



IALAB

AI academy
by doinGlobal

Juan Gustavo **CORVALÁN**, Mariana **SANCHEZ**
CAPARRÓS, Carolina Melanie **MARTÍN**



GOVERNING AGENT AI

COLABORATORS: LOLA RAMOS PEREYRA, GISEL
ALVARADO

AUTHORSHIP



JUAN GUSTAVO CORVALÁN

Doctor in Legal Sciences and Master in AI. Speaker at UN, OAS, MIT and Google Talks. Director of the AI Laboratory at UBA (Law). Director of the 4-volume Treatise on AI and Law. Author and co-author of dozens of research papers and publications on AI. Co-creator of Prometea, the first predictive system in the Justice system of Latin America (2017).



MARIANA SÁNCHEZ CAPARRÓS

Doctor in Legal Sciences and Master in Administrative Law. Deputy Director of UBA IALAB, where she leads R&D projects in Artificial Intelligence, innovation and digital transformation, specializing in the ethical application of AI in the public sector.



CAROLINA MELANIE MARTÍN

Researcher at UBA IALAB and student of Artificial Intelligence Engineering. She specializes in the design of intelligent agents and automation systems applied to legal analysis, complex information management and research. Her career integrates experience in data analysis and criminal investigation with the development of Artificial Intelligence-based solutions.

COLLABORATORS



LOLA RAMOS PEREYRA



GISEL ALVARADO

TABLE OF CONTENTS

PRELIMINARY NOTE: SHIFTING SANDS AND TERMINOLOGICAL DECISIONS	01
A DISTINCTION THIS DOCUMENT WILL USE SYSTEMATICALLY: AGENCY AND AUTONOMY	02
01. INTRODUCTION	03
WHAT WE FIND TODAY IN THE AGENTIC AI ECOSYSTEM	
READING COORDINATES: TWO PARALLEL AND COMPLEMENTARY DIMENSIONS	
02. FROM THE SYSTEM THAT EXECUTES TO THE SYSTEM THAT DECIDES: THREE LEVELS OF AUTOMATION	08
AUTOMATION WITHOUT AI	
TASKS AND WORKFLOWS INTEGRATING GENERATIVE AI	
AI AGENTS: PROACTIVE AI BY DESIGN	
03. FUNCTIONAL COMPONENTS OF AI AGENTS	13
THE REASONING ENGINE: THE LLM	
THE OBJECTIVES: THE MISSION AND LIMITS SET IN THE DESIGN	
THE INPUT: INFORMATION AND CONTEXT OF EACH EXECUTION	
THE TOOLS: ARMS AND SENSES OF THE AGENT IN THE DIGITAL WORLD	
THE OUTPUT: FROM TEXT TO IMPUTABLE ACT	
04. THE THREE LAYERS OF TECHNICAL DECISION	21
LAYER 1: DEPLOYMENT INFRASTRUCTURE	
LAYER 2: WHAT THE AGENT IS BUILT WITH	
LAYER 3: HOW THE AGENT CONNECTS WITH THE REST OF THE ECOSYSTEM	
05. TYPES OF AI AGENTS: PACKAGED, HYBRID AND DESIGNED	26
PACKAGED AGENTS: ONE-CLICK AGENTS	
THE HYBRID ZONE: WHEN THE USER CONFIGURES AGENTS WITHOUT PROGRAMMING	
DESIGNED AGENTS: FULL CONTROL OVER THE ARCHITECTURE	
TWO SPECIALIZED CATEGORIES: COMPUTER USE AND CODING AGENTS	
06. PLATFORMS FOR BUILDING AI AUTOMATIONS, MANAGED PLATFORMS AND FRAMEWORKS	31
AI AUTOMATION PLATFORMS: N8N, MAKE, ZAPIER	
MANAGED PLATFORMS FOR AGENT DESIGN	
CODE FRAMEWORKS FOR AGENT DESIGN	
MULTI-AGENT ORCHESTRATION FRAMEWORKS	
AUTONOMOUS AGENT FRAMEWORKS WITH ENVIRONMENT ACCESS	
AGENTIC BROWSERS: THE EMERGING CATEGORY	
EVALUATION AND OBSERVABILITY FRAMEWORKS	
CLOSING: A SPECTRUM, NOT RIGID CATEGORIES	

