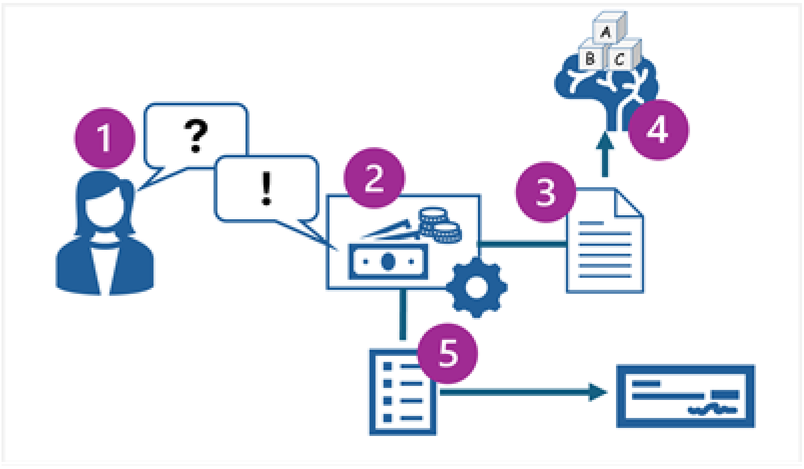
AI Agents

Introduction to AI Agents

AI agents are smart software services that combine generative AI models with contextual data and the ability to automate tasks based on user input and environmental factors that they perceive.

For example, an organization might build an AI agent to help employees manage expense claims. An example of the expenses agent scenario is shown in the following diagram.



The diagram shows the following process:

1. A user asks the expense agent a question about expenses that can be claimed.
2. The expenses agent accepts the question as a prompt.
3. The agent uses a knowledge store containing expenses policy information to ground the prompt.
4. The grounded prompt is submitted to the agent's language model to generate a response.
5. The agent generates an expense claim on behalf of the user and submits it to be processed and generate a check payment.

Core components of an AI Agent

- **Environment:** The external world the agent interacts with (web APIs, users, sensors, robots).
- **Percepts / Sensors:** Inputs the agent receives (text, images, sensor readings, API responses).

- **Actuators / Actions:** Outputs the agent performs (API calls, messages, motor commands).
- **Policy / Decision module:** The algorithm that maps state → action (could be rules, an ML model, planning).
- **Memory / State:** Short/long-term memory the agent uses to maintain context across steps.
- **Goals / Utility:** What the agent tries to accomplish (explicit goals, or maximize a reward/utility).

Types of AI Agents

AI agents can be categorized based on their capabilities, internal representations, and decision-making strategies. Here are the main types, with definitions, simple diagrams, and real-world examples.

1. **Simple Reflex Agents**

   Definition:

   - Act **only on the current percept**, ignoring history.
   - Decision-making is rule-based: *if condition → then action*.
   - Works well in fully observable, simple environments.

   Flow Diagram:

   [Percept] → [Condition-Action Rules] → [Action]

   Example:
   - Thermostat: If temperature < 20°C → Turn on heater; else → Turn off heater.
   - Elevator controller: If "up" button pressed → Move up.

   Pros: Simple, fast.
   Cons: Fails in complex environments needing memory.

2. **Model-based Reflex Agents**
   Definition:
   - Maintain an **internal model** of the environment.
   - Can handle **partially observable environments** by tracking what's not directly visible.
   - Decisions depend on both the **current percept** and the **internal state/model**.

   Internal state is the agent's internal representation or memory of past observations and the environment's unobserved aspects.

Flow Diagram:
[Percept] → [Update Internal Model] → [Condition-Action Rules] → [Action]

Example:

- Robot vacuum cleaner: remembers areas already cleaned, avoids repeating them.
- Self-driving car: maintains an internal map of nearby vehicles and pedestrians, even if temporarily out of sight.

Pros: Better adaptability, can reason about unseen states.
Cons: Complexity increases; requires accurate model.

3. Goal-based Agents

Definition:

- Go beyond reflexes: decisions are guided by **goals** (desired outcomes).
- Agents evaluate possible actions by checking if they help achieve the goal.
- Introduces **planning** capability.

Flow Diagram:

[Percept + State] → [Goal Evaluation / Planner] → [Action]

Example:

- GPS navigation system: selects routes based on the goal (destination).
- Personal assistant AI: chooses to order food only if the goal is "I'm hungry."

Pros: Flexible, can handle new goals.
Cons: Goal search may be computationally expensive.

4. Learning Agents

Definition:

- Can **learn from past experiences** and improve performance over time.
- Have four main components:
    1. **Learning Element:** Improves performance from experience.
    2. **Performance Element:** Selects actions.
    3. **Critic:** Evaluates actions vs desired outcome.
    4. **Problem Generator:** Suggests new actions for exploration.

Flow Diagram:
[Percept] → [Learning Element ↔ Critic/Feedback] → [Performance Element] → [Action]

Example:

- Chess-playing AI (AlphaZero): improves by playing millions of games against itself.
- Voice assistants: learn user's accent, vocabulary, and preferences over time.

Pros: Adaptive, can handle dynamic environments.
Cons: May need lots of data; risk of learning incorrect behavior.

Agentic AI Workflows

An agentic AI workflow is one in which an AI system:

1. Operates autonomously after being given a high-level task.
2. Iteratively refines its approach by reasoning in loops.
3. Uses memory and planning to structure multi-step problem solving.
4. Employs tools/APIs to gather information or take actions.
5. Self-corrects when errors or dead-ends occur.

In contrast to a static ML model, an agent in a workflow *acts like a problem-solving collaborator* that can adapt and evolve its plan as it goes.

