WAVESTONE

AGENTIC AI
PLAYBOOK

From conversational assistants to autonomous agents: a comprehensive journey through the design and orchestration of AI Agents

Agenda

Introduction

From conversational intelligence to agentic AI: A new era for intelligent automation

Generative AI has seen an unprecedented acceleration over the past few years. Once confined to research labs, it has rapidly moved into the mainstream tech landscape. The tipping point? The release of ChatGPT in 2022, a breakthrough that brought this technology into the hands of millions. That pivotal moment turned a niche research field into a global phenomenon, reshaping how we work, access information, and approach automation.

From text generation to planning and action

Generative AI is no longer just about producing text or powering chatbots. It's evolving into a more structured, proactive, and operational form of intelligence. Today's models go beyond generating responses—they plan, reason, and execute. They're becoming systems that can handle complex tasks, adapt to diverse environments, and act on behalf of users. This shift, gradual yet fundamental, is ushering in the age of agentic AI.

Recent developments such as Chain of Thought reasoning have equipped language models with the ability to break down complex problems into logical steps. This structured approach leads to more accurate, coherent, and abstract outputs—an essential capability for tackling real business use cases where a simple text response is no longer enough.

Smarter models for more complex tasks

Models like OpenAI's o3 and o4-mini, DeepSeek R1, IBM's Granite 3.2, and Google's Gemini 2.5 exemplify this shift. With enhanced planning, contextual memory, and multi-step reasoning capabilities, they can follow through multiple rounds of interaction and execute sophisticated sequences of actions. These technical leaps are not just evolutionary—they signal a profound transformation: the move from generation to action. AI is stepping out of its assistant role to become an active player.

An AI agent is more than a powerful model. It's an integrated system designed to take ownership of an overarching goal: devising a strategy, selecting the right tools, executing the necessary steps, and adapting along the way. It operates in digital environments much like a human assistant—considering context, managing constraints, and making intermediate decisions to reach a defined outcome.

Take a simple example: planning a workweek. While a voice assistant might suggest available time slots, an AI agent goes further—it accesses the calendar, identifies availabilities, prioritizes meetings, checks for scheduling conflicts, proposes options, and sends invitations directly to participants. It doesn't just respond—it takes charge, from start to finish.

This new capability relies on a specific architecture, where the language model is embedded within a broader system of specialized functions—commonly referred to as "tools." These tools allow the agent to interact with databases, APIs, enterprise systems, or even other agents. At each step, the agent selects the most appropriate action based on its context and objective.

The Model Context Protocol (MCP) is emerging as the standard for tool integration, enabling agents to dynamically discover, understand, and use tools through standardized interfaces. Meanwhile, new agent-to-agent communication protocols—such as A2A (Agent-to-Agent) and ACP (Agent Communication Protocol)—are beginning to define how agents delegate, coordinate, and collaborate in distributed environments.

Framing agent autonomy

Not all agents operate at the same level of autonomy. Some are tightly governed by predefined rules. Others make decisions independently. The most advanced can generate their own code or adapt their behavior in real time to handle unfamiliar scenarios. This growing agentic capability opens up vast new possibilities—but it also raises critical questions around governance, security, and ethics.

Delegating a task to an agent means granting it a degree of initiative. It means accepting that it may make decisions—sometimes sensitive ones—without immediate human oversight. This shift calls for a redefinition of AI's role within information systems: introducing supervision mechanisms, managing access rights, and implementing traceability for every action taken. The power of agentic AI demands strict control over how it behaves.

A turning point for computing

Agentic systems represent a fundamental shift: software is no longer just a passive executor of instructions. It becomes a semi-autonomous actor—capable of interpreting, deciding, and operating in open-ended contexts. This evolution is redefining the very notion of intelligent automation and laying the groundwork for a new generation of digital systems: more flexible, adaptive, and collaborative by design.

To support this growing autonomy, enterprise architecture must evolve. Language models alone are no longer sufficient, they need to be embedded in a robust ecosystem, complete with new orchestration and governance layers to fully realize the potential of agentic AI.

This white paper aims to explore the technical foundations of AI agents. It is intended for architects, developers, and business leaders who want to look beyond the hype and gain a clear understanding of the underlying mechanisms that power these systems. From agent design to orchestration in distributed environments, it lays the groundwork for the controlled integration of AI agents into modern, robust, and scalable information systems.

01. First concepts: understanding the foundations of agentic AI

An introduction to how AI agents operate and how they can carry out tasks autonomously and adaptively

Chapter 1

First concepts: the foundations of agentic AI

An artificial intelligence agent is a system designed to perform a task on behalf of a user—autonomously, in a structured way, and with the ability to adapt. Unlike traditional applications that follow fixed instructions, an agent relies on a reasoning engine—most often a large language model (LLM*)—and interacts dynamically with its environment to achieve a defined goal.

An agent is built to manage the end-to-end execution of a workflow. It doesn't just provide information—it acts on the user's behalf. This ability rests on three core components: a reasoning engine (LLM), a set of tools to interact with external systems, and a framework of instructions that defines its operational boundaries.

The execution loop: how agents operate

At the heart of agent behavior lies a structured execution loop made up of four key phases:

Analysis: The agent receives a task or request expressed in natural language. It analyzes the input to understand the goal, context, constraints, and available data.

Planning: Based on this analysis, the agent builds a strategy, breaks the objective down into subtasks, and determines the sequence of actions to be performed. This planning may be explicit—as a list of steps—or implicit, embedded in the agent's internal reasoning.

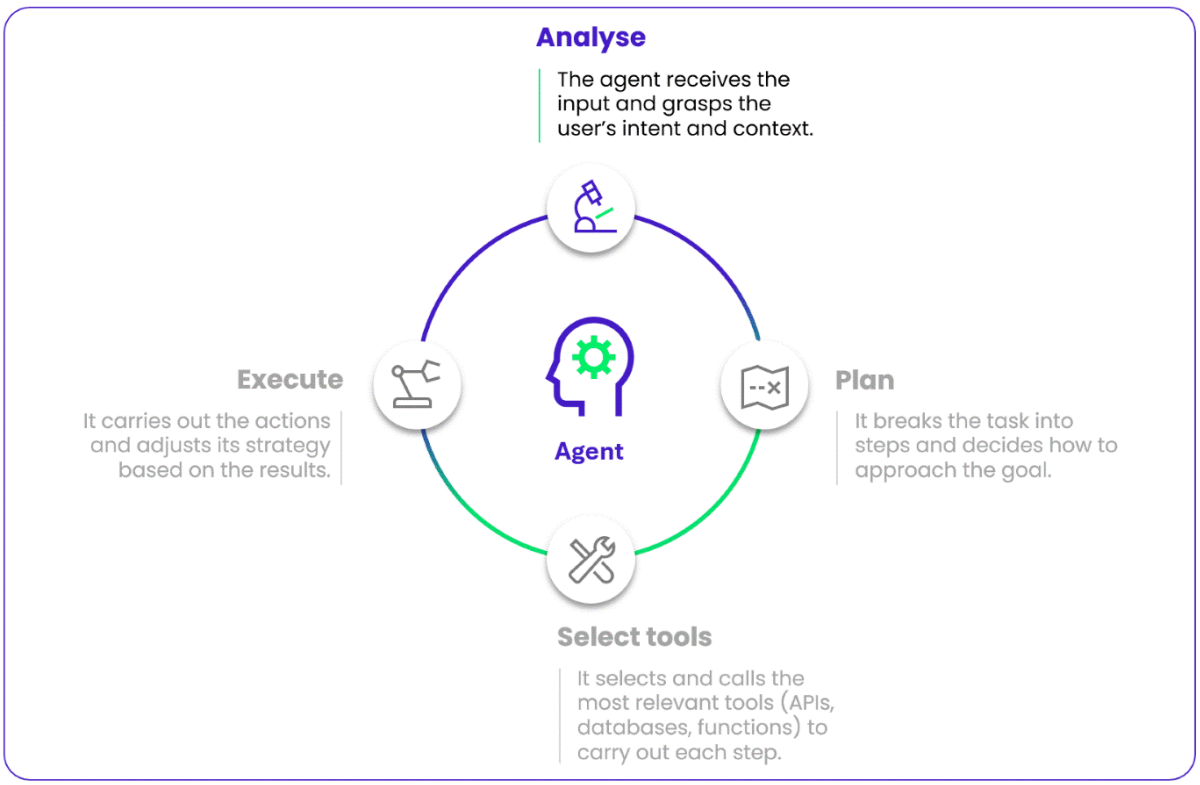
Tool selection: The agent identifies the most relevant tools—APIs, functions, databases, scripts, and so on—from its toolkit to carry out each step in the plan.

