

The world goes crazy with AI agents



Pavol Jesensky
EMEA AIOPS SME
AWS practice lead

CIC Slovakia
pjesensk@sk.ibm.com @pjesensk



Have you already
created or worked
with AI agent?

Game: Agent, assistant or automation?

Dynamic Pricing on a flight booking website that changes the ticket cost every few hours based on real-time demand, competitor prices, and remaining seats.

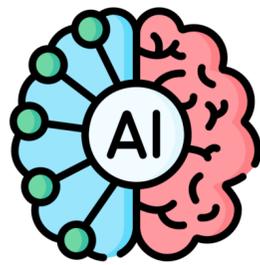
Email Spam Filter that automatically moves any email containing a known malicious URL or a specific, pre-defined set of keywords to the junk folder.

An Internal HR Chatbot that allows an employee to ask, "How many vacation days do I have left?" and instantly pulls the exact number from the HR database and displays it.

Smart Reply in your email app that suggests three short, one-click response options ("Sounds good!", "I'm busy that day," "Can we reschedule?") based on the content of the incoming message.

Autonomous Trading Bot that monitors the stock market 24/7, analyzes complex trend patterns, and executes buy or sell orders based on its own predictive models to meet a long-term profit goal.

A "Plan My Trip" Feature where you input "Find a 3-day trip to Paris next month" and the system provides a curated itinerary including flight, hotel, and activity suggestions, all of which require you to click "Book" or "Confirm."



The brain

	Traditional AI Models	Foundational Models (FMs)	Large Language Models (LLMs)
Focus/Goal	Specific, narrow task (e.g., image classification, simple regression).	General-purpose tasks; adaptation via fine-tuning (e.g., text, vision, code).	Specialized type of FM, focused on human-like language understanding and generation.
Training Data	Small to moderate, task-specific, labeled data.	Massive, diverse, unlabeled datasets (internet scale).	Massive text and code datasets.
Capability	High performance on one specific, trained task.	Emergent Abilities (capabilities not explicitly programmed), zero-shot, few-shot learning.	Contextual understanding, complex reasoning, creative text generation, conversation.
Architecture	Varied (e.g., Decision Trees, SVM, simple Neural Networks).	Typically Transformer architecture.	Typically Transformer architecture (specialized for sequence data).
Cost/Scale	Relatively low.	Very high (computation, data, time).	Very high.

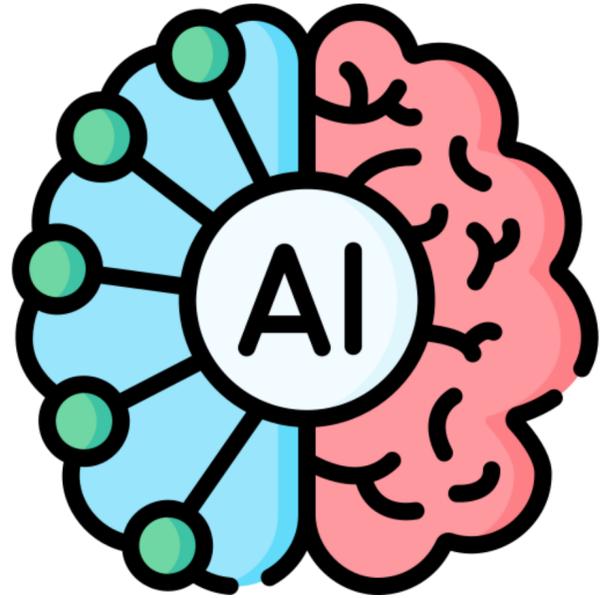
Key Takeaway: LLMs are a specialized type of Foundational Model.

Both FMs and LLMs represent a shift towards large, general-purpose models, while **Traditional AI** models are built for single, narrow objectives.

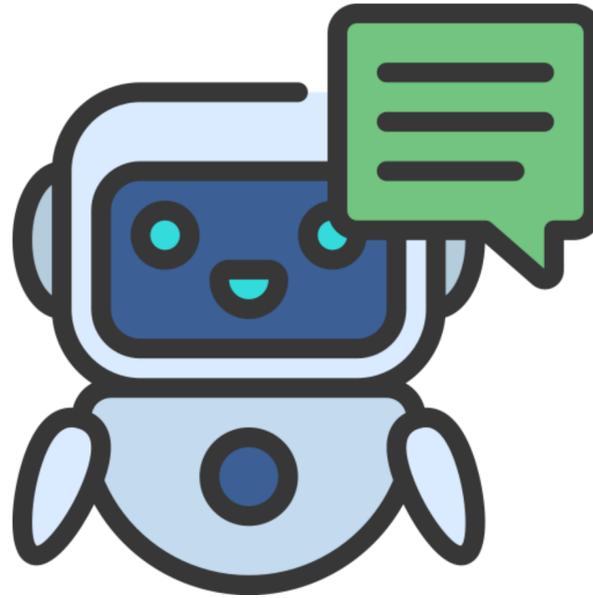
Game: Guess the model type

1. Predicting the next word in an unfinished sentence in any language, trained on nearly all digitized books and internet text.
2. An engine that classifies an MRI scan as malignant or benign based on millions of pre-labeled medical images.
3. A single model that can describe an image, translate the description into German, and generate a line of Python code based on the German text.
4. A program that identifies customer emails containing negative sentiment with over 98% accuracy, deployed after a week of training on a few thousand existing, scored emails.
5. A model that, without any new training, can answer questions about the plot of a newly released movie simply because its massive training data included the movie's script and reviews.
6. A pre-trained visual model that can recognize a dog in a photo, but is then **fine-tuned** by a company to specifically detect only their unique dog-food packaging in store shelf photos.

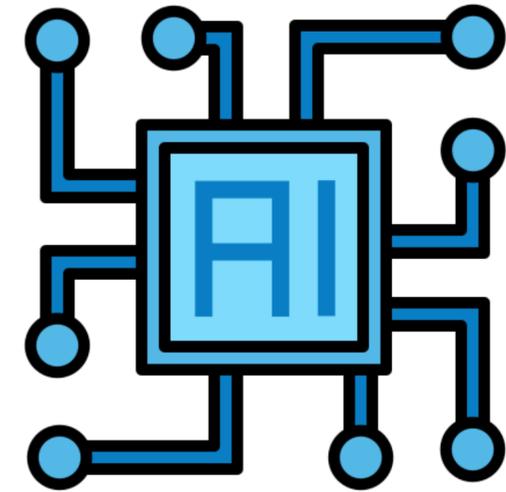
What is AI agent?



AI models are
brain behind
execution



AI assistants are
reactive,
performing tasks
at your request



AI agents are proactive,
working autonomously to
achieve a specific goal by
any means at their
disposal

An **AI agent** is a system that

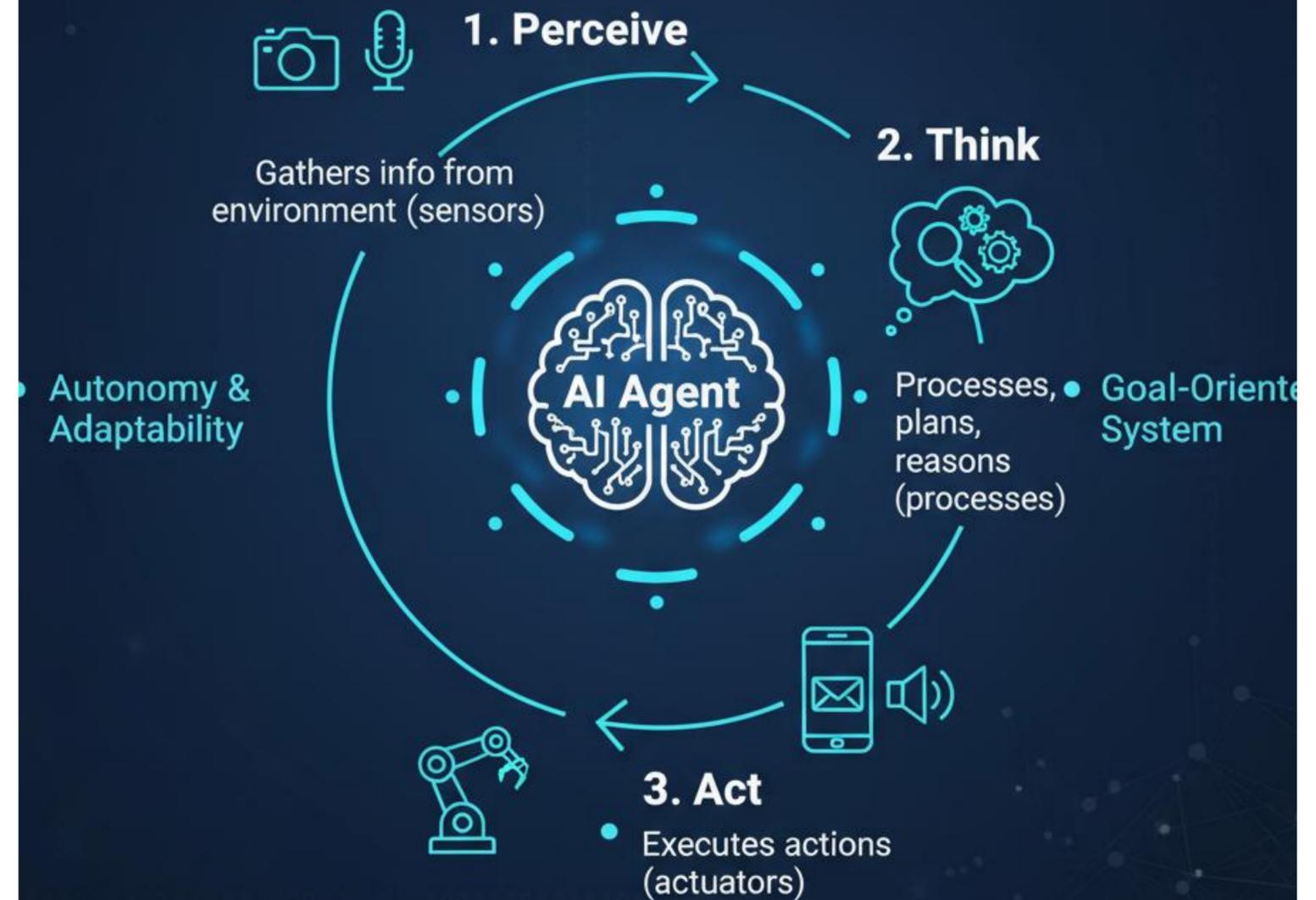
- **perceives** its environment (through sensors),
- takes **actions** (through actuators) to achieve **goals**,
- and
- exhibits **autonomy** and **adaptability**.

An artificial intelligence (AI) agent is a system that autonomously performs tasks by designing workflows with available tools.

It's more than just a model; it's a complete system designed to act **purposefully** and **independently**.

What is an AI Agent?

The "Perceive-Think-Act" Cycle



Key characteristics of AI agent

Action-oriented:

They can execute actions, use tools, and interact with other systems to achieve their goals.

Goal-oriented:

They are designed with specific objectives in mind, and they determine the best course of action to reach them.

Autonomous:

They can act independently to complete tasks based on their programming and goals, rather than requiring constant human input.

Perceptive:

They interact with and "perceive" their environment to gather data and inform their decisions.

Adaptive and learning:

They can improve their performance over time through self-learning and by reflecting on their responses.

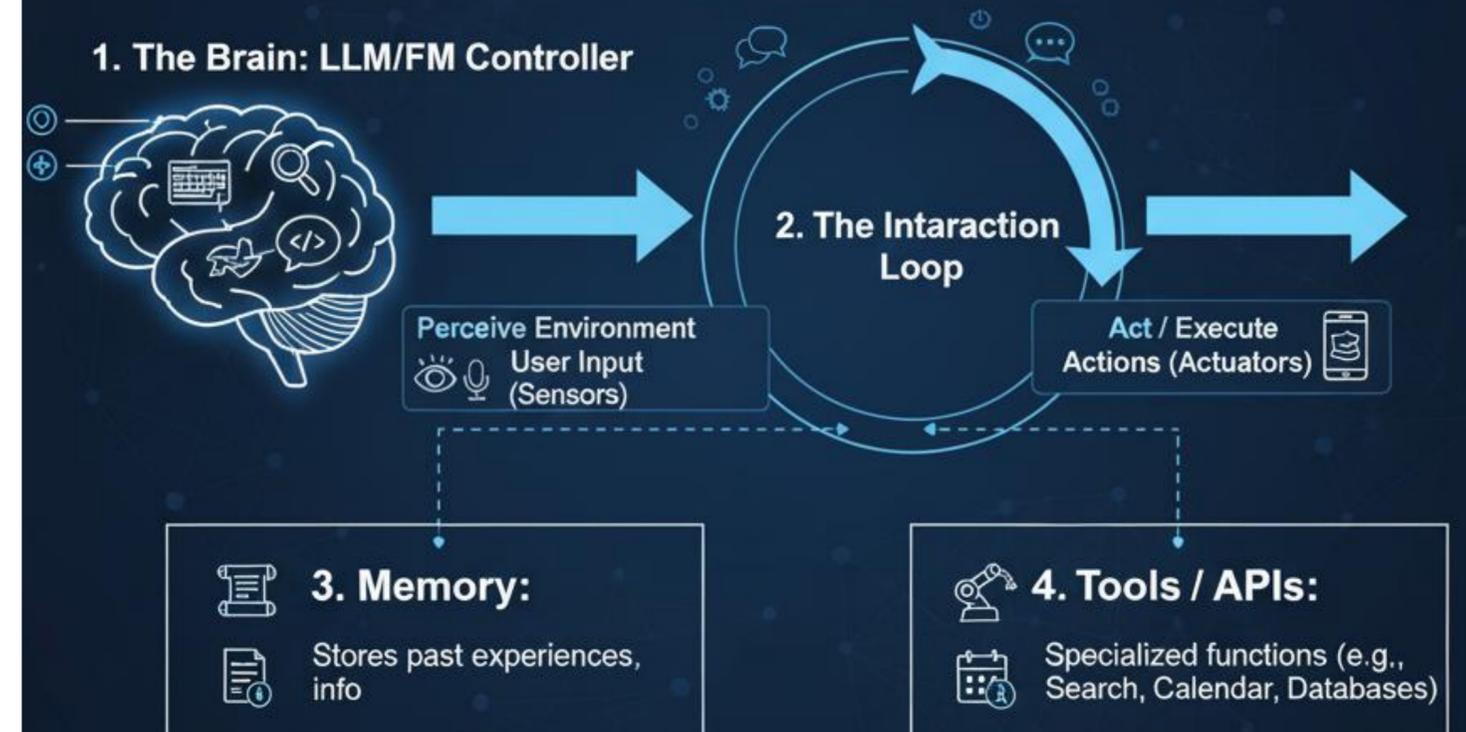
The Agent's Components

An effective AI agent typically combines these elements:

- 1.LLM/FM (The Controller):** Handles planning, reasoning, and conversational interface.
- 2.Memory:** Stores past interactions and information (short-term, long-term).
- 3.Tools (Traditional Models & APIs):** External functions the agent can call (e.g., a simple sentiment classifier, a search API, a calendar tool).