Core Components of Agentic Workflows

1. **Memory**
   - *Short-term (working memory):* Stores current context during reasoning loops.
   - *Long-term memory:* Remembers past interactions, documents, or results.
   - *Why important:* Prevents repetition and enables personalization.
2. **Planning**
   - Breaks tasks into sub-tasks, sequences them intelligently.
   - May use reasoning chains ("Chain-of-Thought") or planners (e.g., ReAct).
3. **Tool Use**
   - External actions like: search engines, APIs, databases, calculators.
   - Lets the agent extend beyond its training knowledge.

4. **Self-Correction**
   - Reflection or evaluation steps.
   - If an answer is low quality, agent retries with modified reasoning.

    - Sometimes uses a “critic” agent to check a “generator” agent.

Frameworks & Libraries for Agentic Workflows

- **LangChain**
    - The most popular framework for building LLM-powered agents.
    - Provides primitives for memory, tools, reasoning chains, orchestration.
- **AutoGPT**
    - Early open-source project showing autonomous task execution.
    - Given a goal, decomposes tasks, uses tools, executes iteratively.
- **LlamaIndex (formerly GPT Index)**
    - Focused on data integration & retrieval augmentation.
    - Allows agents to connect to private documents, databases, APIs.
- **Haystack**
    - Open-source framework for RAG (Retrieval-Augmented Generation) and pipelines.
    - Supports modular workflows for search, question answering, and agent orchestration.

These frameworks provide building blocks to implement memory, planning, tool use, and multi-agent orchestration.

Orchestration of AI Agents

1. Why Orchestration is Needed

- In real-world systems, no single agent can handle everything.
- Multiple agents (specialized for tasks like retrieval, analysis, visualization, etc.) must collaborate.
- Without orchestration, agents may act redundantly, get stuck in loops, or fail to share information effectively.
- Orchestration = coordination, scheduling, and communication between agents to achieve a larger goal.

1. **Tools for Orchestration**
    - **LangChain’s Agent Executor**

    - Allows chaining and managing multiple agents.
    - Handles routing of tasks (decides which agent to call at what step).
    - Example: Directs one agent to search the web, another to summarize, and another to generate insights.
- **CrewAI**
    - Designed for collaborative multi-agent systems.
    - Agents can assume roles (e.g., "Researcher," "Analyst," "Writer").
    - Orchestrator manages dialogue and task allocation.
- **AutoGen (Microsoft)**
    - Multi-agent orchestration framework for conversation-based workflows.
    - Agents can talk to each other, exchange reasoning, and refine outputs collaboratively.
- **LLMOps Orchestration**
    - Goes beyond individual frameworks — focuses on operationalizing AI agents.
    - Includes:
        - ➢ Prompt Management → Standardizing prompts across workflows.
        - ➢ State Management → Tracking agent states, conversations, and goals.
        - ➢ Chaining → Executing multi-step reasoning loops.

Options for agent development

1. **Azure AI Foundry Agent Service**
   Azure AI Foundry Agent Service is a managed service in Azure that is designed to provide a framework for creating, managing, and using AI agents within Azure AI Foundry. The service is based on the OpenAI Assistants API but with increased choice of models, data integration, and enterprise security; enabling you to use both the OpenAI SDK and the Azure Foundry SDK to develop agentic solutions.

   For more information about Foundry Agent Service, visit:

   https://learn.microsoft.com/en-us/azure/ai-foundry/

2. **OpenAI Assistants API**

   The OpenAI Assistants API provides a subset of the features in Foundry Agent Service, and can only be used with OpenAI models. In Azure, you can use the Assistants API with Azure OpenAI, though in practice the Foundry Agent Service provides greater flexibility and functionality for agent development on Azure.

For more information about using the OpenAI Assistants API in Azure, visit: https://learn.microsoft.com/en-us/azure/ai-foundry/openai/how-to/assistant

3. **Semantic Kernel**
   Semantic Kernel is a lightweight, open-source development kit that you can use to build AI agents and orchestrate multi-agent solutions. The core Semantic Kernel SDK is designed for all kinds of generative AI development, while the *Semantic Kernel Agent Framework* is a platform specifically optimized for creating agents and implementing agentic solution patterns.

   For more information about the Semantic Kernel Agent Framework, visit: https://learn.microsoft.com/en-us/semantic-kernel/frameworks/agent/?pivots=programming-language-csharp

4. **Copilot Studio agent builder in Microsoft 365 Copilot**
   Business users can use the *declarative* Copilot Studio agent builder tool in Microsoft 365 Copilot to author basic agents for common tasks. The declarative nature of the tool enables users to create an agent by describing the functionality they need, or they can use an intuitive visual interface to specify options for their agent.

   For more information about authoring agents with Copilot Studio agent builder, visit: https://learn.microsoft.com/en-us/microsoft-365-copilot/extensibility/copilot-studio-lite-build

Example: Multi-Agent Orchestration Scenario

Use case: Building a Data-driven Insights Dashboard

1. **Data Retrieval Agent →** Pulls relevant data from APIs or databases.
2. **Analysis Agent →** Cleans and analyzes the dataset (e.g., applying ML models).
3. **Visualization Agent →** Prepares charts, graphs, and summaries for users.
4. **Orchestrator →** Determines the workflow execution order, ensures smooth information transfer, handles failures, and provides the final integrated output.

Flow Diagram:

[User Query]

↓

```
[Orchestrator Agent]
     ↓
┌──────────┬─────────┬───────────┐
|Retrieval  | Analysis  | Visualization |
|  Agent    |  Agent    |   Agent       |
└──────────┴─────────┴───────────┘
     ↓
 [Final Integrated Report]
```