Execution: The agent performs the planned actions, observes the outcomes, detects potential errors or anomalies, and adjusts its strategy if needed.
This process repeats itself iteratively and dynamically until the goal is reached. With each cycle, the agent re-evaluates the situation, incorporates new information, and continuously refines its behavior to respond to changes or unexpected conditions.

**An LLM (Large Language Model) is an artificial intelligence model based on a transformer architecture. It is trained on massive textual datasets to predict and generate coherent, context-aware sequences of natural language. Today's most advanced models—such as those in the GPT family (Generative Pre-trained Transformer)—are built on this architecture.*

This architecture enables the agent to break down complex tasks into smaller, manageable steps—each of which can leverage a specialized tool. This modular approach improves both execution robustness and the accuracy of outcomes.

This is what's known as an agentic workflow: rather than simply generating a one-off response, the model actively orchestrates a sequence of actions—with built-in control and oversight at every stage.



With each iteration, the agent consumes tokens, the basic units of computation used to process text (inputs, outputs, and context). This directly impacts execution costs, especially in environments where the agent interacts with large, powerful language models. It's therefore crucial to keep this constraint in mind when designing an agent, in order to strike the right balance between autonomy, decision relevance, and operational efficiency.

Evaluating when to use an AI agent

Not every process justifies the use of an agent. However, according to *A Practical Guide to Building Agents* (OpenAI) and *Building Effective Agents* (Anthropic), there are three key criteria that help determine whether a given workflow truly benefits from an agentic approach:

1. Decision complexity

Agents are particularly well suited to tasks that require nuanced decision-making. This includes situations where choices must be weighed, exceptions managed, weak signals interpreted, or conflicting information reconciled.

2. Fluid or hard-to-codify business rules

In some workflows, business rules aren't necessarily unstable but they're difficult to formalize algorithmically. They involve edge cases, exceptions, or depend on contractual, legal, or contextual nuances that require semantic understanding rather than brute logic.

For example, a supplier compliance review in a large industrial or energy company. Each supplier is evaluated based on documents like internal policies, safety standards, or regulatory guidelines. These documents are often lengthy, complex, and their interpretation depends on operational or legal context.

In such cases, an LLM-based agent can step in—not to enforce fixed rules, but to interpret the content, identify ambiguous scenarios, and reason like a compliance expert. These agents are especially valuable when the rules are understandable in natural language but difficult to translate into deterministic code.

3. Nature of the data involved

Agents shine when dealing with unstructured data: free text, helpdesk tickets, emails, PDFs, meeting notes, or Teams conversations—all of which are challenging for traditional systems to process effectively. An agent can read, extract, cross-reference, and synthesize this information as part of its reasoning process.

A common use case is customer support. An agent can identify key elements (product, issue, tone of the message), consult internal documentation, draft a personalized response, and even trigger follow-up actions—such as issuing a refund, escalating the request, or updating a support ticket.

When a process goes beyond rigid, sequential logic, when it requires judgment, adaptive rules, or the handling of unstructured data—an AI agent becomes a strategic enabler. It doesn't just automate. It understands context, reasons like a human, and takes action accordingly.

AI agents are not a one-size-fits-all solution and in some contexts, they simply aren't the best choice. If a process is strictly linear, with no intermediate decisions, clearly defined and stable rules, and well-structured data that can be queried via SQL or APIs, then using an agent adds unnecessary complexity.

Similarly, in scenarios where every step requires human supervision or where real-time traceability is critical—such as in accounting or highly regulated environments—simpler automation approaches like RPA, scripting, or webhooks are often more appropriate. They're easier to audit, more predictable, and typically more reliable.

Concrete examples of the tipping point

Take the case of a company that handles 500 support tickets per day. A simple rules-based system is used to triage requests based on keywords. It works—until the requests become more complex:
"I got an error after trying to update my profile."
"My product arrived damaged, and I haven't heard back since my first email."

At that point, the rules engine starts to break down. It misclassifies requests. It misses context. It creates frustration.

In such cases, an agent can step in. It reads the message, understands the intent, checks the customer's history, reviews product documentation, proposes a tailored response—and can sometimes resolve the issue without human intervention. It doesn't just follow rules—it understands and acts.

Another example: a marketing team wants to assess the impact of a campaign. A traditional dashboard provides static KPIs—click-through rates, conversions, impressions. An AI agent, on the other hand, goes further. It pulls and structures data from multiple sources: CRM, social media, web analytics, internal surveys.

It identifies key trends, surfaces weak signals (like a sudden spike in negative mentions or an unexpected keyword), and starts formulating hypotheses. From there, it can generate an executive summary, suggest areas for improvement, or propose an A/B test. More than that, it can trigger actions: alert a team, schedule a corrective post, or draft a brief for the next campaign. It's no longer just an analysis tool—it's a decision partner that observes, anticipates, and acts.

A decision that depends on context

Building an AI agent comes at a cost—in time, infrastructure, and oversight. It also requires a certain level of technical and governance maturity. That's why it's important not to "over-agentize." The right approach is to start from the actual business need, assess the true complexity of the workflow, and choose the architecture that fits best. Sometimes, simpler really is smarter.

02. Different levels of agentic intelligence

Exploring the various degrees of agent autonomy and assessing their technological maturity

Chapter 2

Different levels of agentic intelligence

Not all AI agents are created equal. The term "agent" covers a wide spectrum of systems—from basic conversational assistants to entities capable of generating and executing their own code. This diversity calls for a more nuanced classification: one based on levels of agentic intelligence.

Agentic intelligence refers to a system's capacity to act autonomously within a given environment. It goes beyond answering a question or executing a predefined command. It involves initiative, the ability to manage uncertainty, to plan actions—and, in some cases, to create new tools or strategies to achieve a goal. This capacity exists on a continuum, from passive assistant to fully generative agent.

To structure this diversity, we draw on a classification inspired by AI company Hugging Face, which defines five levels of agentic capability, from 0 to 4. This framework is a useful tool to assess system maturity, anticipate supervision requirements, and calibrate operational expectations.

Level 0 (☆☆☆☆) – Conversational assistant: replies without action

At this level, the agent takes no initiative. It simply responds to user questions or commands, without triggering any external actions. It may access data sources (search engines, internal knowledge bases), but remains fundamentally passive.

It doesn't decide when to act, nor how: there is no functional autonomy. This is the realm of basic chatbots—a more sophisticated interface, perhaps, but not a true agent.

Level 1 (★☆☆☆) – Deterministic workflow agent: guided execution

Here, the agent can decide when to act, but not how. It follows pre-defined paths built by developers, behaving like an intelligent router.

Typically, it identifies user intent and routes the request to a predefined API, tool, or workflow. It doesn't construct strategies or combine actions on its own.

This is the most common model in industrial contexts today: useful, robust, but with very limited autonomy.

Level 2 (★★☆☆) – Semi-autonomous agent

At this stage, the agent can decide both when to act and which tool to use, along with relevant parameters. It assembles sequences of API calls based on context.

This marks the beginning of real functional autonomy—albeit within clear limits. The agent knows which tools are available and how to use them to reach a simple objective.

Example: a customer service agent that understands a query, queries a database, extracts the right information, and generates a personalized response. It acts independently, but within a tightly scoped domain.

Level 3 (★★★☆) – Autonomous orchestration agent

At this level, the agent can plan multiple steps to solve a problem, adapt its strategy as it goes, and decide whether to continue or stop.

It can chain actions, coordinate multiple tools or sub-agents, and structure its own logic. This is where agentic decision-making matures: the agent becomes a conductor orchestrating a dynamic process.

We're talking here about advanced personal assistants, agents that can manage schedules, prioritize tasks, or coordinate multiple systems. Such use cases remain rare in production due to their complexity and the need for strict oversight.

Level 4 (★★★★) – Fully autonomous agent

This is the most advanced stage. The agent can create its own means of action: it writes code, executes it, and adapts its behavior to new or evolving environments.

Such an agent can detect a limitation in its current capabilities, write a Python function to overcome it, run that code, incorporate it into its toolkit, and use it to fulfill its objective. It can devise and revise strategies without human intervention.