Components of an AI Agent

The “Brain” & The “Body” / Interaction Loop



Is AI Agent Turing complete?

An AI agent, particularly a sophisticated one, can be considered **Turing-complete in practice**, or at least capable of simulating a Turing machine.

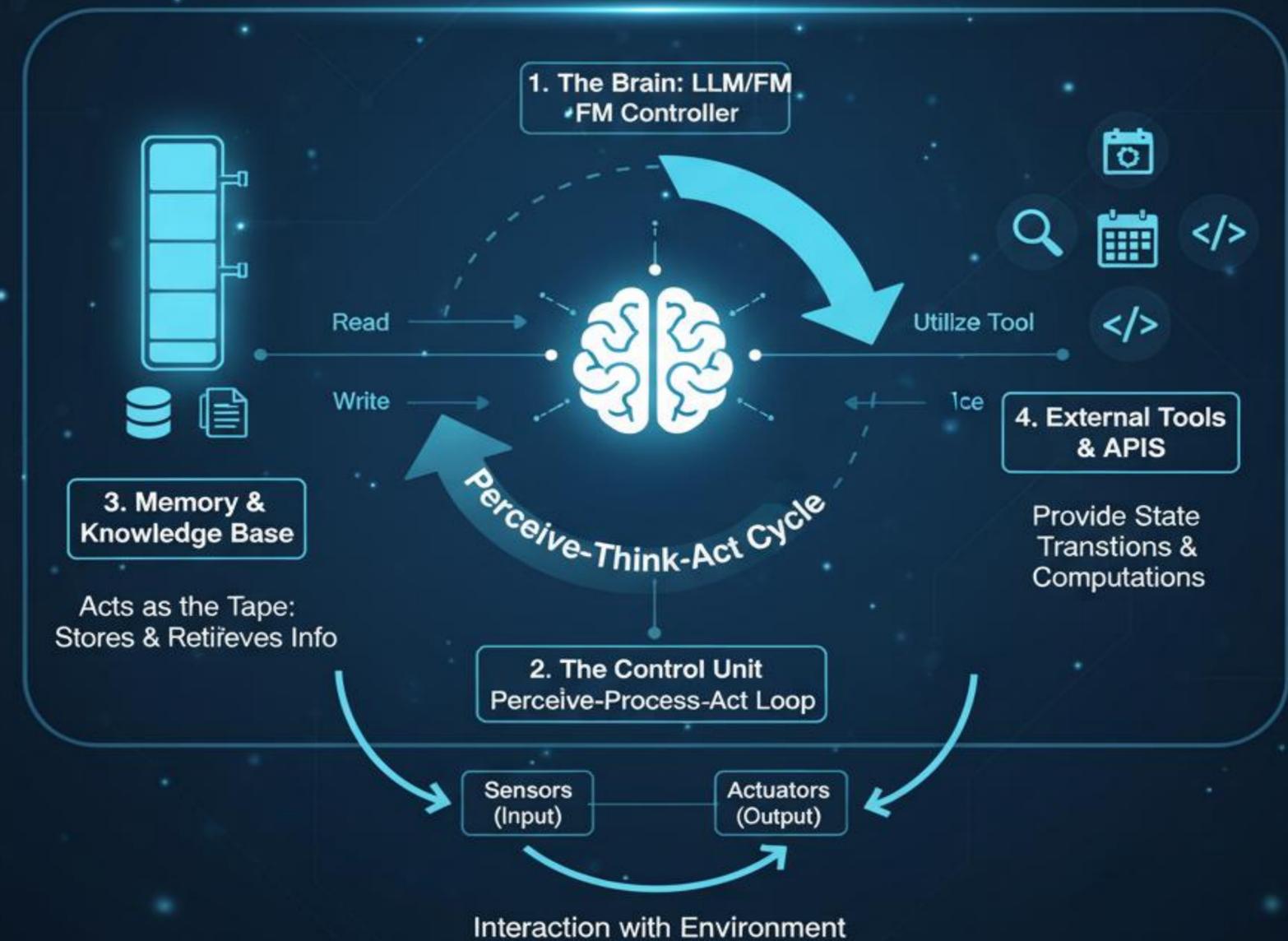
- **LLMs as Universal Function Approximators:** Large Language Models are essentially universal function approximators. With enough parameters and training data, they can theoretically learn to perform any computable function.

- **Access to Tools:** A key aspect of modern AI agents is their ability to use external tools (APIs, code interpreters, databases). These tools provide the "read/write tape" and the "state transitions" that a Turing machine needs. For example:

- **Code Interpreter:** An agent that can write and execute arbitrary code (Python, etc.) effectively has the power of a Turing machine.
- **Memory:** The agent's memory (short-term context, long-term vector databases) acts like a tape, storing and retrieving information.
- **Perception & Action:** The perception of new input and the execution of actions are analogous to reading from and writing to the tape, and changing the machine's state.

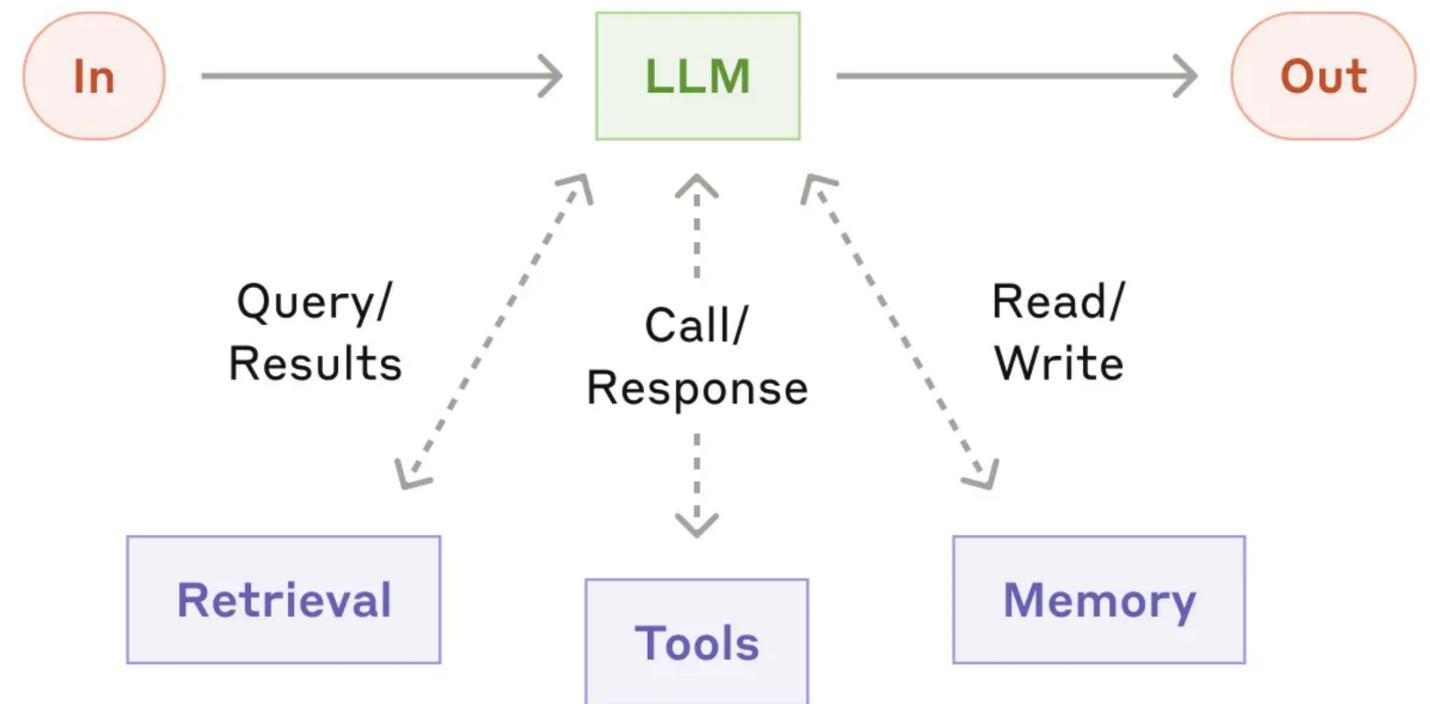
Is an AI Agent Turing Complete?

Simulating a Turing Machine



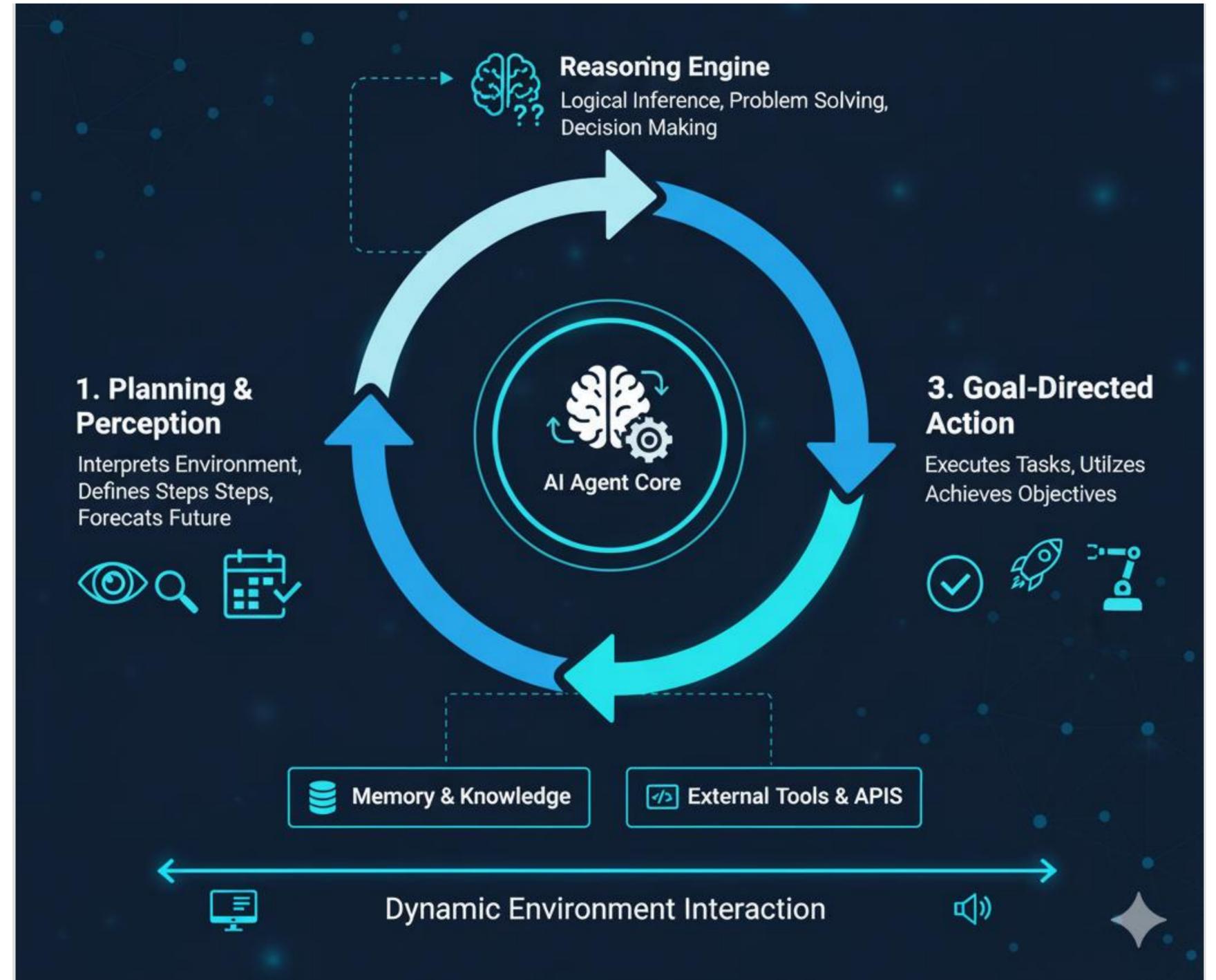
AI Agents: Advantages of AI Agents

1. Feedback Mechanisms via Critic Agents and Human-in-the-Loop to test and improve responses.
2. Digital Workforce Via Autonomous & Semi-Autonomous AI Systems to fully or partially automate business processes with an appropriate level of agency.
3. Proactive Instead of Reactive, Longer Horizon Tasks in Addition to Real-Time Interaction
Not just chat, but agents can initiate conversations and perform longer running tasks like report compilation or working entire cases.



