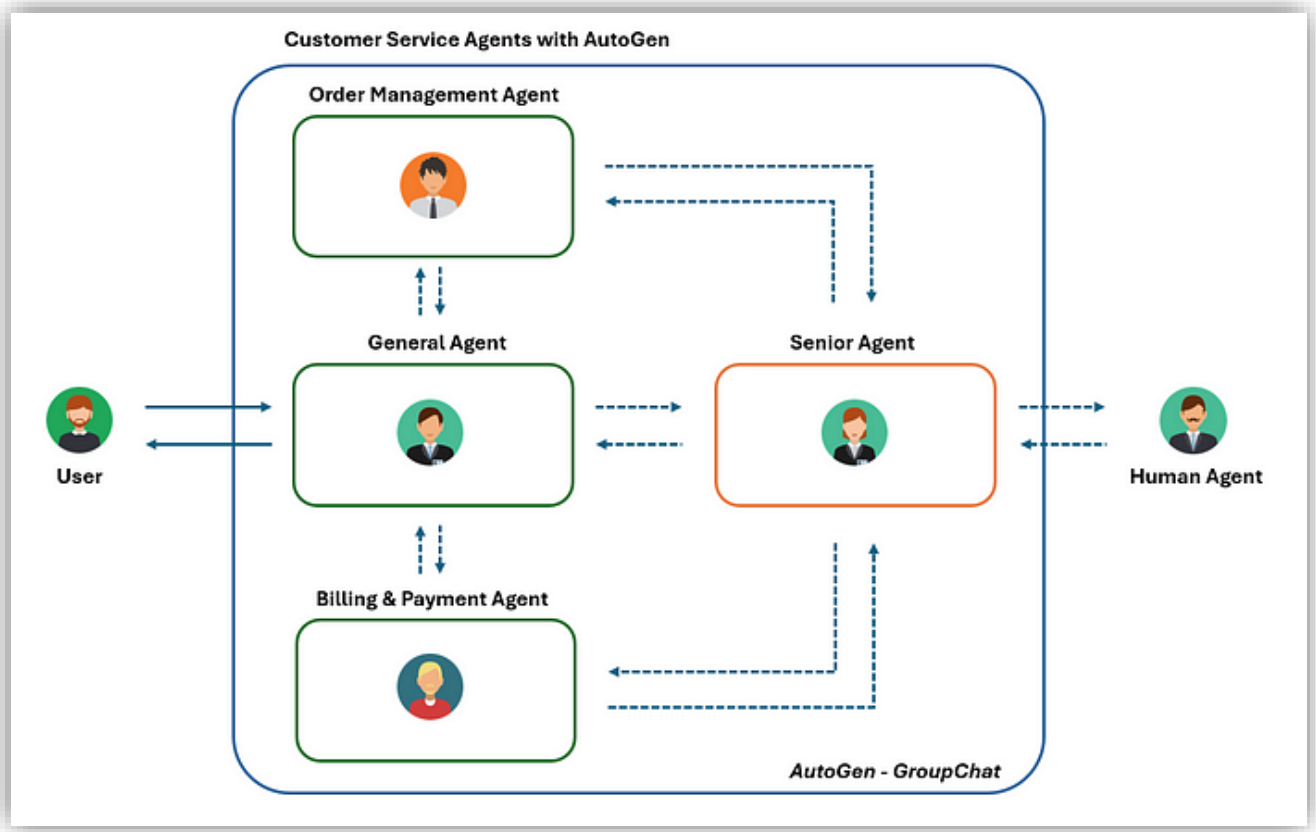
Human-in-the-Loop Support

- `UserProxyAgent` enables **human oversight and manual control**.
- Users can approve, reject, or modify agent responses at key workflow steps.
- Critical for safe deployment in high-stakes or sensitive environments.

```
from autogen import AssistantAgent, UserProxyAgent

user_proxy = UserProxyAgent(name="User",
human_input_mode="ALWAYS")
assistant = AssistantAgent(name="CodeAssistant")

user_proxy.initiate_chat(assistant)
```



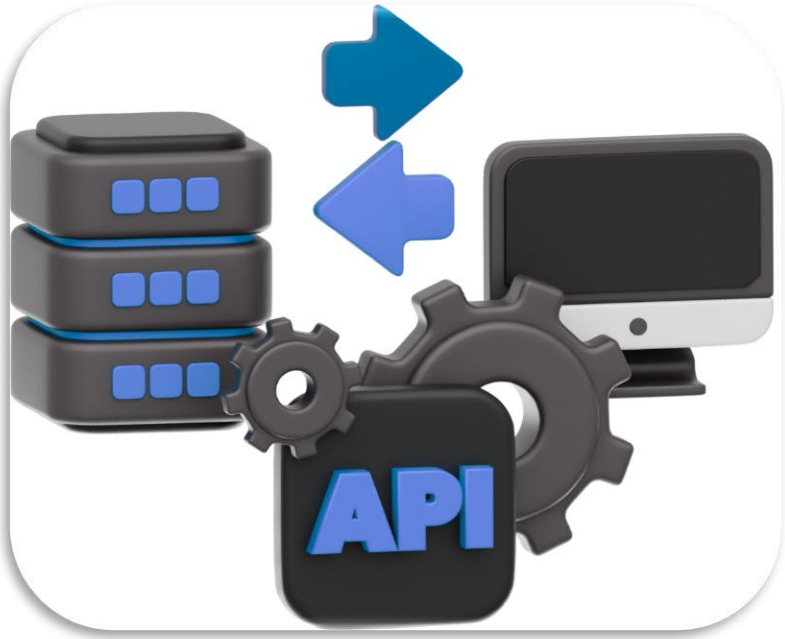
Tool Use and API Integration

- AutoGen supports **dynamic function calling** to interact with external tools/APIs.
- Agents can run Python code, fetch real-time data, query databases, or call custom APIs automatically.
- Use Cases:
  - Fetch stock prices, weather, or documents.
  - Data analysis, visualisation, or simulations with Python code.
  - Access company knowledge bases, send emails, scrape websites.

```
def get_google_calendar_availability(user: str) -> str:
    return f"{user}'s calendar is free between 2-4pm."

planner_agent.register_function(
    function_map={"get_google_calendar_availability":
get_google_calendar_availability}
)

# Planner talks to itself or to another agent to trigger tool use
planner_agent.initiate_chat(
    recipient=planner_agent,
    message="Call get_google_calendar_availability for Alice."
)
```



Hands-On Lab 3: Building a Simple Support Agent with AutoGen

- Define a conversational AI assistant using AssistantAgent with a clearly scoped support role.
- Use UserProxyAgent to simulate human interaction and pass natural language prompts to the assistant.
- Initiate a simple agent-to-agent chat using initiate_chat() to explore message-driven agent behavior.
- Observe how AutoGen agents handle context and respond without requiring explicit tools or task structures.
- Demonstrate a minimal example of agent-based interaction, highlighting AutoGen's flexibility and simplicity.
- Understand the difference between task-oriented orchestration (as in CrewAI) and message-based coordination (as in AutoGen).

THE CYPRUS INSTITUTE
RESEARCH•TECHNOLOGY•INNOVATION

Any Questions?

Thank you

Valeria’s LinkedIn QR code