While this level of autonomy unlocks powerful new possibilities, it also raises serious concerns: around security, governance, and traceability. These agents remain largely experimental today.

Where do we stand today?

In most real-world applications, operational agents currently sit between level 0 and level 1. That means they can decide when and how to act using pre-defined tools—but only within tightly controlled parameters.

They respond to queries or route them to existing workflows, without true decision-making or adaptive reasoning. These systems are deterministic, robust, and auditable—qualities that are still highly valued in enterprise environments.

Level 2 agents—those capable of dynamically calling tools and adapting their behavior to context—are starting to emerge, especially in startups where process flexibility and agile tech stacks allow for more experimentation. These agents are typically used in individual or exploratory contexts, often by developers, innovation teams, or technical profiles.

However, they are not yet deployed at scale. The lack of full verifiability, the potential for errors, and the need for close supervision still limit their production use.

Full autonomy—levels 3 and 4—remains largely in the realm of advanced research. It introduces major governance challenges, as it implies giving a system not only the authority to act but also to reconfigure itself. Delegating that kind of power is only acceptable if paired with extremely robust guardrails, oversight mechanisms, and validation processes.

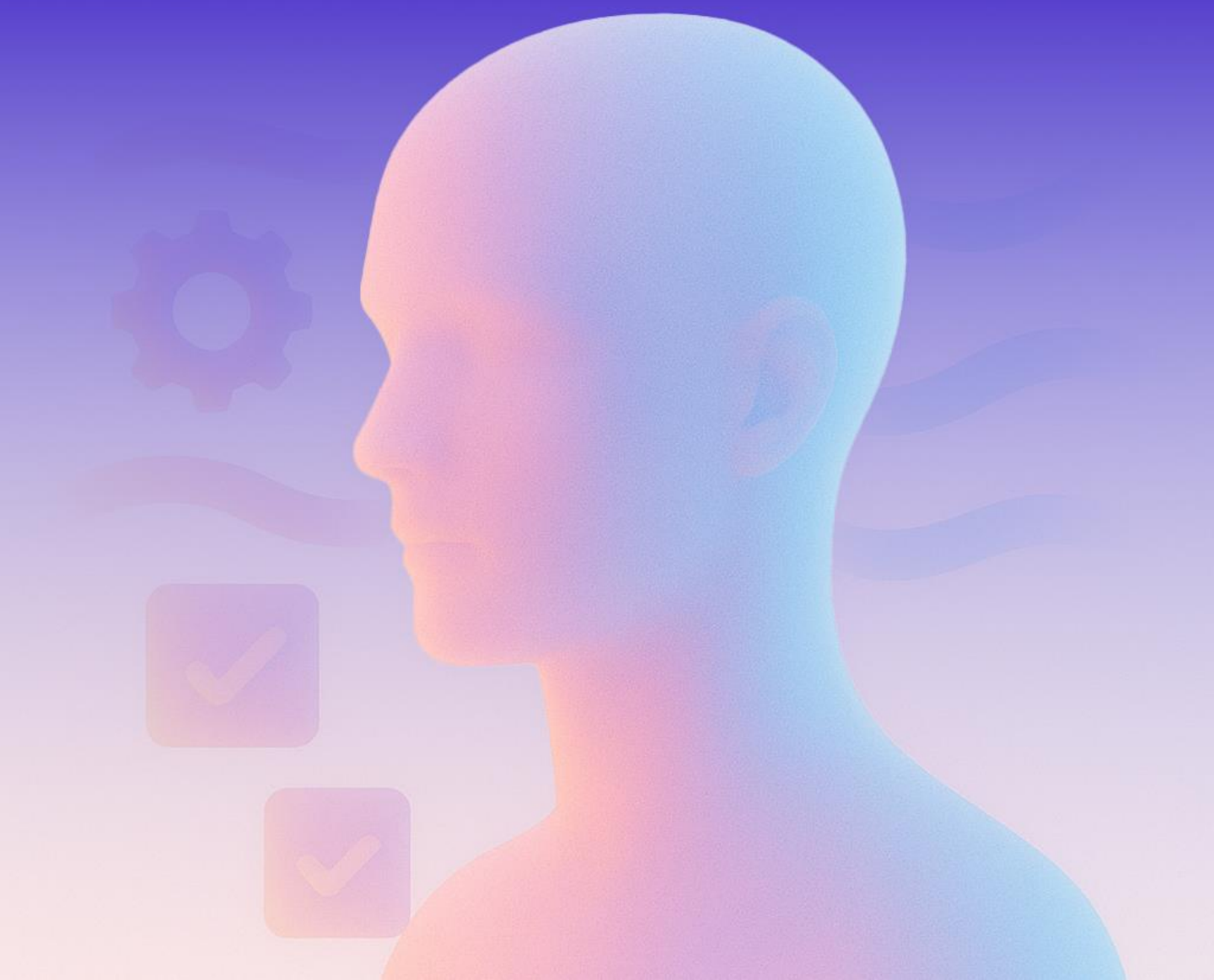
A spectrum, not a label

It's important to recognize that this typology isn't a set of rigid boxes—it's a continuum. An agent's level of autonomy can shift depending on the task at hand, the permissions granted, or the level of supervision in place.

Agentic capability is not an absolute property—it's contextual.
This graduated approach allows technical teams to better scope their ambitions, define acceptable levels of risk, and design agents that are aligned with real-world needs—rather than an idealized vision of what AI *could* do.

03. Core components of an AI Agent

Models, tools, and instructions—the foundational building blocks of agentic AI. An introduction to agent communication protocols (MCP, A2A, and ACP)



Chapter 3

Core components of an AI agent

An AI agent is more than just a language model wrapped in a user interface. It's a composite system whose ability to act autonomously depends on the interplay between several foundational components. Understanding the nature and role of these building blocks is essential to designing agents that are effective, stable, and controllable.

The exact architecture may vary depending on the implementation, but every agent relies on a core triad: a model, a set of tools, and a framework of instructions. These three elements form the agent's backbone. It's the combination—and orchestration—of these components that defines the system's capabilities, boundaries, and degree of autonomy.

The model: the agent's reasoning engine

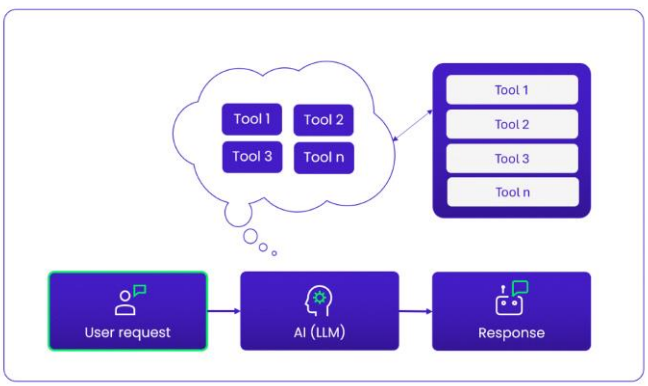
The model serves as the agent's cognitive core. It interprets user input, formulates action plans, decides which tools to invoke, and reacts to intermediate results. It plays the role of the "brain" in the agent's architecture.

In most cases, this model is a Large Language Model (LLM)—such as GPT-4 (OpenAI), Claude (Anthropic), Mixtral (Mistral), or Llama (Meta). This choice is far from trivial: different models offer different trade-offs in terms of performance, cost, stability, and capabilities. Some are optimized for speed, others for reasoning, memory, or multi-document processing.

According to OpenAI's Practical Guide to Building Agents, a recommended best practice is to start with the most capable model available in order to establish a solid functional baseline. Once that's done, it's possible to explore model downsizing—assessing the acceptable trade-offs in performance to optimize for cost.

It's also possible—and often advisable—to combine multiple models within a single agentic system. A lightweight model might handle simpler tasks (intent detection, quick classification), while a more advanced one takes over for complex reasoning or planning stages.

Finally, the agent architecture should remain flexible enough to swap one model for another as the ecosystem evolves. Because LLMs are improving rapidly, a sound technical design includes abstraction layers and fallback mechanisms that allow new models to be integrated without needing to overhaul the entire agent framework.



An agent doesn't just generate text, it takes action. To act, it must be able to manipulate digital objects, interact with systems, query databases, and execute business functions. That's where tools come in.