Agentic AI solution

Planning
Reasoning
Action



Types of AI agents

Simple reflex agents

Designed to operate based on direct responses to environmental conditions.

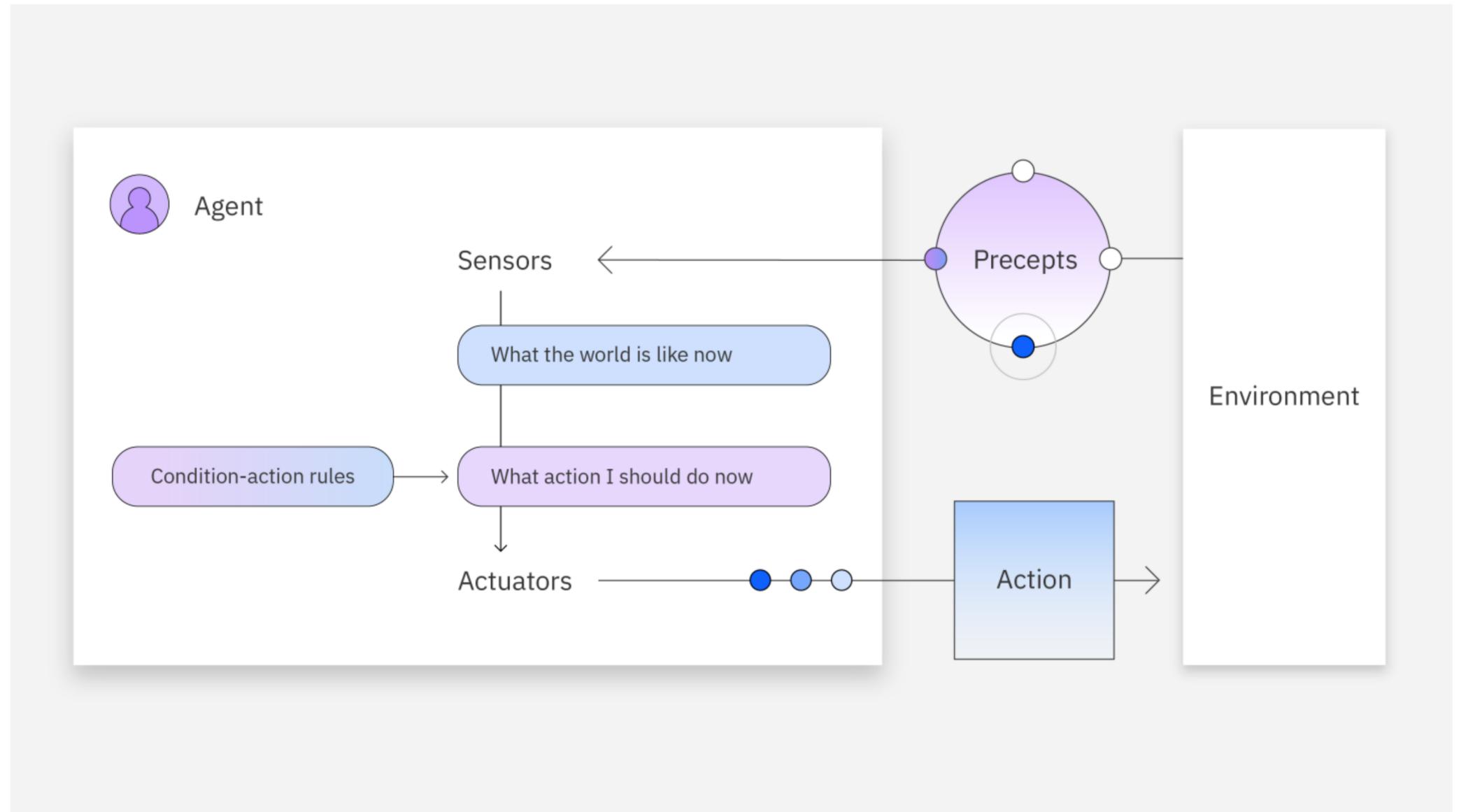
Follow predefined rules, known as condition-action rules, to make decisions without considering past experiences or future consequences.

Reflex agents apply current perceptions of the environment through sensors and take action based on a fixed set of rules.

For **example**, a thermostat is a simple reflex agent that turns on the heater if the temperature drops below a certain threshold and turns it off when the wanted temperature is reached

Simple reflex agents are effective **in structured and predictable** environments where the rules are well-defined. However, they **struggle** in dynamic or complex scenarios that require memory, learning or long-term planning.

Because they do not store past information, they can repeatedly **make the same mistakes** if the predefined rules are insufficient for handling new situations.



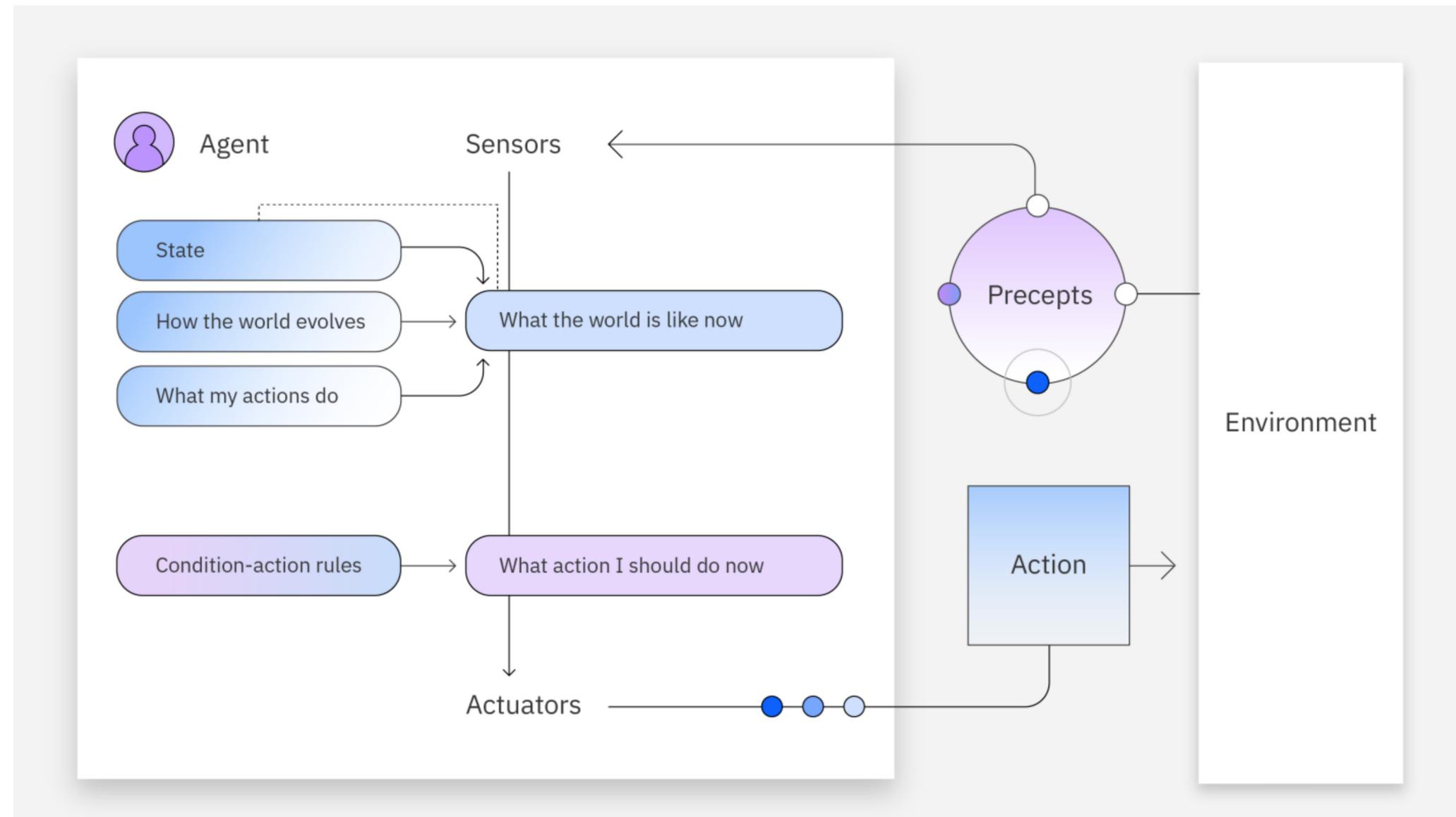
Types of AI agents

Model-based reflex agents

More advanced version of the simple reflex agent. While it still relies on condition-action rules to make decisions, it also **incorporates an internal model** of the world. This model helps the agent track the current state of the environment and understand how past interactions might have impacted it, allowing it to make more informed decisions. Model-based reflex agents use their internal model to **reason** about the environment's dynamics and make decisions accordingly.

For instance, a robot navigating a room might not just react to obstacles in its immediate path but also consider its previous movements and the locations of obstacles that it has already passed.

They can handle situations where the context needs to be **remembered** and used for future decisions, making them more adaptable than simpler agents. However, while model-based agents improve flexibility, they still lack the advanced reasoning or learning capabilities required for truly complex problems in dynamic environments.



Types of AI agents

Goal-based agents

Extends the capabilities of a simple reflex agent by incorporating a **proactive, goal-oriented** approach to problem-solving.

Goal-based agents consider their ultimate objectives and use **planning** and **reasoning** to choose actions that move them closer to achieving their goals.

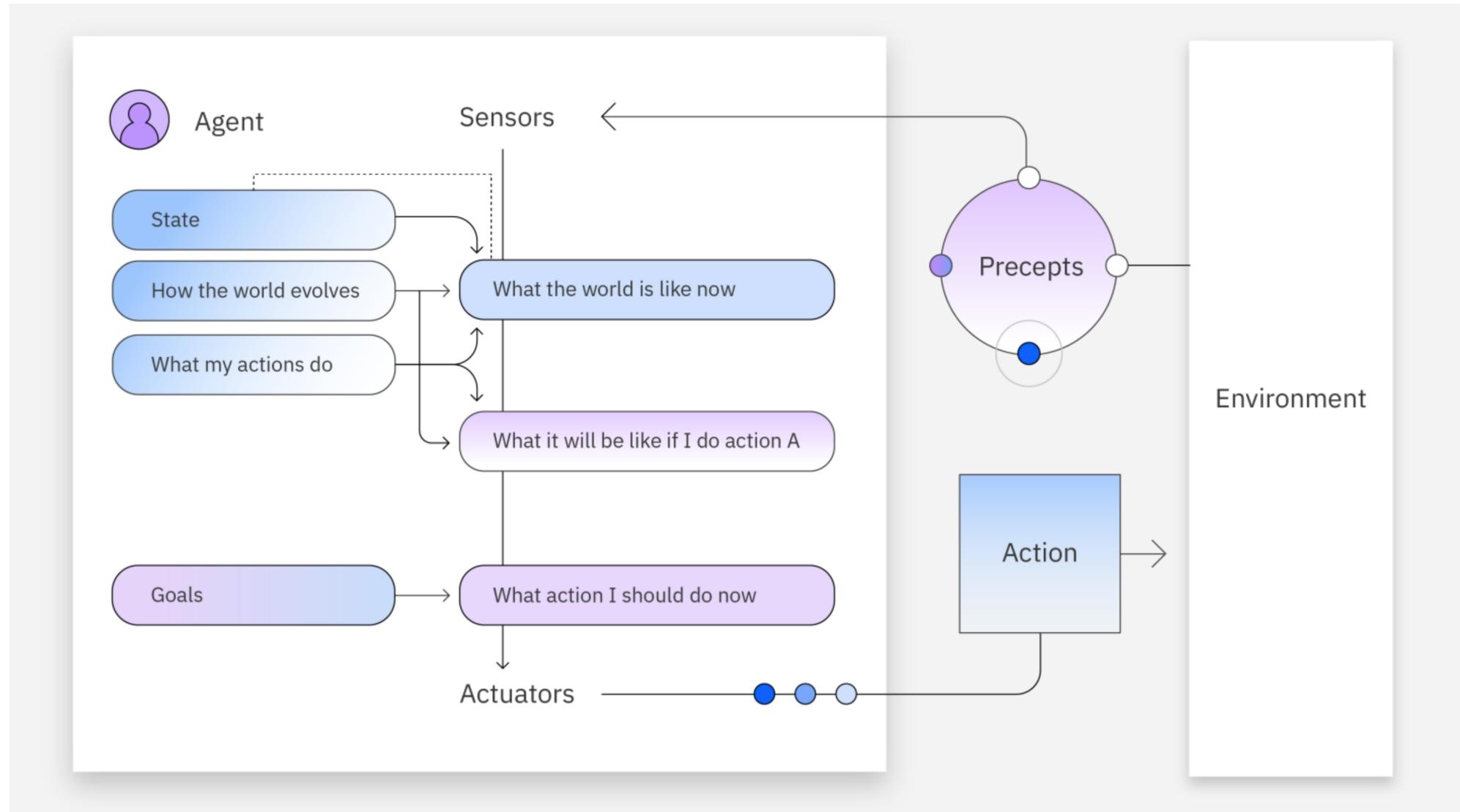
These agents operate by setting a specific goal, which guides their actions. They **evaluate** different possible actions and select the one most likely to help them reach that goal.