A tool is an external function made available to the agent. It could be an API, a SQL query, a Python function, a shell script, or even a simulated click within a user interface. The agent doesn't need to understand how the tool works internally—it only needs to know what it does and how to invoke it.

At initialization, the agent is provided with a list of available tools. Too many poorly defined tools can create confusion. Too few tools make the agent ineffective. The ideal setup includes a small number of clearly named, well-documented tools that have been independently tested outside the model loop.

This logic can be extended through a modular sub-agent architecture, where each sub-agent functions like a tool in its own right. For instance, a "Mail" agent might bundle capabilities like searching emails, sending messages, or creating calendar invites. Other agents can then delegate tasks to it, treating it as a callable resource. This agent-as-tool model promotes modularity, reusability, and tighter governance over complex behaviors.

The system prompt: instructions that define the agent's behavioral framework

A model and tools alone are not enough. To behave in a consistent, goal-aligned, and predictable way, an agent needs explicit instructions. These are typically provided through a system prompt—a foundational text that defines the agent's role, responsibilities, constraints, and priorities.

System instructions serve as a code of conduct. They tell the agent what it is supposed to do, under what circumstances, and with which boundaries. These instructions may include:

- **Role definition**: "You are an agent responsible for handling customer support inquiries related to digital products."
- **Behavioral rules**: "Always greet the user, remain neutral, and never promise a refund without verification."
- **Procedural logic**: "If the user requests a password reset, call the reset_password tool."
- **Edge case handling**: "If the user is angry or threatening, escalate immediately to a human operator."

Writing these instructions is a critical step. It requires a careful balance between clarity, completeness, and brevity. Vague prompts result in unpredictable behavior; overly detailed prompts can make the agent rigid or overly verbose. A good practice is to build on existing operational content—support scripts, internal charters, or procedure manuals—to draft high-quality prompts.

In more advanced systems, instructions can be dynamic. A single agent may adapt its role depending on the context, shift priorities based on real-time signals, or integrate session-specific parameters (user name, interaction history, priority level, etc.).

These three components—model, tools, and instructions—form the backbone of any AI agent.They define what the agent can do, how it does it, and why it takes action. The way these elements are connected determines not only the agent's performance, but also its ability to operate in real-world environments, collaborate with humans or other agents, and comply with the technical and ethical requirements of an information system.

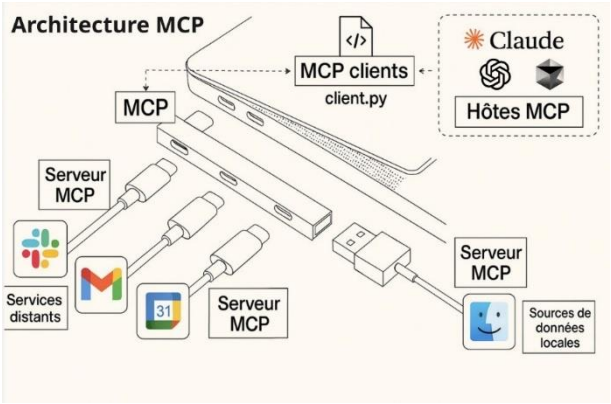
The Model Context Protocol (MCP): a standard for tool integration

In the context of tool-enabled agents, a new standard has emerged to simplify the connection between AI agents and external resources: the Model Context Protocol (MCP). Introduced by Anthropic in November 2024, MCP provides a unified method for exposing tools to agents—without requiring complex technical adaptations.

MCP acts as a universal interface, much like what HTTP represented for the web. It defines a common schema for describing available tools: their functions, accepted parameters, and usage rules. This standardization allows any agent—regardless of its underlying model—to understand how to use a given tool, provided the tool complies with the MCP specification.

In practical terms, an MCP-compatible tool can be discovered, queried, and invoked by an agent through a formal schema—without rewriting integration code. This dramatically simplifies multi-agent deployments, business logic reuse, and interoperability across frameworks.

For LLMOps teams, MCP streamlines the path to production: it reduces integration time, improves the reliability of tool calls, and simplifies testing and monitoring. For business teams, it ensures that agents remain interoperable with critical information system assets while allowing for better scalability.

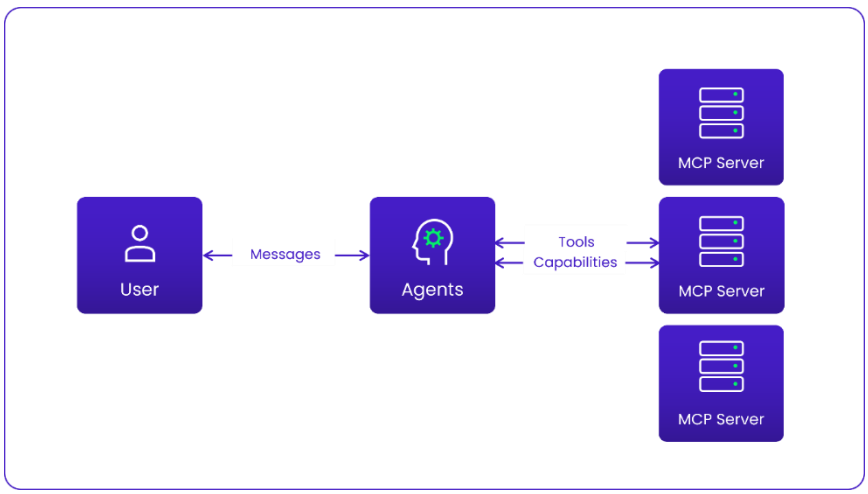


Source : https://norahsakal.com/blog/authors/norah/

Source: https://norahsakal.com/blog/authors/norah/

MCP is to artificial intelligence what HTTP was to the web: a universal standard that enables seamless interaction between AI models and external environments.

Designed to simplify how agents connect to tools, data, and services, MCP has been adopted by major players including Microsoft (Copilot Studio, Azure AI Agent, Autogen), Amazon (Bedrock), OpenAI (Agent SDK, announced for ChatGPT Desktop), Google (Agent Development Kit – ADK), and IBM. It is now supported by most agent frameworks.



04. Orchestrating Agents

Designing early agentic architectures and coordinating agent behavior

Chapter 4

Orchestrating agents

Designing an AI agent isn't just about picking a model, giving it tools, and writing a prompt. These components must be brought together within a coherent execution logic—one that can turn a user request into a structured sequence of actions. That's the role of orchestration.

Orchestration refers to the set of mechanisms that guide the agent over time: how it sequences its actions, how it makes decisions at each step, and how it handles uncertainty, errors, or unexpected situations. An agent doesn't act just once—it reacts, observes, adjusts, and iterates. This dynamic process—this continuous loop of reasoning, action, and observation—is at the heart of how agentic systems function.

As discussed earlier, an AI agent operates through a central execution loop known as the agent loop. This loop is typically structured in four main stages, led by the language model:

🔍 **1. Analysis :** The agent receives an input (task, request, instruction), analyzes the context, interprets user intent, and identifies constraints and available data.

🧭 **2. Planning** : It builds a strategy, breaks the goal into steps, selects an approach, and anticipates dependencies or success conditions.

🛠️ **3. Tools selection :** It chooses the most relevant tools (APIs, functions, databases, etc.) for each step, determines how to invoke them, and in what order.

⚙️ **4. Execution** : It carries out the actions, observes the outcomes, detects errors or anomalies, and adjusts the plan if necessary. The loop then restarts, continuing until the task is complete.

This iterative, adaptive cycle allows the agent to continuously refine its actions based on context and feedback. This view of orchestration—and the architectural patterns it implies—is shared by both OpenAI and Anthropic in their foundational work on autonomous agents.

Single-agent orchestration

In simple architectures, a single agent handles the entire process and makes all decisions. This is the single-agent model. The agent receives a request, decides what to do, calls the appropriate tools, evaluates the outcome, and loops until completion.

This approach works well when the task is well-defined, limited to a single domain, or short enough that no delegation is required. It's also easier to develop, test, and monitor. Most early agents deployed in production have followed this model.

However, as task complexity increases—due to a wider range of actions, domains, or longer execution times—this model starts to show its limits. It becomes difficult for a single agent to handle all aspects of a distributed or specialized workflow efficiently.

Multi-agent architectures: specialization and coordination

To overcome the limitations of single-agent setups, more advanced systems turn to multi-agent architectures. In this model, several agents work together, each taking on a specific role in completing the task. Orchestration then becomes a matter of coordination: who does what, when, and under which interaction rules.

According to OpenAI's Practical Guide to Building Effective Agents, there are two main patterns for orchestrating multiple agents.

The hierarchical model: the manager agent

In this approach, a primary agent—known as the manager—receives the user request. It analyzes the task, breaks it down into subtasks, and delegates each subtask to a specialized agent.

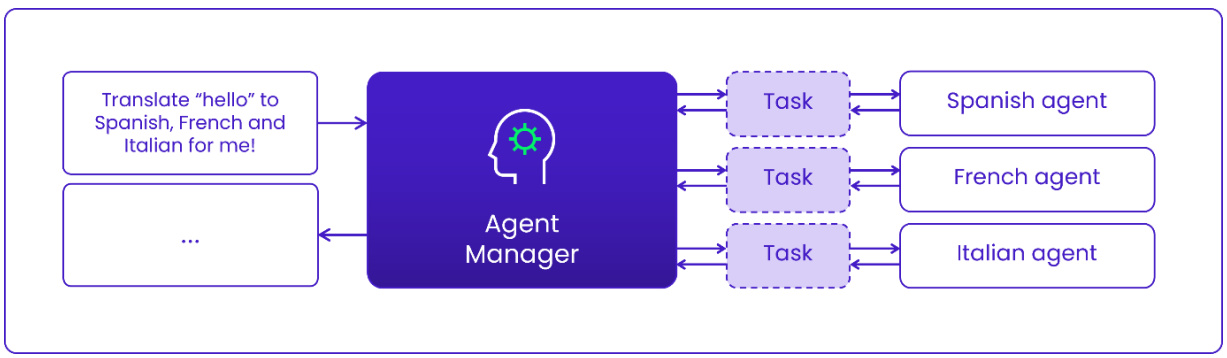
Each sub-agent is assigned a clearly defined mission. It operates autonomously, using its own tools and following its own set of instructions. Once completed, it returns a partial result to the manager.

The manager then consolidates all outputs and delivers a final response to the user.This model is especially well suited for workflows where each step involves a distinct domain of expertise—such as information extraction, semantic processing, business decision-making, or communication. It enables cognitive load distribution, modular behavior design, and the reuse of specialized agents across different contexts.

Today, most agent frameworks rely on handoffs to structure coordination between agents. These handoffs define who does what, and when.

During the design phase, it's critical to clearly specify these transitions: each agent must be assigned a well-defined task, with clearly structured input and output formats—distinct from those of other agents.

This clarity ensures smooth cooperation, avoids ambiguity, and helps maintain robustness even as the system scales or evolves.



Orchestration - agent manager that routes a translation task to specialized sub-agents — OpenAI example

The collaborative model: agent-to-agent dialogue

In a more distributed setup, agents operate on equal footing and interact through dialogue. Each agent is autonomous, capable of initiating or responding to interactions, and contributes to the collective resolution of the task.

This architecture—rooted in distributed AI and multi-agent systems—is more challenging to control, but it opens up compelling opportunities in terms of adaptability, resilience, and creative problem-solving. It enables agents to brainstorm, challenge each other's hypotheses, and validate each other's plans in real time.

A hybrid architecture: manager-led coordination with collaborative sub-agents

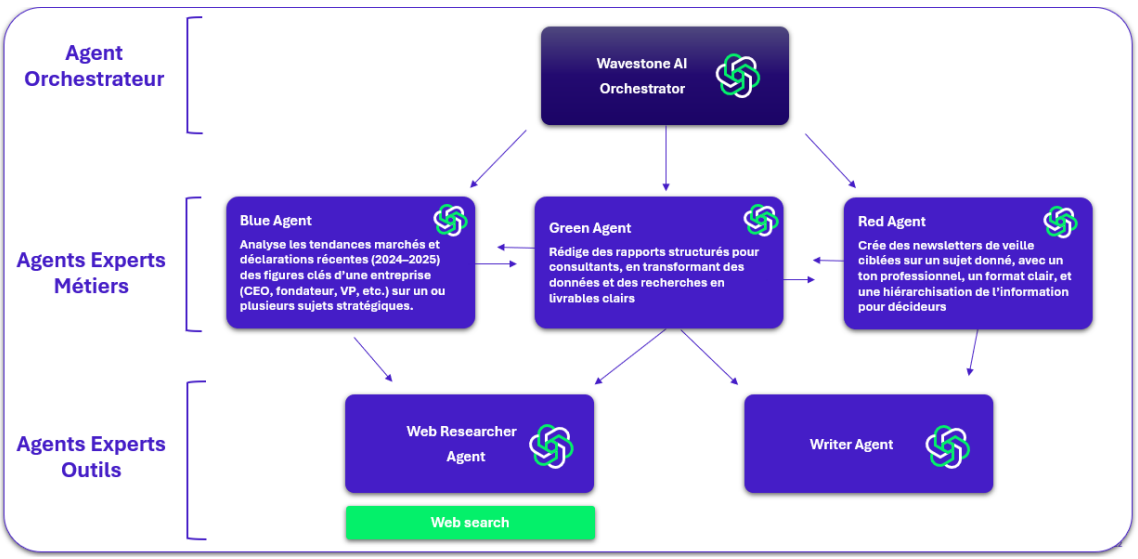
Consider a prototype of an agent-based platform designed to support research and document analysis. The architecture follows a hybrid model, combining a central manager agent for coordination and multiple specialized sub-agents, each responsible for a specific capability: querying internal databases, conducting web research, extracting insights, structuring summaries, and more.

In this setup, the central manager agent receives user requests and breaks them down into specific subtasks. For instance, when a consultant asks for an in-depth market trends analysis, the manager agent identifies the components of the task and delegates them accordingly.

One sub-agent, specialized in Key Opinion Leader (KOL) identification, is tasked with detecting relevant influencers in the target domain. At the same time, another agent focuses on trend analysis, collecting and interpreting current market data. These two agents collaborate by exchanging intermediate results and refining their respective analyses.

Meanwhile, a news monitoring agent tracks the latest developments, providing real-time updates to the others. Each of these agents may, in turn, rely on additional sub-agents—for instance, to conduct online searches or draft detailed reports.

Once the subtasks are completed, the manager agent gathers all partial results, integrates them, and synthesizes a final response to deliver back to the consultant. This modular and cooperative approach allows each agent to contribute its domain expertise while ensuring efficient coordination and a high-quality, unified output.



Agent cooperation protocols (A2A & ACP): toward universal interoperability

As discussed earlier, orchestration can be built around a pyramidal structure of specialized sub-agents. At the base, tool agents execute simple functions—file reading, API calls, classification. Higher up, manager agents delegate tasks to these specialized agents. This model makes it easier to break down complex workflows into reusable components.

But to make such pyramids work at scale, seamless agent-to-agent cooperation becomes essential. That's where emerging inter-agent protocols come in—designed to enable heterogeneous agents to collaborate without friction.
While the Model Context Protocol (MCP) has standardized how agents connect to their environment (tools, data), the next step is enabling direct cooperation between agents themselves.

Two protocols are gaining traction:

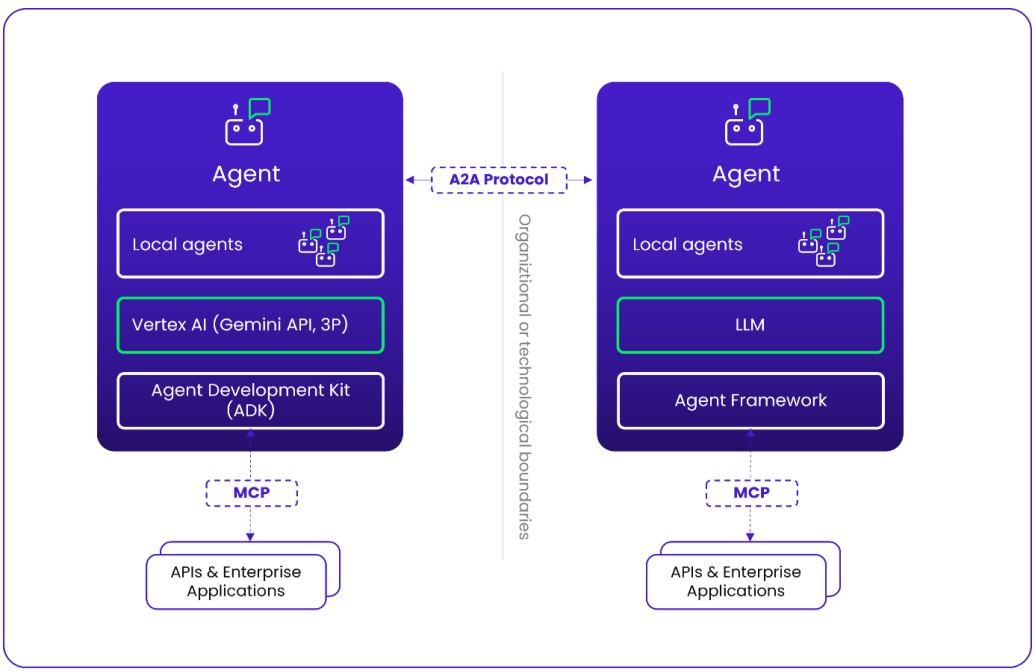
- A2A (Agent-to-Agent) – developed by Google (April 2025)
- ACP (Agent Communication Protocol) – launched by IBM (February 2025)

These protocols aim to standardize how agents discover each other, delegate tasks, track execution, and return results or artifacts. The goal is simple: allow agents developed using different frameworks, running on different platforms, to interact without the need for custom integration.