For instance, a robot designed to navigate a building might have a goal of reaching a specific room. Rather than reacting to immediate obstacles only, it plans a path that minimizes detours and avoids known obstacles, based on a logical assessment of available choices.

It **considers future states** and their potential impact on reaching the goal.

They often **rely on preprogrammed strategies** or decision trees for evaluating goals.

Goal-based reflex agents are widely used in **robotics**, **autonomous vehicles** and **complex simulation systems** where reaching a clear objective is crucial.



Types of AI agents

Utility-based agents

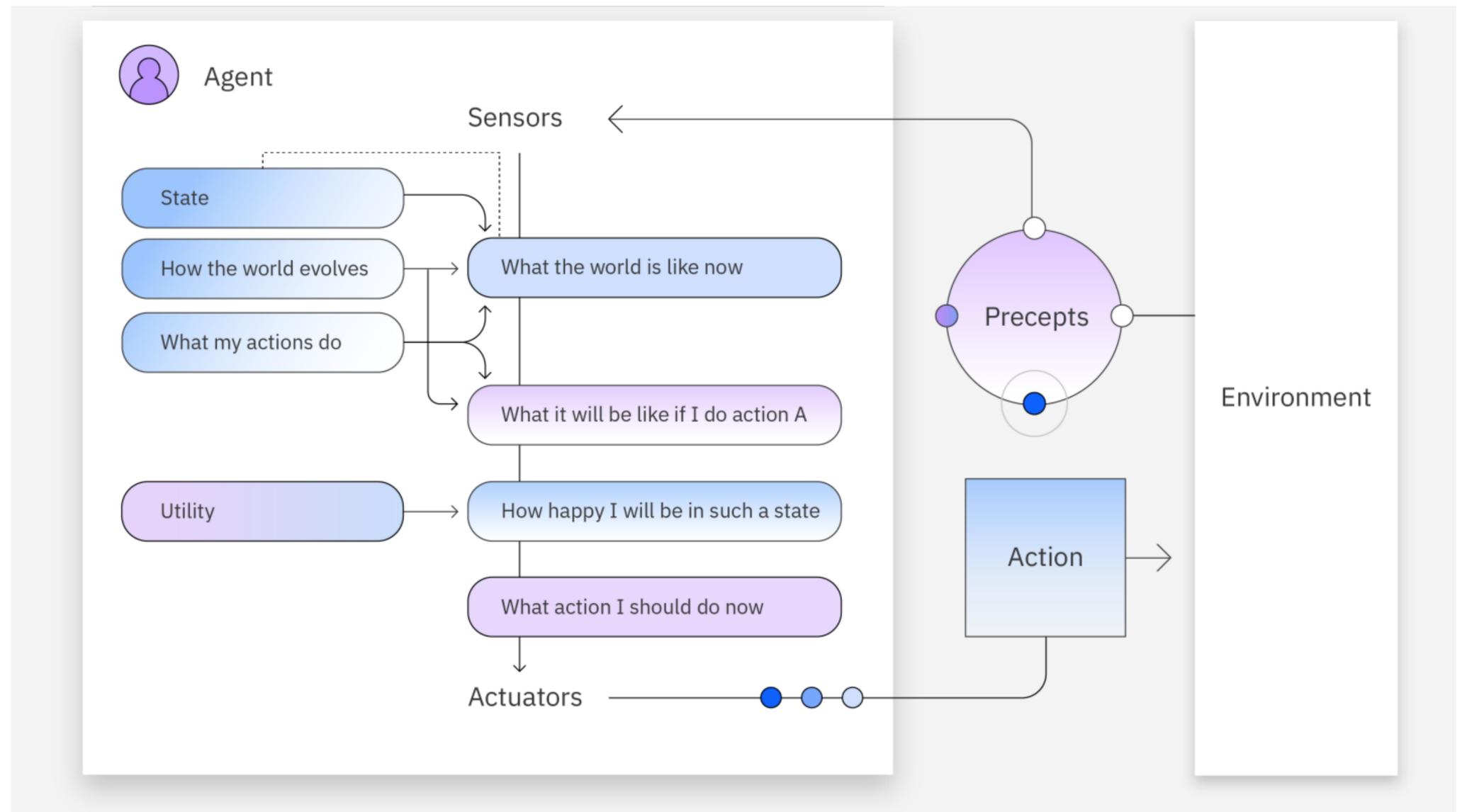
A utility-based reflex agent goes beyond simple goal achievement by using a **utility function** to evaluate and select actions that maximize overall benefit.

Utility-based agents consider a range of possible outcomes and assign a utility value to each, helping them **determine** the most optimal course of action. Particularly effective in situations where multiple goals or tradeoffs are involved.

For example an e-commerce company might employ a utility-based agent to optimize pricing and recommend products. The agent evaluates various options, such as sales history, customer preferences and inventory levels to **make informed decisions** on how to price items dynamically.

Utility-based reflex agents are effective in **dynamic** and **complex** environments, where simple binary goal-based decisions might not be sufficient.

They **help balance** competing objectives and adapt to changing conditions, ensuring more intelligent, flexible behavior.



Types of AI agents

Learning agents

A learning agent improves its performance over time by adapting to new experiences and data.

Learning agents continuously update their **behavior** based on **feedback** from the environment. This allows them to enhance their decision-making abilities and perform better in dynamic and uncertain situations.

Learning agents typically consist of 4 main components:

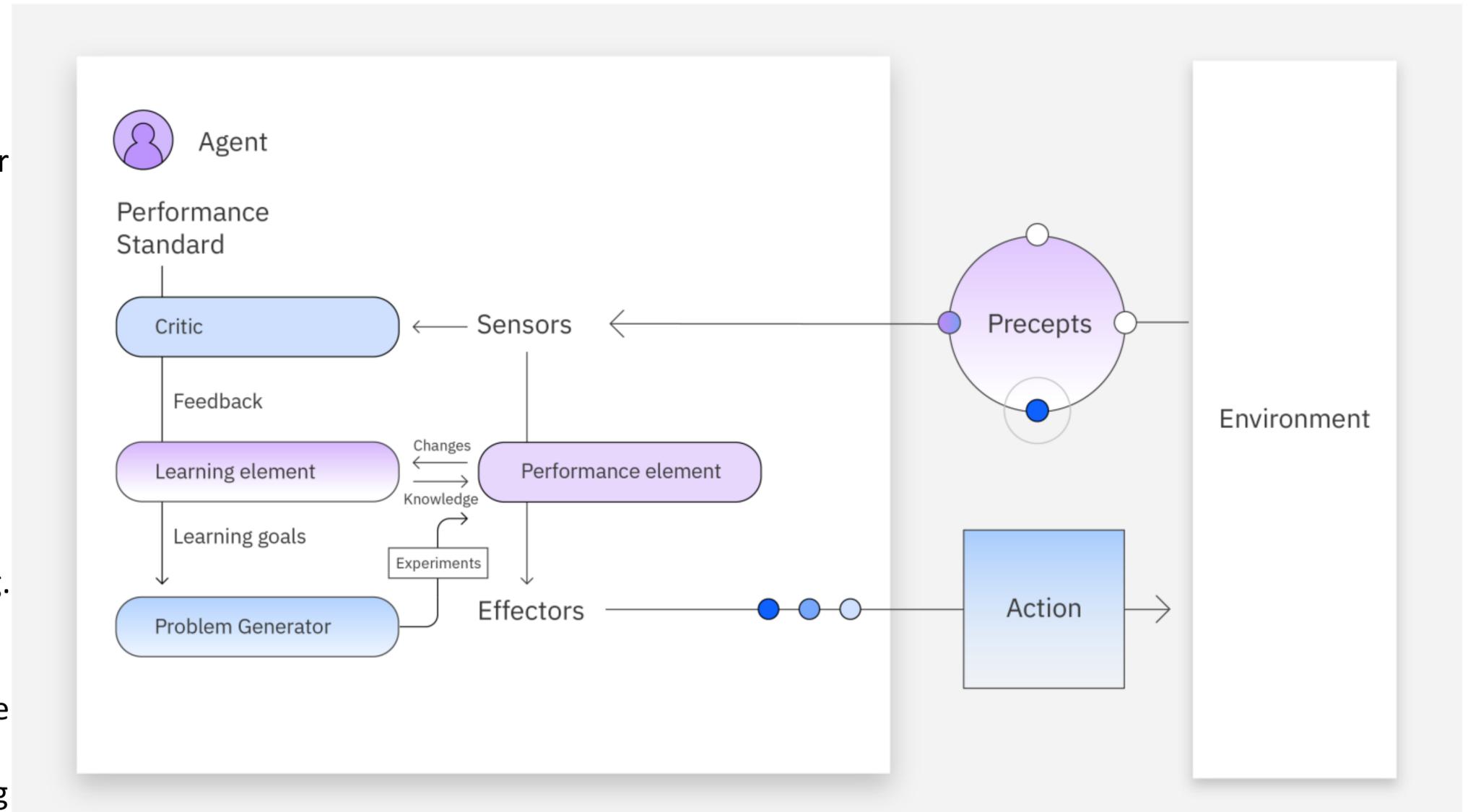
Performance element: Makes decisions based on a knowledge base.

Learning element: Adjusts and improves the agent's knowledge based on feedback and experience.

Critic: Evaluates the agent's actions and provides feedback, often in the form of rewards or penalties.

Problem generator: Suggests exploratory actions to help the agent discover new strategies and improve its learning. For example an agent might explore different strategies, receiving rewards for correct actions and penalties for incorrect ones. Over time, it learns which actions maximize its reward and refine its approach.

Learning agents are **highly flexible** and capable of handling complex, ever-changing environments.



Requirements dictate design,
and the Reference Architecture
provides the blueprint.

A Reference Architecture reduces the
complexity, cost, and risk of solution
deployment and sets the foundation for
business transformation.

Agentic AI Architectures

Single-agent architectures

Features a single autonomous entity making centralized decisions within an environment. A single AI agent operates independently to perceive its environment, make decisions and take actions to achieve a goal.

Key features

Autonomy: The agent operates independently without requiring interaction with other agents.

Strengths

Simplicity: Easier to design, develop and deploy compared to multi-agent systems. Requires fewer resources because it does not need to manage multiple agents or communication protocols.

Predictability: Easier to debug and monitor because the agent operates independently.

Speed: No need for negotiation or consensus-building among multiple agents.

Cost: Less expensive to maintain and update compared to complex multi-agent architectures. Fewer integration challenges when deployed in enterprise applications.

Weaknesses

Limited scalability: A single agent can become a bottleneck when handling high-volume or complex tasks.

Rigidity: Struggles with tasks that require multistep workflows or coordination across

Multi-agent architectures

Go beyond the AI capabilities of traditional, single-agent setups.

Agents adapt their roles based on evolving tasks, helping to ensure flexibility and responsiveness in dynamic scenarios.

Multi-agent systems are more flexible.

Key features

Hierarchy: Roles are clearly defined.

Centralized communication: Agents report to the leader.

Strengths

Task efficiency: Ideal for sequential workflows.

Clear accountability: Leader aligns objective.

Weaknesses

Bottlenecks: Leader reliance can slow progress.

Single point of failure: Vulnerable to leader issues.

Best use cases

Workflow automation: Multistep approvals.

Document generation: Sections overseen by a leader.

Horizontal AI architectures

Structure

Peer collaboration model: Agents work as equals in a decentralized system, collaborating freely to solve tasks.⁶

Key features

Reactive architectures

Reactive architectures map situations directly to actions. They are reflexive, making decisions based on immediate stimuli from the environment rather than drawing on memory or predictive capabilities. They can't learn from the past or plan for the future.

Cognitive architectures

Cognitive architectures

An advanced AI system that mimics human-like thinking, reasoning, learning and decision-making.

These agents incorporate elements of perception, memory, reasoning and adaptation, each represented by individual modules, enabling them to operate in complex, uncertain environments while improving over time. This is the most advanced type of agentic architecture.

Deliberative architectures

A deliberative architecture is an AI system that makes decisions based on reasoning, planning and internal models of the world. Unlike reactive agents, deliberative agents analyze their environment, predict future outcomes and make informed choices before acting.

BDI architecture

This architecture models human-like reasoning in a BDI agent, with:

Beliefs (B): The agent's knowledge of the world

Desires (D): The agent's goals or objectives

Intentions (I): The course of action that the agent commits to in order to achieve its desires

Single agent vs Multi agent trade-offs

Single agent

Pros:

1. Lower and more predictable runtime cost.
2. Lower latency for time critical systems.
3. Low implementation complexity.

Cons:

1. Zero shot accuracy/quality limited by performance of selected model.
2. Less flexible for implementation of complex workflows.

Multi agent

Cons:

1. Increasing and less predictable runtime cost since the number of LLM invocations increase and is not fully deterministic.
2. Increasing latency especially for parts of workflows that cannot be parallelized due to sequencing of LLM calls.
3. Increasing implementation complexity where it is often harder to anticipate and evaluate the impact of changes to the solution.

Pros:

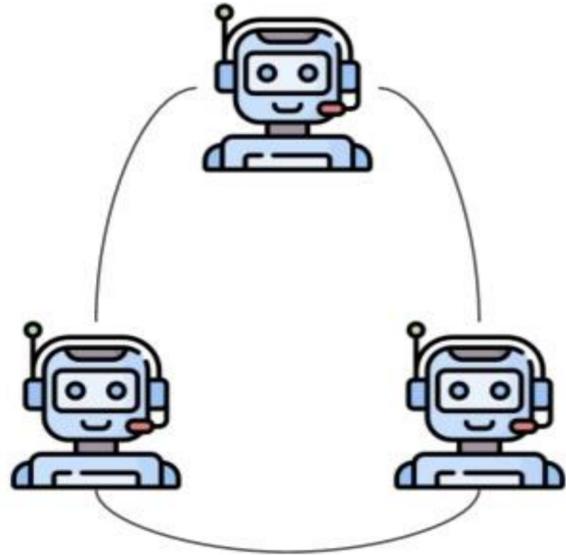
1. Significant empirical evidence that the accuracy/quality of multi agent systems outperforms single agent zero shot solutions.
2. Scales to automation of sophisticated workflows with state memory and human in the loop to support complex business processes.