If MCP handles vertical integration (agent ↔ environment), A2A and ACP enable horizontal coordination (agent ↔ agent). Together, they lay the groundwork for distributed ecosystems—where agents can coordinate, learn from one another, and collectively solve complex tasks.

Concrete use case: recruitment automation

An HR agent initiates a candidate sourcing request. This task is delegated to multiple specialized agents, each querying different databases. The results are aggregated, and another agent takes over to schedule interviews. All of this happens automatically, within a shared workspace, thanks to inter-agent cooperation protocols.



Simplicity and modularity: lessons from Anthropic on agent design

As AI systems grow more autonomous, new architectural challenges are emerging. While most agents currently in production still fall within levels 0 and 1 of agentic intelligence—conversational assistants or smart routers—the long-term goal is to build systems capable of managing their own behavior, as seen in levels 2 through 4.

In that journey, the insights shared by Anthropic are particularly instructive. Having supported many enterprise teams in developing LLM-based agents, their takeaway is clear: the most effective implementations are often the simplest.

Anthropic distinguishes two main families of architectures:

- Agentic workflows: sequences of actions predefined by the developer, with the LLM executing them without initiative. These correspond to levels 0 and 1—agents that act, but only within rigid boundaries.

- Autonomous agents: systems in which the LLM manages the flow—choosing steps, selecting tools, and adapting to the environment. This is the core of levels 2 to 4, involving growing autonomy, dynamic planning, and contextual adaptation.

Anthropic proposes a typology of battle-tested orchestration patterns, which can be layered to gradually increase system complexity:

- **Prompt chaining (Level 1):** breaking down a task into fixed steps, where each output feeds the next input

- **Routing (Level 1–2):** directing requests to specialized processes based on their nature

- **Parallelization (Level 2):** handling subtasks in parallel, or generating multiple variations for comparison

- **Orchestrator–worker (Level 3):** a central agent dynamically plans and delegates tasks to sub-agents, then integrates their results

- **Evaluator–optimizer (Level 3–4):** a feedback loop where one LLM critiques and improves the outputs of another

- **Autonomous agents (Level 4):** systems capable of planning, executing, observing, adjusting—and even generating their own code to expand their capabilities

These patterns show that there is no sudden leap to full autonomy, but rather a step-by-step progression toward agentic intelligence—by assembling simple, controllable building blocks.

"Success in the LLM space isn't about building the most sophisticated system. It's about building the right system for your needs— Anthropic

This incremental approach prevents teams from skipping critical maturity steps. Even reaching level 3, the orchestrator agent, requires strong guardrails: logging, sandboxing, and human-in-the-loop supervision.

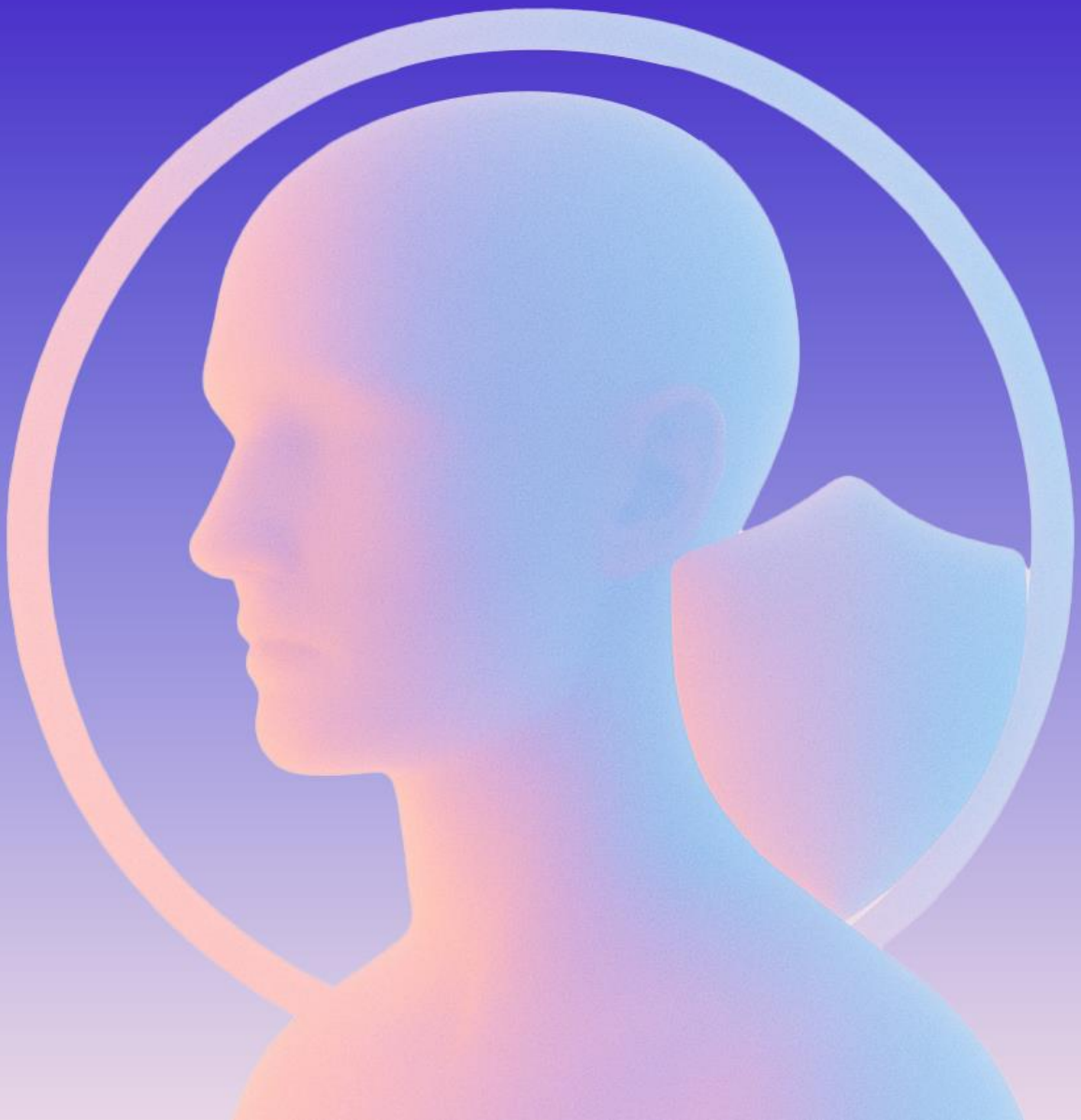
As for level 4 agents—generative and adaptive—their use in production remains extremely limited today.

The key takeaway is simple: only add complexity when it truly improves performance.

This measured approach to agentic design helps keep both costs and environmental impact under control: the more complex and autonomous an agent is, the higher its computational needs—and therefore, its energy consumption.

05. Guardrails and Agent Security

Implementing protective mechanisms and early insights into the security risks of autonomous agents



Chapter 5

Guardrails and Agent Security

As AI agents gain autonomy, their behavior becomes harder to predict, test, and validate.

This rise in agentic capability—while necessary to handle complex tasks—inevitably leads to a loss of control if no oversight mechanisms are in place. The more freedom an agent has in choosing its actions, the more likely it is to interact with sensitive systems, trigger unintended side effects, or make real-world mistakes.

For this reason, deploying an AI agent into production requires a robust system of safeguards, commonly referred to as guardrails. These mechanisms constrain the agent's autonomy, limit its scope of action, filter its outputs, and ensure that its behavior remains aligned with business expectations, regulatory requirements, and security standards.

Why guardrails are necessary

The need for guardrails stems directly from the probabilistic and generative nature of language models. Unlike deterministic code, an AI agent:

- does not follow a single, predictable execution path,

cannot guarantee the accuracy of its outputs,

does not fully "understand" the real-world impact of its actions.

It may hallucinate facts, misinterpret prompts in unexpected ways, or trigger tool sequences that were not anticipated during testing. Moreover, errors may not be immediately visible—poor decisions can produce delayed, subtle, but critical consequences.In this context, guardrails serve a dual purpose. On one hand, they protect the user by enforcing reliability, consistency, and alignment with the intended use. On the other, they protect the organization, by constraining risky behavior, logging decisions, and ensuring full traceability.As shown in PHARE, a benchmark published by French startup Giskard, even the most widely used models can confidently generate incorrect answers, shift their reliability based on the user's tone, or see their accuracy drop sharply when operating under brevity constraints. These risks make strong supervision mechanisms essential for any production-grade AI agent.

What exactly do guardrails control?

Guardrails can target several dimensions of agent behavior. Common examples include:

1. **Response quality** – Avoiding outputs that are inappropriate, irrelevant, or incoherent.
2. **Business rule enforcement** – Preventing or flagging actions that require approval or must meet strict conditions (e.g., refunds above €100, contract cancellations, or alert triggers).;
3. **Data protection** – Ensuring the agent does not expose sensitive information or handle personal data without explicit consent.

Main types of guardrails

Syntactic guardrails

Syntactic guardrails are simple filters applied to the agent's inputs or outputs. These can include regular expressions or validation rules for expected formats, type or structure checks for JSON responses or tool arguments, as well as automatic normalization of values like dates, units, or codes.

These filters ensure that the model's outputs comply with basic formatting constraints before they're used in production. Their implementation typically doesn't require another LLM—basic deterministic rules are often sufficient.

Semantic guardrails

Semantic guardrails focus on the meaning of the agent's responses or actions. They include:

- Toxicity filters, which screen for offensive, discriminatory, hateful, or violent content—even when expressed indirectly or unintentionally. These are essential to maintain safe, compliant communication, particularly in high-exposure or sensitive environments.
- PII (Personally Identifiable Information) detection, which automatically identifies and flags sensitive data such as names, addresses, phone numbers, or user IDs—preventing accidental disclosure.
- Security filters, which validate function arguments before execution (e.g., ensuring a "price" field remains within authorized limits).

These guardrails are often powered by smaller, specialized generative models rather than full-scale LLMs, making them more efficient and easier to operate.

Behavioral or business logic guardrails

Behavioral guardrails restrict or condition specific actions, regardless of whether they are technically valid. They're often implemented directly in system prompts (instructions) or embedded within the orchestration logic.

This layer is essential for enforcing business rules and domain-specific constraints—it ensures that agents don't just act correctly, but appropriately within the operational context.

Dynamic guardrails

Dynamic guardrails are context-aware rules, triggered based on the agent's behavior, task history, or confidence level. For example:

- An agent may be programmed to abort its task if it encounters too many consecutive errors.
- It may pause execution if the user expresses doubt ("Are you sure?", "This doesn't seem right…").

These mechanisms introduce a form of built-in caution, embedding a reflex of prudence into agent behavior and promoting safer, more responsible execution.

Human guardrails

Human oversight remains a vital complement to automated guardrails. In all sensitive situations, agents should be able to:

- Escalate to a human operator in cases of ambiguity or uncertainty.
- Submit critical actions for manual approval before completion.

Security risks: insights from early field reports

As AI agents become more autonomous and start being deployed in real-world environments, early feedback—particularly from Palo Alto Networks in their May 2025 report "AI Agents are Here, So Are the Threats"—highlights a series of critical vulnerabilities that must be anticipated and addressed.

Prompt injection attacks are becoming more sophisticated

One of the most concerning threats is prompt injection, where malicious users manipulate the agent's behavior, extract confidential information, or hijack its tools for unintended purposes. These attacks don't necessarily require deliberate malice—ambiguous or loosely framed prompts can already create exploitable openings.

Vulnerabilities often stem from design flaws, not frameworks

Broadly speaking, most security issues observed in agent deployments are not tied to any specific framework or technology, but rather to poor architectural choices, weak integration practices, or misconfigurations.

Agents connected to poorly secured third-party tools—such as scripts, APIs, or databases—have a vastly increased attack surface. A single weak link can compromise the entire system.

For example, a malicious user could exploit a vulnerability in one of these tools to bypass application-level guardrails, access sensitive data, execute unauthorized code, or redirect the agent from its intended task.

If an agent has unrestricted access to a database management tool, a carefully crafted prompt could lead it to run a destructive command—believing it to be a legitimate user request. The attacker wouldn't need infrastructure access; instead, they would manipulate the agent into executing the command on their behalf, in good faith.

Such an attack—enabled by a lack of tool-level containment—could lead to critical data loss with no immediate alert.

Data leaks with critical consequences

Another major threat is the leakage of sensitive data. AI agents may inadvertently disclose API keys, tokens, or passwords—exposing infrastructure toimpersonation and with minimal execution rights. attacks or privilege escalation.

Agents equipped with code interpreters are also vulnerable to code execution attacks, especially if their execution environments are not properly sandboxed or restricted.

In light of these risks, no single measure is sufficient to ensure safety. What's needed is a systemic security strategy, with multiple layers of control applied throughout the agent's lifecycle. This includes:

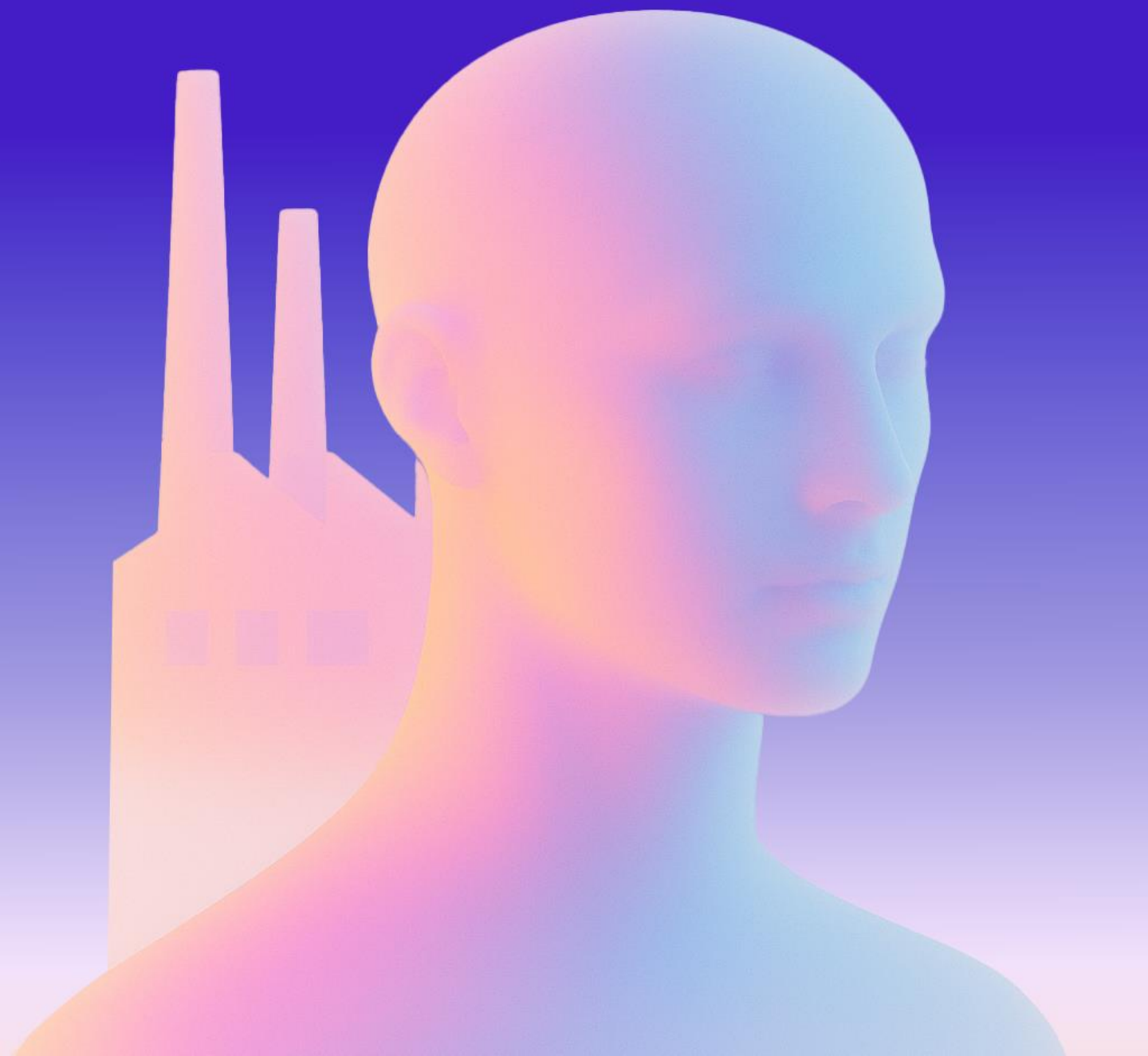
- Strengthening prompt design to prevent out-of-scope queries
- Implementing real-time content filters to block malicious instructions
- Strictly validating the data sent to external tools invoked by the agent
- Running agents in secure, sandboxed environments—with limited network access and minimal execution privileges