Start simple and increase complexity when needed.

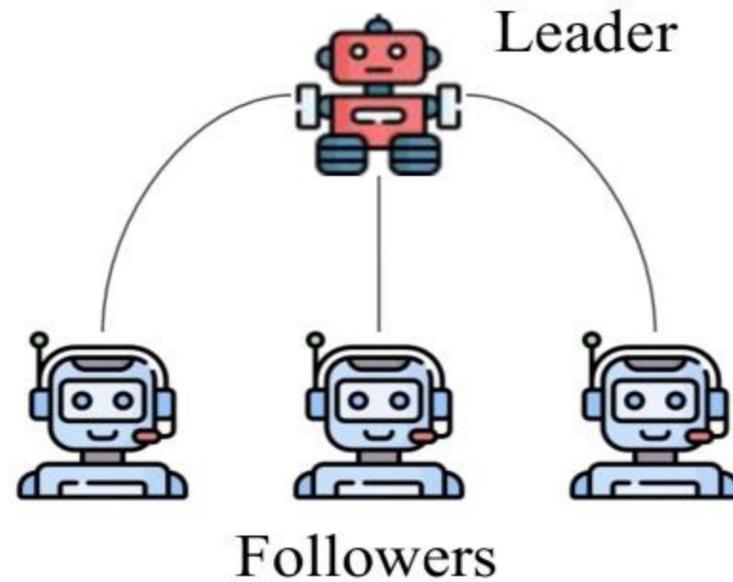


Structure of Multi-agent Systems



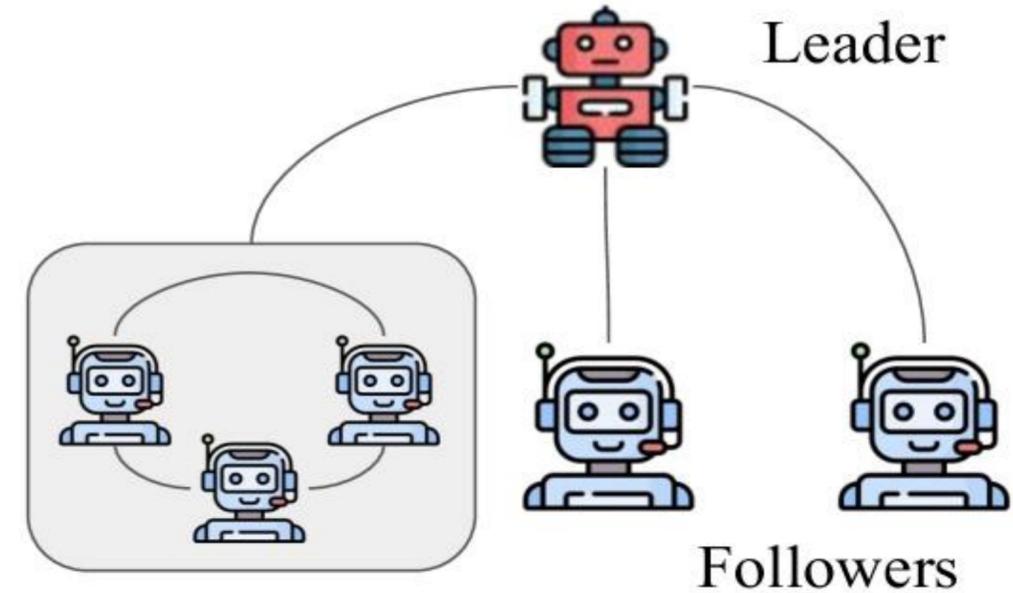
a. Equi-level structure

- Agents operate at the same hierarchical level.
- Collaborate or negotiate without a centralized leader.
- Shared responsibilities and collective decision-making.



b. Hierarchical structure

- Consists of leaders and followers.
- Leaders guide and plan; followers execute instructions.
- Used in scenarios requiring coordinated efforts.

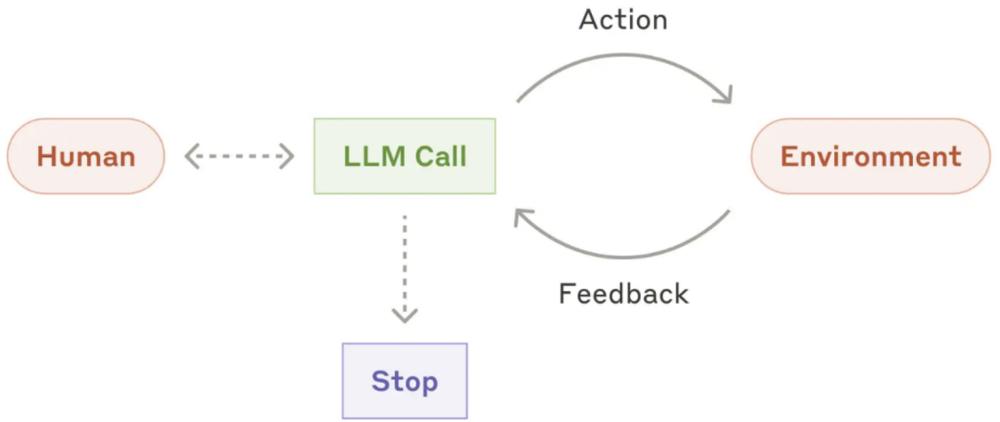


c. Nested structure

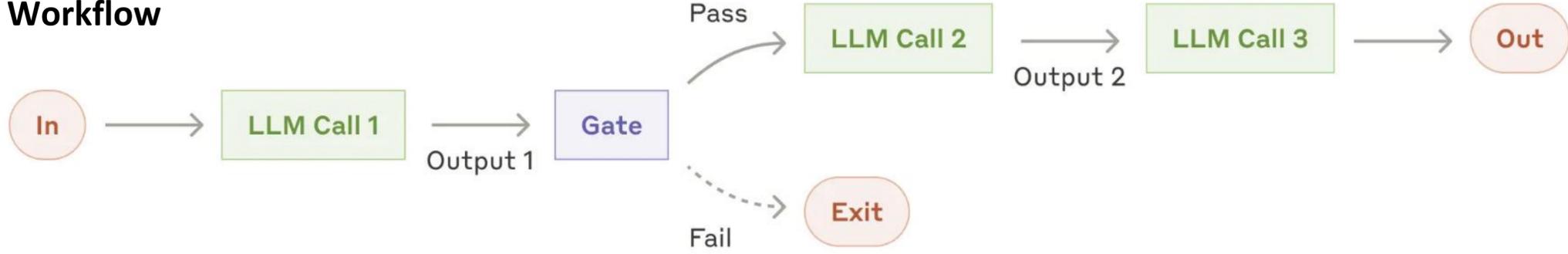
- Combines equi-level and hierarchical structures.
- Complex tasks broken into sub-tasks managed by sub-systems.
- Interplay between hierarchy and peer-to-peer interaction.

Multiagent patterns

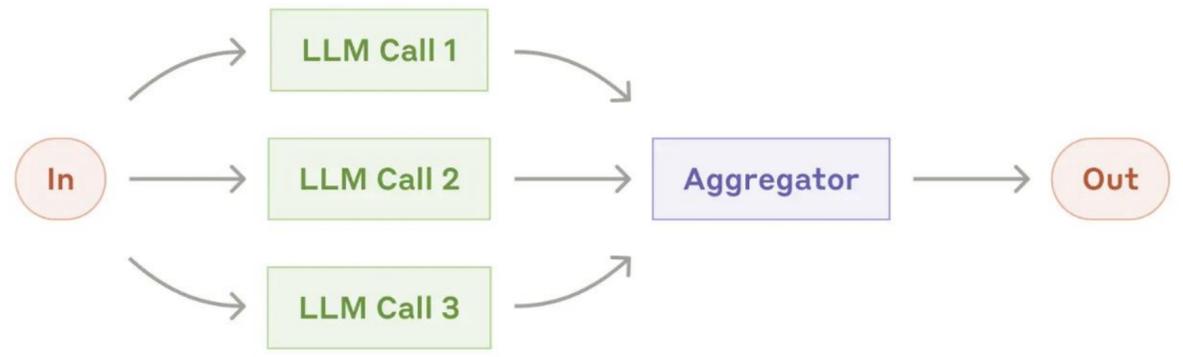
Agents



Workflow



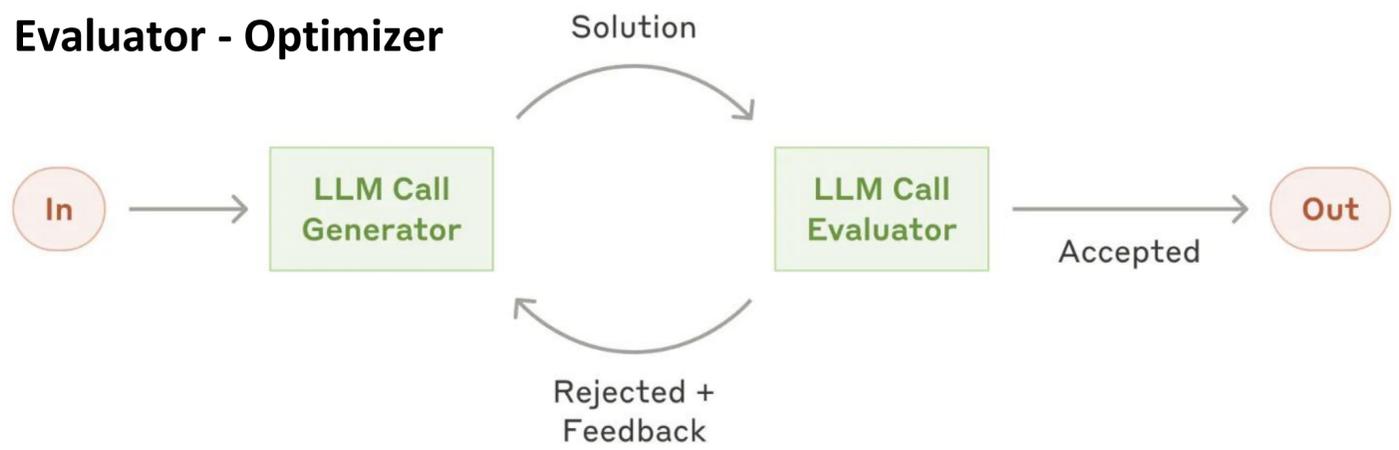
Parallelization



Orchestrator



Evaluator - Optimizer



AI Agent protocols

MCP – Model Context protocol

Function: Connecting Agents to the Outside World (Tools & Data)

Definition: An open standard (developed by Anthropic) for an AI system (**Client**) to access external data sources and tools (**Servers**).

Key Concept: Standardizes how an agent can "**see**" and "**use**" enterprise data, code, or APIs securely and reliably.

Impact: Replaces fragmented API connectors with a single, universal protocol for tool use, providing **real-time context** for better agent decision-making.

A2A – Agent-to-Agent protocol

Function: Agents Talking to Agents (Interoperability & Teamwork)

Definition: An open protocol (developed by Google) enabling autonomous AI agents, built on diverse frameworks and platforms, to **communicate and coordinate** tasks.

Key Concept: Allows agents to collaborate on long-running, complex workflows without needing to expose their internal memory or code.

Impact: Creates a **multi-agent ecosystem** where specialized agents can be composed into powerful "crews" to solve problems that no single agent could handle (e.g., a "Researcher Agent" delegating a query to a "Data Analyst Agent").

Challenges in Agentic AI Design

Coordination & Collaboration 🤝

Challenge: Ensuring agents communicate effectively, understand shared goals, and efficiently divide tasks. Poor coordination leads to duplication of work or, worse, task failure.

Risk: Inefficient resource use and delays.

Conflict Resolution ✨

Challenge: Agents with competing objectives (or different interpretations of the same objective) may actively work against each other.

Risk: Deadlocks, system instability, and unintended adversarial behavior.

Emergent Behavior 💡

Challenge: Unexpected, complex behaviors that arise from the interaction of many simple agent rules. The system as a whole behaves in a way that wasn't explicitly programmed.

Risk: Unpredictability, making debugging and verification extremely difficult.

Ethical Considerations ⚖️

Challenge: Agents may exhibit bias, make unfair decisions, or prioritize efficiency over ethical compliance when operating in real-world systems (e.g., hiring, lending).

Mitigation Need: Clear ethical guidelines and "**value alignment**" in the training and planning phases.

Safety & Control 🛑

Challenge: The autonomous nature of agents means they can execute unintended actions at high speed and scale. This includes resource consumption, infinite loops, and irreversible real-world actions.

Mitigation Need: Robust **Guardrails, Human-in-the-Loop (HIL)**, and **protocol-based limits**

Transparency & Explainability (XAI) ❓

Challenge: Understanding *why* an agent made a particular decision, especially when that decision involved complex reasoning or collaboration with other agents.

Risk: Difficulty in auditing failures and maintaining trust in the system.

Lets get our hands dirty

Frameworks overview

Open Source Frameworks

- Microsoft Frameworks
 - Enterprise framework Semantic Kernel
 - Research framework Autogen, Magentic-One (based on Autogen)
- CAMEL
- **CrewAI**
- LangGraph
- LlamaIndex Agents & Workflows plus llama_deploy (was llama_agents)
- Mosaic AI Agent Framework and Mosaic AI Agent Evaluation
- Spring AI Agentic Patterns
- phidata

Commercial Platforms

- Amazon Bedrock Agents
- Celonis AgentC – integrated with Microsoft Copilot Studio, watsonx Orchestrate, Amazon Bedrock Agents, CrewAI
- Google Vertex AI Agents
- Microsoft Copilot Studio (was Power Virtual Agents)
- OCI Generative AI Agents
- Salesforce Agentforce 2.0 (was Einstein Copilot)
- SAP AI Agents in Joule
- ServiceNow Agents
- watsonx Orchestrate



<https://gofastmcp.com/getting-started/welcome>

FastMCP handles all the complex protocol details so you can focus on building.

In most cases, decorating a Python function is all you need — FastMCP handles the rest.

 **Fast:** High-level interface means less code and faster development

 **Simple:** Build MCP servers with minimal boilerplate

 **Pythonic:** Feels natural to Python developers

 **Complete:** Everything for production — enterprise auth (Google, GitHub, Azure, Auth0, WorkOS), deployment tools, testing frameworks, client libraries, and more

Core components:

 **Tools:** Tools are the core building blocks that allow your LLM to interact with external systems, execute code, and access data that isn't in its training data. In FastMCP, tools are Python functions exposed to LLMs through the MCP protocol.

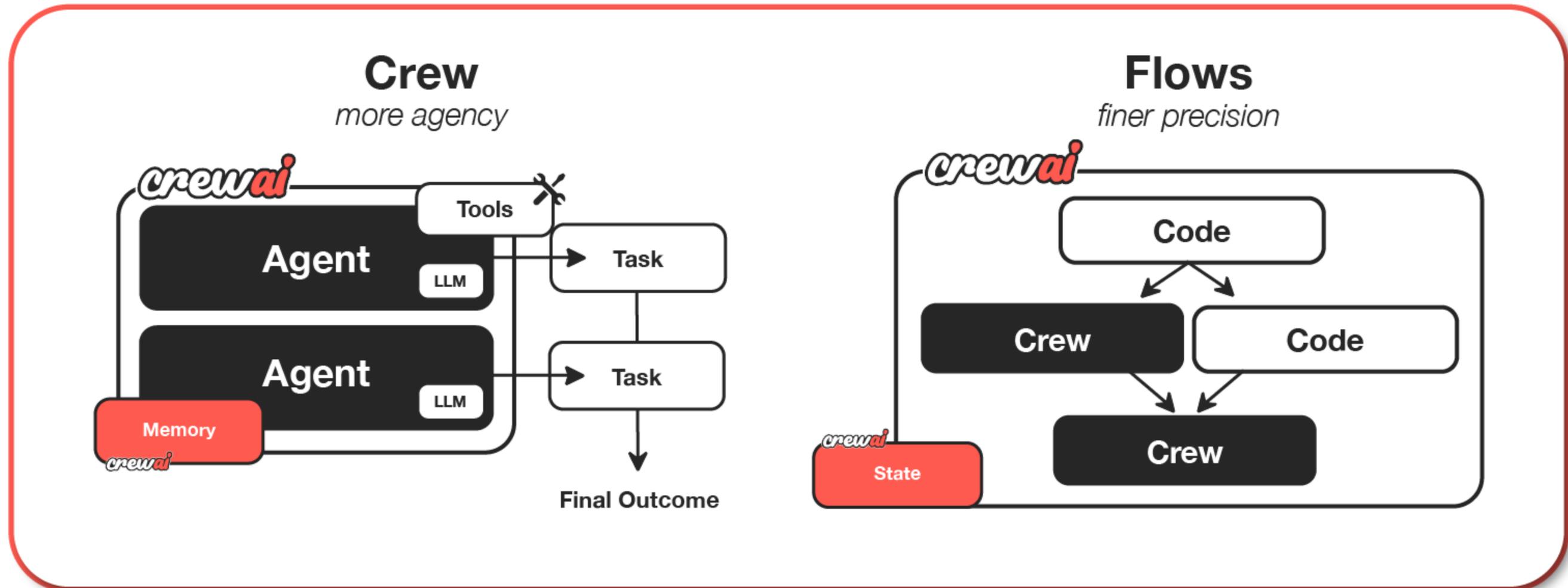
 **Resources:** Resources represent data or files that an MCP client can read, and resource templates extend this concept by allowing clients to request dynamically generated resources based on parameters passed in the URI.

 **Prompts:** Prompts are reusable message templates that help LLMs generate structured, purposeful responses.



<https://github.com/crewAIInc/crewAI>

Fast and Flexible Multi-Agent Automation Framework



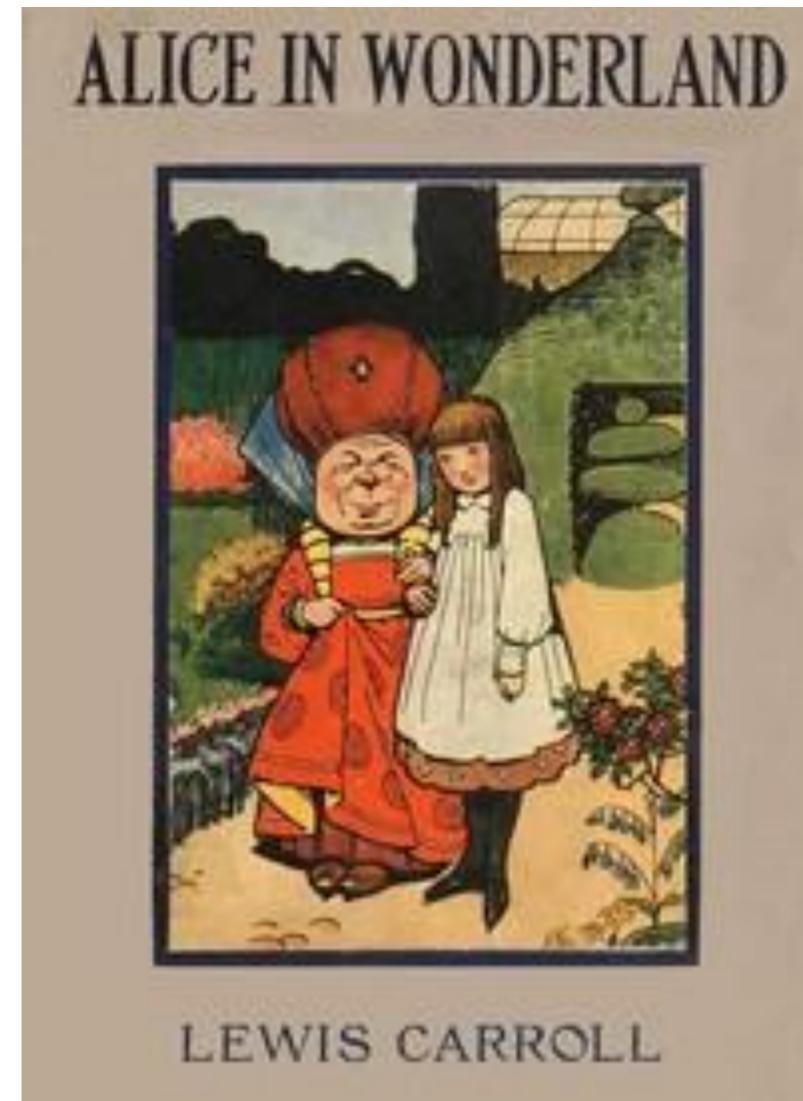


<https://www.gutenberg.org>

<https://github.com/palkokec/gutread>

Project Gutenberg is a library of over 75,000 free eBooks

Choose among free epub and Kindle eBooks, download them or read them online. You will find the world's great literature here, with focus on older works for which U.S. copyright has expired. Thousands of volunteers digitized and diligently proofread the eBooks, for you to enjoy.



Tool Args:

 Crew: crew

 Task: **get_schema** (ID: 59d34b97-f277-496a-91f4-c9e8b182b3f8)
Status: Executing Task...

 Agent Started

Agent: **Guttenberg advisor**

Task: get the database schema using the get_schema tool

 Crew: crew

 Task: **get_schema** (ID: 59d34b97-f277-496a-91f4-c9e8b182b3f8)

 Crew: crew

 Task: **get_schema** (ID: 59d34b97-f277-496a-91f4-c9e8b182b3f8)
Status: Executing Task...

 Used get_schema (1)
 Thinking...

 Agent Tool Execution

Agent: **Guttenberg advisor**

Thought: Action: get_schema

Using Tool: **get_schema**

Tool Output

```
--
-- PostgreSQL database dump
--

\restrict bLYxXJa28dfKyAHMmIdz2e0Svd1Z5hVvMdwL1zAlcXccwBBq8zHvX5TtjKwaCW3

-- Dumped from database version 15.14 (Debian 15.14-0+deb12u1)
-- Dumped by pg_dump version 15.14 (Debian 15.14-0+deb12u1)

SET statement_timeout = 0;
SET lock_timeout = 0;
SET idle_in_transaction_session_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SELECT pg_catalog.set_config('search_path', '', false);
SET check_function_bodies = false;
SET xmloption = content;
SET client_min_messages = warning;
SET row_security = off;

SET default_tablespace = '';

SET default_table_access_method = heap;

--
```

Agent: Guttenberg advisor

Tool Args:

 Crew: crew

📁 Task: **get_schema** (ID: 59d34b97-f277-496a-91f4-c9e8b182b3f8)

Assigned to: Guttenberg advisor

Status:  **Completed**

└─  Used get_schema (1)

📁 Task: **sql_search** (ID: 923affef-7a7e-4ef4-8b3b-c65d4e54a6fe)

Status: Executing Task...

└─  Thinking...

 Agent Started

Agent: **Guttenberg advisor**

Task:

Use the database schema from task 1 to construct a SQL query that answers this question:

'Please provide me all the books by author Moody'.

Use like clause and case insensitive string comparison rather than strict equal in where clause in sql statement.

Then, execute the query using the 'sql_search' tool.

Agent: **Guttenberg advisor**

Thought: Action: `sql_search`

Using Tool: **`sql_search`**

Tool Input

```
{  
  "query": "SELECT T2.title FROM author AS T1 INNER JOIN author_ebook AS T3 ON T1.id = T3.author_id INNER JOIN ebook  
AS T2 ON T3.ebook_id = T2.id WHERE LOWER(T1.name) LIKE LOWER('%Moody%');"  
}
```

Tool Output

```
[[["Moody's Anecdotes And Illustrations\r\nRelated in his Revival Work by the Great Evangelist"],["That Gospel Sermon  
on the Blessed Hope"],["The Way to God and How to Find It"],["Sovereign Grace: Its Source, Its Nature and Its  
Effects"],["Men of the Bible"],["Sowing and Reaping"],["To The Work! To The Work! Exhortations to Christians"],["The  
Overcoming Life, and Other Sermons"],["Moody's Stories: Being a Second Volume of Anecdotes, Incidents, and  
Illustrations"],["Weighed and Wanting: Addresses on the Ten Commandments"],["Secret Power; or, The Secret of Success  
in Christian Life and Work"],["Wondrous Love, and other Gospel addresses"],["Pleasure & Profit in Bible  
Study"],["Bible Characters"],["Prevailing Prayer: What Hinders It?"],["Junius Unmasked\r\nOr, Thomas Paine the author  
of the Letters of Junius and the Declaration of Independence"],["Historical records of the Buffs, East Kent Regiment  
(3rd Foot) : $b Formerly designated the Holland Regiment and Prince George of Denmark's Regiment"],["The Railroad  
Builders: A Chronicle of the Welding of the States"],["Gloucester Moors and Other Poems"],["The Faith Healer: A Play  
in Three Acts"],["Backwoods Surgery & Medicine"],["Edaville Railroad : $b The Cranberry Belt"],["Index to Library  
Reports"],["Dwight L. Moody: Hänen elämänsä ja vaikutuksensa"]],["title"]]
```

Agent: **Guttenberg advisor**

Task:

From the given array of results construct nice list of author, book name and brief summary description maximum 150 charatcers.

 Crew: crew

 Task: **get_schema** (ID: 59d34b97-f277-496a-91f4-c9e8b182b3f8)

Assigned to: Guttenberg advisor

Status:  **Completed**

└─  Used get_schema (1)

 Task: **sql_search** (ID: 923affef-7a7e-4ef4-8b3b-c65d4e54a6fe)

Assigned to: Guttenberg advisor

Status:  **Completed**

└─  Used sql_search (1)

 Task: **format_answer** (ID: 910f399d-146b-4599-8c1c-e02db63b3563)

Assigned to: Guttenberg advisor

Status:  **Completed**

 Agent Final Answer

Agent: **Guttenberg advisor**

Final Answer:

* ** ** ** **

Name: crew

ID: 5ee4e7ca-fb99-4f61-ae1-1888e2c9889d

Tool Args:

Final Output: * **Author**: D.L. Moody

Book Name: Moody's Anecdotes And Illustrations

Summary: Related in his Revival Work by the Great Evangelist.

* **Author**: Unknown

Book Name: That Gospel Sermon on the Blessed Hope

Summary: A gospel sermon discussing the blessed hope.

* **Author**: Unknown

Book Name: The Way to God and How to Find It

Summary: A guide on finding the way to God.

* **Author**: Unknown

Book Name: Sovereign Grace

Summary: Its Source, Its Nature and Its Effects.

* **Author**: Unknown

Book Name: Men of the Bible

Summary: A study of various men from the Bible.

* **Author**: Unknown

Book Name: Sowing and Reaping

Summary: Discusses the principles of sowing and reaping.