As AI agents grow more autonomous, they also become more exposed to complex, evolving vulnerabilities—risks that cannot be mitigated through isolated fixes. Only a risk-driven, cyber-by-design approach, embedded from the earliest design phases and integrating security, governance, and resilience, can effectively prevent misuse and targeted attacks.

This requires close collaboration between cybersecurity and AI teams, as demonstrated by the crisis simulation conducted by ANSSI and Wavestone at the AI Summit, which aimed to raise awareness around these emerging threats.

06. Building an *Agent Factory*

Laying the groundwork to experiment with and deploy agentic AI within organizations



Chapter 6

From experimentation to industrialization: Building an *Agent Factory*

To move from experimentation to controlled industrialization, organizations need to structure their agent development efforts through an Agent Factory. This approach is built on two fundamental pillars:

A shared technical foundation
At its core, the Agent Factory relies on a centralized architecture that organizes all resources required to operate AI agents. This includes:

- **Unified access to AI models**, with supervision, monitoring, and traceability mechanisms;
- **Centralized access to business functions, databases, and connectors,** exposed via a standardized interface—making them easily discoverable and usable by agents without custom development;
- **A single point of reference for all deployed agents**, describing their roles, capabilities, current status, and how they interact with one another.

A dedicated team of Agent Architects: this cross-functional team acts as a center of excellence. It supports business teams in identifying opportunities, designing agent workflows, defining agent roles, and structuring pilots or experiments. The team also plays a key role in governance, risk assessment, and ensuring alignment with the organization's strategic priorities.

Together, these two pillars ensure a gradual and controlled maturity curve for the deployment of agentic systems across the enterprise.

Identifying high-impact use cases and early quick wins

Before scaling up, it's essential to focus on use cases where agentic AI already demonstrates tangible value—automating not only repetitive tasks, but also processes that involve reasoning, planning, and autonomous action. Several families of agents are already emerging as promising candidates for early experimentation.

SWE agents (Software Engineering Agents)

Code generation is a well-established use case where generative AI delivers clear value. Today, the paradigm is shifting—from conversational assistants helping developers write code on request, to semi-autonomous agents capable of operating across entire codebases.

These agents now support developers in a wide range of technical tasks: code generation, unit test creation, documentation, refactoring, vulnerability scanning, and error analysis via log inspection.

Microsoft reports that 30% of its code is currently generated by AI, and projects that figure to rise to 95% by 2030—reinforcing the agent's role as a key contributor to software engineering workflows.

Deep Search agents

Designed to conduct in-depth research across large and heterogeneous information sources—internal documentation, search engines, intranets, legal or scientific databases—these agents can synthesize information, cross-reference sources, and produce structured reports.

They are particularly useful in roles involving strategic intelligence, competitive analysis, audit, and decision support.

Customer support agents

These agents understand the context of user requests—including request history, tone, and the product involved—then access relevant knowledge bases, generate personalized responses, and can even take direct action, such as escalating to technical support, issuing a refund, or routing the case to a qualified human agent.

They reduce the workload of support teams while enhancing the overall customer experience.

Such use cases are particularly well suited to agentic AI, as they combine natural language understanding, business logic, and interactions with third-party systems—a combination that generative AI can now reliably orchestrate at scale.

A centralized infrastructure to orchestrate agents and tools

Deploying agentic systems at scale requires a structured infrastructure, typically built in three foundational layers:

The first step is to centralize access to generative AI models through a Model Registry. This registry lists all models available within the organization. It manages versioning, usage policies, quotas, and security constraints. It is essential for ensuring traceability of LLM usage, monitoring performance, managing inference costs, and can allow agents to dynamically select the most appropriate model for a given task.

The second step is the implementation of a Resource Registry, which centralizes all technical resources (APIs, databases, functions, connectors). Thanks to the MCP protocol, agents can dynamically discover these resources, inspect their schemas, and connect to them without specific implementation. The Resource Registry also plays a key role in governance: it defines access rights, authentication, traceability, and security protocols for critical systems.

The third step is to deploy an Agent Registry. This acts as a directory of agents: listing those available, their roles, statuses, preferences, technical capabilities, and collaboration modalities. This registry enables the activation of coordinated multi-agent environments, leveraging protocols such as A2A or ACP. For example, a manager agent can dynamically delegate a task to a remote agent, or two agents can negotiate how best to divide a complex workload.

Equipping teams to start experimenting

Designing an effective AI agent involves more than just connecting an LLM to a few tools. It requires full orchestration: context awareness, dynamic planning, action selection, error handling, and continuous adaptation.

Agent frameworks play a central role in this effort. They provide:

- Native handling of the perception → action → feedback loop
- Connectors to predefined tools
- memory and logging capabilities
- Evaluation and result interpretation mechanisms

For simple or exploratory use cases, a manual implementation or low-code tools like Langflow or Copilot Studio may suffice. But for critical operations, distributed systems, or sensitive use cases, a robust framework becomes essential. The choice of framework will depend on the organization's tech ecosystem, the required level of customization, and the deployment strategy (cloud, on-premise, edge, etc.).

07. Conclusion

Toward a thoughtful and effective approach to agentic AI

Conclusion

AI agents represent a shift in automation: capable of understanding, planning, and acting, they pave the way for more adaptive and autonomous systems — provided that their governance, technical integration, and reliability limits are rigorously managed.

It is in this context that Wavestone's AI Practice offering truly comes into its own. Aware of the technical, human, and organizational challenges tied to the rise of agent-based systems, we have developed a structured approach to guide companies through their transformation. This approach covers the entire lifecycle of an AI agent — from defining robust governance, to identifying high-impact use cases, through to experimentation, industrialization, and secure production deployment.

Our ambition is clear: to harness agent-based systems in service of business operations with rigor and clarity — unlocking their full potential without compromising on control, data security, or system stability.

Authors

Paul BARBASTE

Senior Consultant AI

AI Researcher, Head of the AI & Neuroscience Program at École Polytechnique and HEC

Wavestone AI Experts

Chadi HANTOUCHE

Partner AI

Stéphan MIR

Associate Partner AI

Julien FLOCH

Associate Partner AI

Acknowledgments

Marie LANGÉ
Senior Manager AI

Badr ACHOUR
Manager AI

Tom WILTBERGER
Consultant Senior AI

Clement PEPONNET
Consultant Senior AI

About Wavestone

Wavestone is a consulting powerhouse, dedicated to supporting strategic transformations of businesses and organizations in a world that is undergoing unprecedented change, with the ambition to create positive and long-lasting impacts for all its stakeholders.

Drawing on more than 5,500 employees in 17 countries across Europe, North America and Asia, the firm offers a 360° portfolio of high-value consulting services, combining seamlessly first-class sector expertise with a wide range of cross-industry capabilities.

Wavestone is listed on Euronext Paris and recognized as a Great Place to Work®.

www.wavestone.com