* **Author**: Unknown

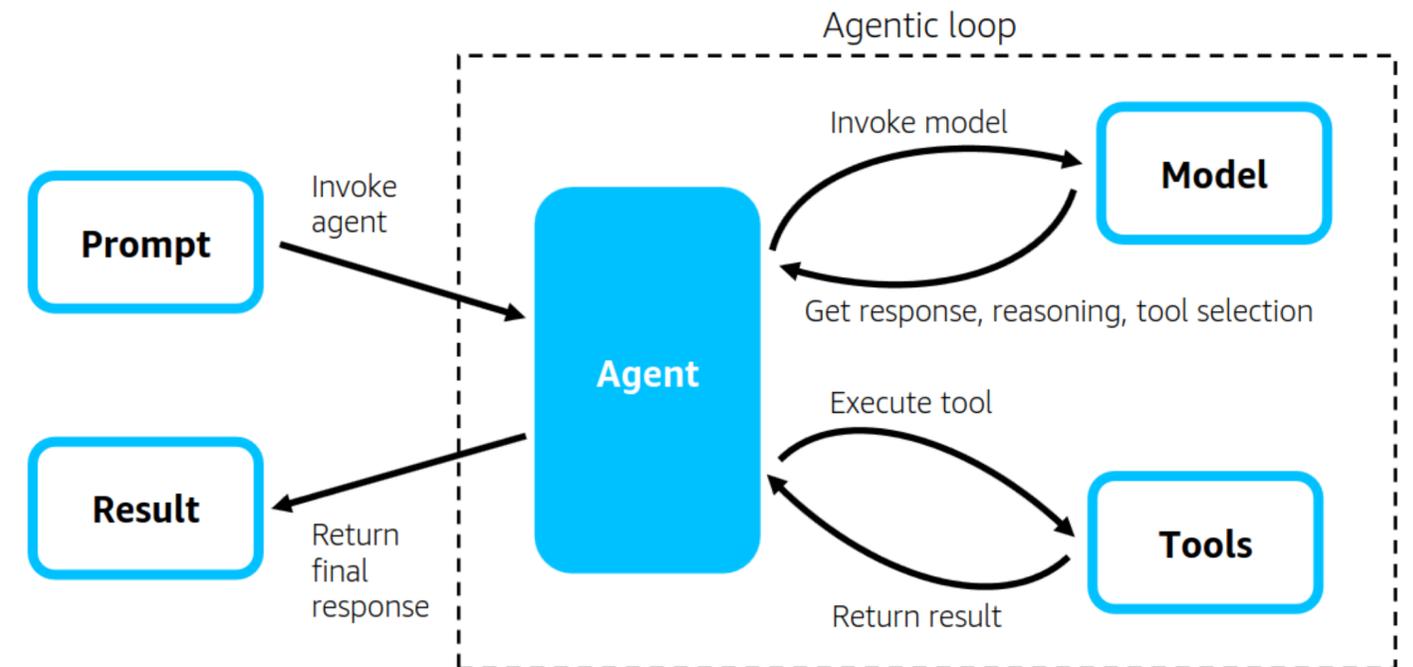


<https://strandsagents.com/latest/>

[Strands Agents](#) is a simple-to-use, code-first framework for building agents.

Strands Agents is an open source SDK that takes a model-driven approach to building and running AI agents in just a few lines of code. Strands scales from simple to complex agent use cases, and from local development to deployment in production.

Strands simplifies agent development by embracing the capabilities of state-of-the-art models to plan, chain thoughts, call tools, and reflect



https://github.com/strands-agents/docs/blob/main/docs/examples/python/weather_forecaster.md

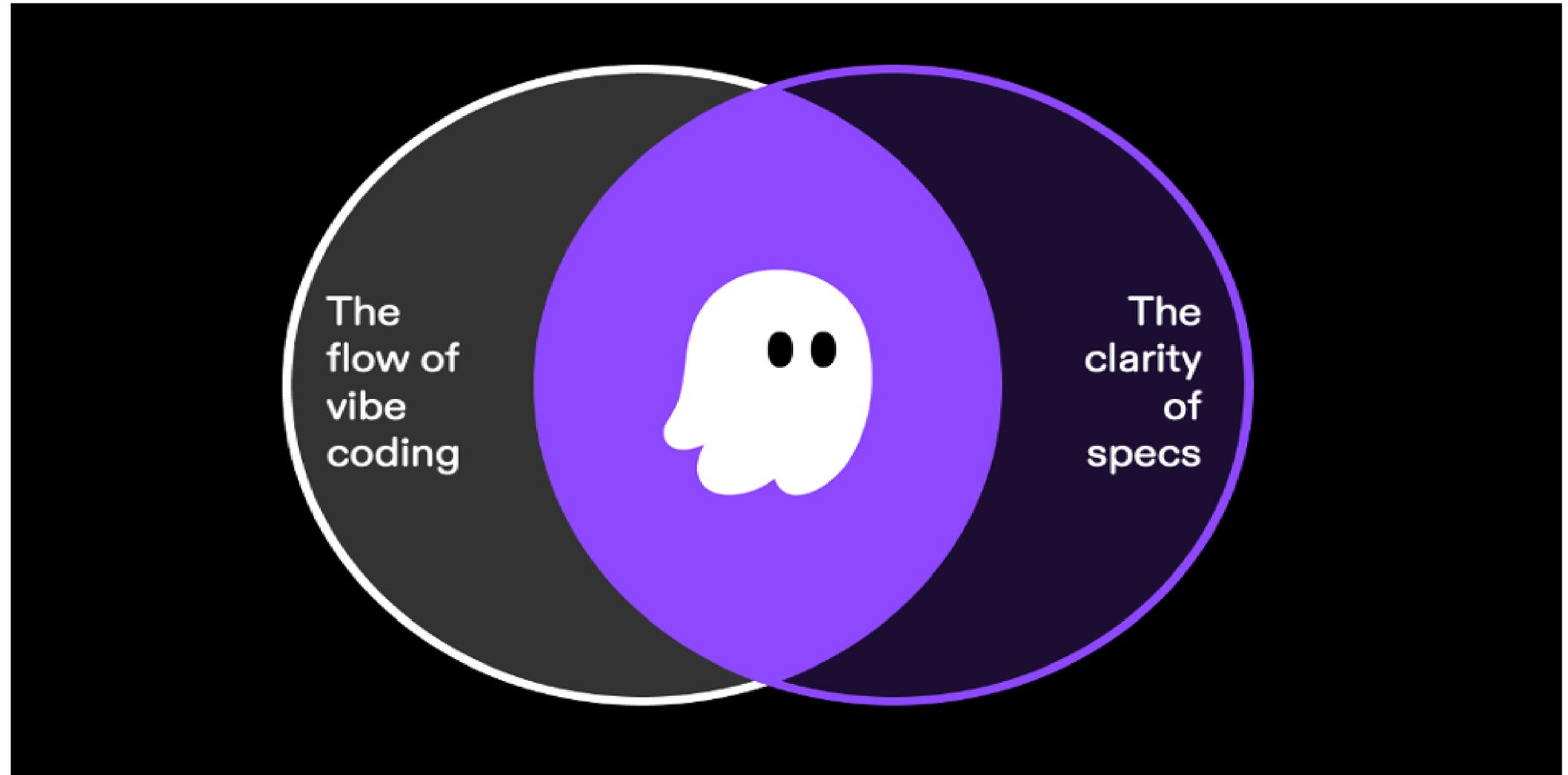
How many enterprise
grade agentic applications
have you seen?

I'm sure you've been there: prompt, prompt, prompt, and you have a working application. It's fun and feels like magic. But getting it to production requires more. What assumptions did the model make when building it? You guided the agent throughout, but those decisions aren't documented. Requirements are fuzzy and you can't tell if the application meets them. You can't quickly understand how the system is designed and how that design will affect your environment and performance. Sometimes it's better to take a step back, think through decisions, and you'll end up with a better application that you can easily maintain. That's what Kiro helps you do with spec-driven development.



<https://kiro.dev/>

An AI IDE that helps you deliver from concept to production through a simplified developer experience for working with AI agents. Kiro is great at ‘vibe coding’ but goes way beyond that—Kiro’s strength is getting those prototypes into production systems with features such as **specs** and **hooks**.



Core concepts



Kiro specs

are artifacts that prove useful anytime you need to think through a feature in-depth, refactor work that needs upfront planning, or when you want to understand the behavior of systems—in short, most things you need to get to production. Requirements are usually uncertain when you start building, which is why developers use specs for planning and clarity. Specs can guide AI agents to a better implementation in the same way.



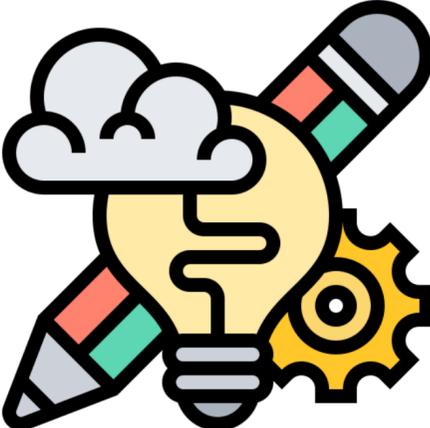
Kiro hooks

act like an experienced developer catching things you miss or completing boilerplate tasks in the background as you work. These event-driven automations trigger an agent to execute a task in the background when you save, create, delete files, or on a manual trigger.

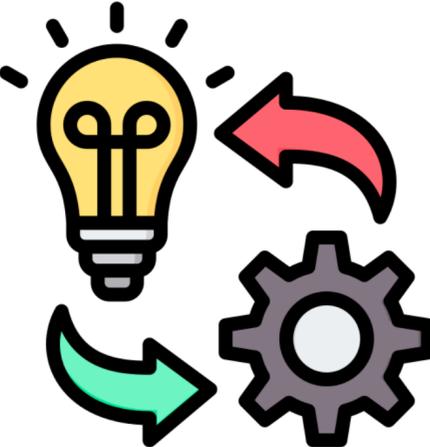
Kiro process



Requirements



Design



Implementation

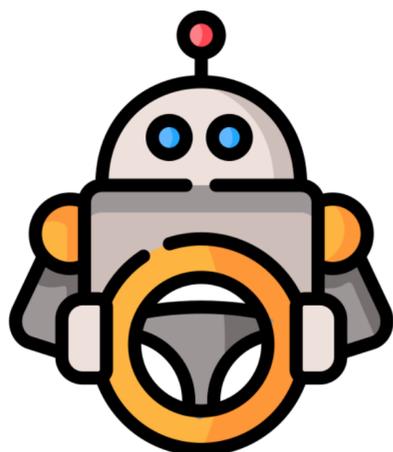
Kiro best practices

 **Security First:** Always add sensitive files to `.gitignore`. Never commit environment keys, secrets, or sensitive configuration files. Kiro's steering files help enforce these practices across your team. Ask Kiro in natural language: `Identify any sensitive/secret/config/env files from #codebase and also add all these files to project's .gitignore`

 **Stay Current:** Always use the latest package versions and npm. Kiro helps you maintain up-to-date dependencies automatically.

 **Avoid File Proliferation:** When working on changes and iterations, avoid the underlying Claude model creating multiple files with suffixes like `fixed`, `clean`, `new`, etc. Kiro's approach keeps your codebase clean and organized.

 **Leverage MCP Servers:** Always use MCP servers for AWS services and refer to existing markdown files for context. Don't create new markdown files - refer to the same context using MCP servers like `Context7` and `AWS Labs MCP server` for AWS docs.



Autopilot Mode: Maximum Velocity

- **Complete Code Implementation** - Watch as Kiro transforms ideas into working code in seconds
- **Rapid Iteration** - Slash development time by eliminating back-and-forth approvals
- **Maximum Efficiency** - Focus on the big picture while Kiro handles implementation details

For my AI Compliance Auditor project, Autopilot mode was perfect for implementing the foundational components: data models, Lambda functions, and infrastructure code.

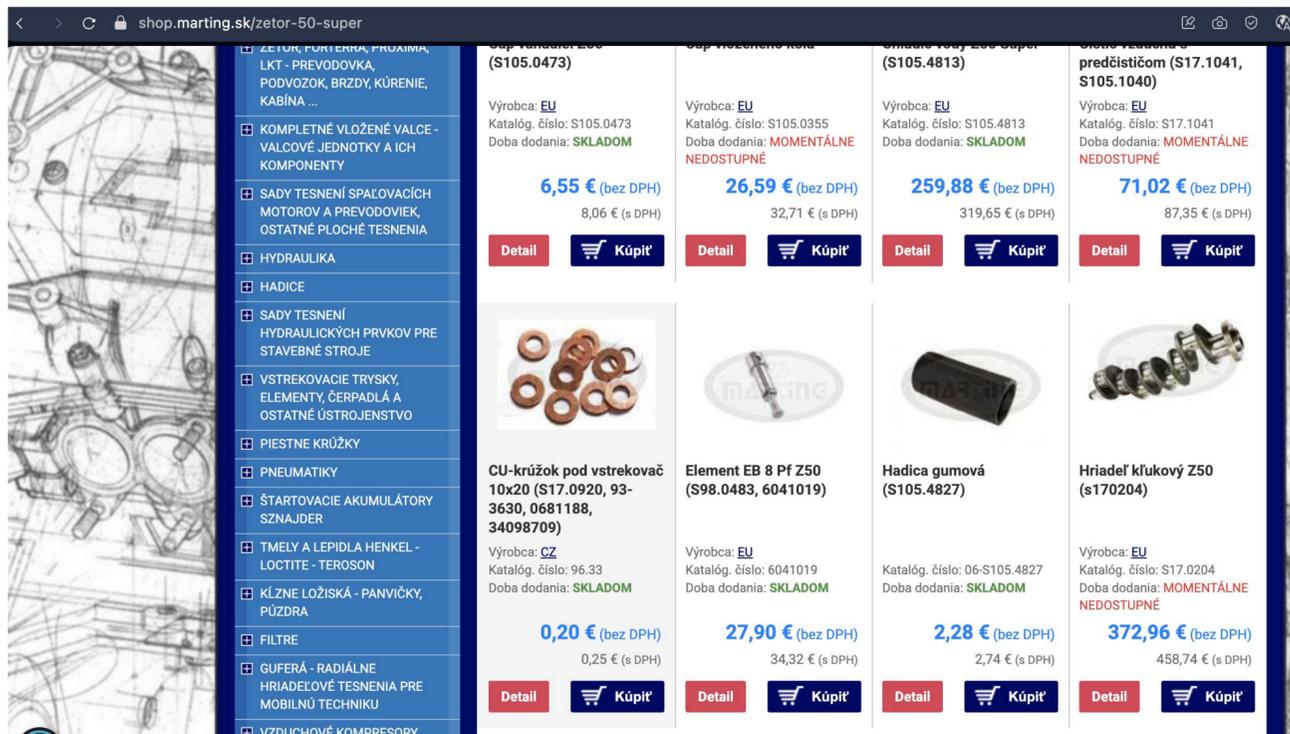


Supervised Mode: Collaborative Control

- **Request Approval** - Before making changes, Kiro presents its plan and waits for explicit approval
- **Show Detailed Steps** - See exactly what actions Kiro intends to take
- **Maintain Full Control** - Accept or reject proposed changes with convenient buttons

I used Supervised Mode for critical components like the policy engine and Bedrock intelligent prompt caching configurations, where I wanted to review every change.

Hi, I would like to build simple agentic app with crewai and fastmcp to search the web for tractor parts by parts name.



SPECS
 tractor-parts-search

AGENT HOOKS
 Automate repetitive tasks with smart triggers

AGENT STEERING
 Guide agent behavior and responses
 Generate Steering Docs

MCP SERVERS
 Connect external tools and data sources

specs > tractor-parts-search > requirements.md > # Requirements Document

Spec: tractor-parts-search 1 Requirements 2 Design 3 Task list

```

1 # Requirements Document
2
3 ## Introduction
4
5 This document outlines the requirements for a tractor parts search application that uses CrewAI for agent orchestration and FastMCP for web search capabilities. The system will allow users to search for tractor parts by name and retrieve relevant information from web sources.
6
7 ## Glossary
8
9 - Tractor Parts Search System: The complete application that searches for tractor parts information
10 - CrewAI: The multi-agent framework used for orchestrating search agents
11 - FastMCP: The Model Context Protocol implementation used for web search capabilities
12 - Search Agent: An AI agent responsible for conducting web searches for tractor parts
13 - Parts Query: A user-provided search term specifying the tractor part name
14 - Search Results: Structured information about tractor parts returned from web sources
15
16 ## Requirements
17
18 ### Requirement 1
19
20 User Story: As a tractor owner, I want to search for parts by name, so that I can find
  
```

Kiro

I'll help you create a spec for building an agentic app with CrewAI and FastMCP to search for tractor parts. Let me start by creating the requirements document for this feature.

Created requirements.md

I've created the initial requirements document for your tractor parts search application. The requirements cover the core functionality including:

- Searching for tractor parts by name using web search
- Using CrewAI for agent orchestration
- Integrating FastMCP for web search capabilities
- Returning structured search results
- Providing a simple command-line interface

Each requirement follows the EARS (Easy Approach to Requirements Syntax) format and includes specific acceptance criteria that are measurable and testable.

Do the requirements look good? If so, we can move on to the design.

Move to design phase

Ask a question or describe a task...

- design.md
- requirements.md

```

1 # Design Document
2
3 ## Overview
4
5 The Tractor Parts Search System is a Python-based
  application that leverages CrewAI for multi-agent
  orchestration and FastMCP for web search
  capabilities. The system provides a command-line
  interface for users to search for tractor parts
  by name and receive structured results from web
  sources.
  
```

```

6
7 ## Architecture
8
9 The system follows a layered architecture with
  clear separation of concerns:
  
```



```

25 ### Key Components:
26
  
```

Architecture: A layered approach with CLI interface, application layer, CrewAI agents, and FastMCP integration

Key Components:

- CLI interface for user interaction
- CrewAI agents (Search Coordinator and Web Search Agent) for task orchestration
- FastMCP integration for web search capabilities
- Result processing and caching systems

Data Models: Structured classes for PartResult, SearchQuery, and AgentResponse

Error Handling: Comprehensive strategy with retry logic, graceful degradation, and clear user feedback

Testing Strategy: Unit, integration, and performance testing approaches

The design leverages CrewAI's multi-agent capabilities to coordinate searches efficiently while using FastMCP to access web search functionality. The system includes caching for performance and robust error handling for reliability.

Does the design look good? If so, we can move on to the implementation plan.

Move to implementation plan

Ask a question or describe a task...

- design.md
- requirements.md
- tasks.md

```

1  # Implementation Plan
2
3  ⚡ Start task
4  - [ ] 1. Set up project structure and dependencies
5    - Create Python project structure with proper
6      package organization
7    - Set up pyproject.toml with CrewAI, FastMCP,
8      and other required dependencies
9    - Create configuration files for environment
10     variables and settings
11    - _Requirements: 3.1, 4.1_
12
13  ⚡ Start task
14  - [ ] 2. Implement core data models and utilities
15
16  ⚡ Start task
17  - [ ] 2.1 Create data model classes
18    - Implement PartResult, SearchQuery, and
19      AgentResponse dataclasses
20    - Add validation methods and serialization
      support
      - _Requirements: 2.1, 2.2_
21
22  ⚡ Start task
23  - [ ] 2.2 Implement cache manager
24    - Create in-memory cache with TTL support
25    - Add methods for storing, retrieving, and
26      cleaning expired entries
27    - _Requirements: 4.5_
28
29  ⚡ Start task
30  - [ ] 2.3 Create configuration management
  
```

```

# Implementation Plan
1.1.1 Set up project structure and dependencies
1.1.2 Create Python project structure with proper package organization
1.1.3 Set up pyproject.toml with CrewAI, FastMCP, and other required dependencies
1.1.4 Create configuration files for environment variables and settings
1.1.5 _Requirements: 3.1, 4.1_
1.2.1 Implement core data models and utilities
1.2.2 Create data model classes
1.2.3 Implement PartResult, SearchQuery, and AgentResponse dataclasses
1.2.4 Add validation methods and serialization support
1.2.5 _Requirements: 2.1, 2.2_
1.3.1 Implement cache manager
1.3.2 Create in-memory cache with TTL support
1.3.3 Add methods for storing, retrieving, and cleaning expired entries
1.3.4 _Requirements: 4.5_
1.4.1 Create configuration management
1.4.2 Implement configuration management
1.4.3 _Requirements: 4.5_
1.5.1 Testing
1.5.2 Comprehensive test coverage
1.5.3 (marked as optional)
1.6.1 Packaging
1.6.2 Entry point and deployment
  
```

- Project Setup:** Dependencies and structure
- Core Models:** Data classes and utilities
- FastMCP Integration:** Web search capabilities
- CrewAI Agents:** Multi-agent orchestration
- Result Processing:** Data aggregation and formatting
- Application Controller:** Main workflow coordination
- CLI Interface:** User interaction layer
- Logging & Monitoring:** Observability features
- Testing:** Comprehensive test coverage (marked as optional)
- Packaging:** Entry point and deployment

Each task references specific requirements and builds incrementally on previous tasks. The testing tasks are marked as optional to focus on core functionality first.

The current task list marks some tasks (e.g. unit tests, documentation) as optional to focus on core features first.

- Keep optional tasks (faster MVP)
- Make all tasks required (comprehensive from s

Waiting on your input... Cancel

Ask a question or describe a task... 48

▼ .kiro / specs / tractor-parts-search

- design.md
- requirements.md
- tasks.md

.kiro > specs > tractor-parts-search > tasks.md > # Implementation Plan

Spec: tractor-parts-search 1 Requirements 2 Design 3 Task list

```

1 # Implementation Plan
2
3 ⚡ Start task
4 - [ ] 1. Set up project structure and dependencies
5   - Create Python project structure with proper
6     package organization
7   - Set up pyproject.toml with CrewAI, FastMCP,
8     and other required dependencies
9   - Create configuration files for environment
10    variables and settings
11   - _Requirements: 3.1, 4.1_
12
13 ⚡ Start task
14 - [ ] 2. Implement core data models and utilities
15
16 ⚡ Start task
17 - [ ] 2.1 Create data model classes
18   - Implement PartResult, SearchQuery, and
19     AgentResponse dataclasses
20   - Add validation methods and serialization
21     support
22   - _Requirements: 2.1, 2.2_
23
24 ⚡ Start task
25 - [ ] 2.2 Implement cache manager
26   - Create in-memory cache with TTL support
27   - Add methods for storing, retrieving, and
28     cleaning expired entries
29   - _Requirements: 4.5_
30
31 ⚡ Start task
32 - [ ] 2.3 Create configuration management

```

Make all tasks required (comprehensive from s

Perfect! The spec is now complete with optional testing tasks to focus on building a faster MVP.

Your tractor parts search application spec is ready with:

- ✔ **Requirements:** 5 user stories with EARS-compliant acceptance criteria
- ✔ **Design:** Comprehensive architecture using CrewAI and FastMCP
- ✔ **Tasks:** 10 main implementation tasks with optional testing

You can now begin executing tasks by opening the `tasks.md` file and clicking "Start task" next to any task item. I recommend starting with task 1 to set up the project structure and dependencies.

The implementation will create a Python CLI application that uses CrewAI agents to coordinate web searches for tractor parts through FastMCP, with caching and structured result formatting.

Credits used: 0.95 Elapsed time: 14m 48s

> 3 changes accepted ([View all](#))

Revert X

Ask a question or describe a task...

49↑

EXPLORER: KIRO_DEMO

- ▼ .kiro / specs / tractor-parts-search
 - design.md U
 - requirements.md U
 - tasks.md U
- > tests
- > tractor_parts_search
- \$.env.example U
- gitignore U
- pyproject.toml U
- README.md U

```
tasks.md U | main.py U | .gitignore U
```

.kiro > specs > tractor-parts-search > tasks.md > # Implementation Plan

Spec: tractor-parts-search 1 Requirements 2 Design 3 Task list

```
1 # Implementation Plan
2
3 Task in progress | View changes
4 [-] 1. Set up project structure and dependencies
5   - Create Python project structure with proper
6     package organization
7   - Set up pyproject.toml with CrewAI, FastMCP,
8     and other required dependencies
9   - Create configuration files for environment
10    variables and settings
11   - _Requirements: 3.1, 4.1_
12
13 Start task
14 [-] 2. Implement core data models and utilities
15   Start task
16   - [-] 2.1 Create data model classes
17     - Implement PartResult, SearchQuery, and
18       AgentResponse dataclasses
19     - Add validation methods and serialization
20       support
21     - _Requirements: 2.1, 2.2_
22
23 Start task
24 [-] 2.2 Implement cache manager
25   - Create in-memory cache with TTL support
26   - Add methods for storing, retrieving, and
27     cleaning expired entries
28   - _Requirements: 4.5_
29
```

Notifications may include icon badges.

Created main.py

Created __init__.py

Created conftest.py

Searched workspace

Verify the complete project structure has been created correctly

Perfect! Now let me verify that the configuration works correctly by testing it:

```
python -c "from tractor_parts_s"
```

The command has been flagged for manual review. Learn more about how commands are evaluated for safety.

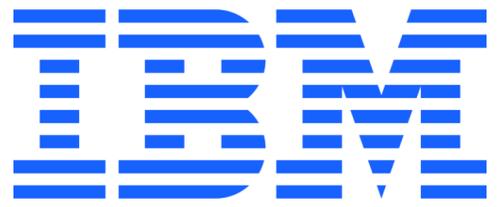
Waiting on your input.. Reject Trust Run

Tips & tricks

- 1.Start with the Spec** - Never jump straight into coding. Use Kiro's spec workflow to clarify requirements first. You can also use the free-form chat mode whenever needed for general assistance and vibe coding.
- 2.Leverage MCP Servers Extensively** - The Context7 and AWS Labs MCP servers provide incredible value. Use them for every AWS related task.
- 3.Configure Steering Early** - Set up your steering files immediately. They dramatically improve Kiro's contextual understanding.
- 4.Use Agent Hooks for Repetitive Tasks** - Automate git commits, documentation updates, and code quality checks.
- 5.Mix Autopilot and Supervised Modes** - Use Autopilot for foundational work, Supervised for critical components.
- 6.Keep Tasks Granular** - Break down complex features into small, manageable tasks in your tasks.md.
- 7.Iterate on Requirements** - Don't be afraid to refine your requirements as you learn more about the problem space.
- 8.Reuse your hooks, steering, and reusable prompts** - Create reusable templates of your `.kiro/hooks/`, `.kiro/steering/`, and `.prompts/` directories to accelerate new project setup.
- 9.Leverage Natural Language Automation** - Use natural language prompts to create your own agent hooks and agent steering rules to automate everything.

Key takeaways

- Agentic AI empowers more autonomous and proactive systems.
- Design patterns and architectures are crucial for robust and scalable solutions.
- Frameworks like CrewAI and AWS Strands, combined with strong orchestration (FastMCP), accelerate development.
- Tools like Kiro helps building robust enterprise grade agentic applications



Icons downloaded from
<https://www.flaticon.com/>