07. GOVERNANCE OF THE AGENTIC AI ECOSYSTEM

42

INTRODUCTION: GOVERNING AN ECOSYSTEM, NOT A TOOL OR A PLATFORM

CLASSICAL INSTITUTIONAL VERTICAL

TRACEABILITY

AUDITABILITY

TRANSPARENCY, EXPLAINABILITY AND TWO DISTINCT OPACITIES

PERSISTENCE AND THE RISKS OF MEMORY IN AGENTS

THE AGENT RELIABILITY GAP AS A STRUCTURAL CAUTION

ATTACK SURFACE: TRAPS FOR AI AGENTS

OPERATIONAL RISK MANAGEMENT: THE CASE OF THE USER FACING HYBRID TOOLS

COGNITIVE VERTICAL: GOVERNING ATROPHY, NOT JUST LEGALITY

THE FOUR PARADOXES OF AUTOMATION APPLIED TO THE AGENTIC ECOSYSTEM

THE COMPLETE PRODUCTIVITY CYCLE: FIVE PHASES

SUPERVISION MODELS: HUMAN FIRST, IN-THE-LOOP, ON-THE-LOOP

BEYOND THE INDIVIDUAL AGENT: COLLECTIVE PROCESSES AND MASSIVE HARMS

08. CLOSING

56

PRELIMINARY NOTE

SHIFTING SANDS AND TERMINOLOGICAL DECISIONS

A necessary warning before we begin. The agentic AI ecosystem moves at a speed that defies any attempt at stable mapping. The platforms we describe in this material permanently incorporate new functionalities, modify their architectures or redefine their access models. This is a condition of the object of study: we are working on shifting sands.

That same speed of evolution makes something difficult that in other fields is taken for granted: terminological consensus. What exactly is an "assistant," what is an "agent," where does "agentic" begin, what distinguishes an automated flow with AI from a proper agent, are questions that industry, academia and professional practice answer today in different and sometimes contradictory ways.

The providers themselves use these terms with commercial criteria that do not always coincide with technical definitions, and the technical definitions do not always coincide with each other. At the time of writing this book, we found no stabilized vocabulary. In fact, the "moving target" phenomenon is intensifying.

Faced with this situation, this work adopts some operational definitions that will remain consistent throughout the entire text, but which may not coincide with those of other texts.

In this document we call *generative AI-based assistant* the minimum unit of the agentic ecosystem: platforms like Claude, ChatGPT or Gemini that already exhibit agency by design but operate with limited autonomy, within the conversation windows or workspaces available to users on the platforms.

At the other end of the spectrum we will place *AI agents* proper, as systems in which multiple specialized agents, mounted on a software engineering harness, coordinate among themselves to execute complete processes with minimal human intervention or even without human intervention.

Between both poles there is a spectrum of possibilities, a continuum that is not necessarily a leap. That is, each tool we will analyze is located at some point on that spectrum, with greater or lesser degree of agency and autonomy, with greater or lesser need for technical knowledge to design and use AI assistants and agents.

A DISTINCTION THIS DOCUMENT WILL USE SYSTEMATICALLY: AGENCY AND AUTONOMY

Before entering the introduction, it is worth clarifying two terms that we tend to use interchangeably and that in the rest of this material we will distinguish.

Agency

Agency is the capacity to act upon the environment: invoking tools, modifying files, querying databases, sending messages. Any assistant with *tool use* enabled has agency.

Autonomy

Autonomy is the degree to which that action occurs without direct human intervention. An assistant may have high agency because it has many tools available, but low autonomy because each use requires a prompt or explicit action from the user. An agent may have moderate agency because it has a limited repertoire of tools, but high autonomy because it can execute long chains of actions across various tasks without consulting the user.

The distinction is important because governance decisions vary depending on the second axis, not so much the first: what makes a control framework relevant is not how many things a system can do, but how many it does before requesting validation, auditing, or oversight.

In summary

This book does not offer an exhaustive inventory or a definitive map, but rather a conceptual framework: a way of thinking about the agentic ecosystem, the tools that coexist within it, their design components, and their governance requirements, considering the state of the art as of early May 2026.

01. INTRODUCTION

WHAT WE FIND TODAY IN THE AGENTIC AI ECOSYSTEM

This material seeks to present the state of the art of agentic AI based on the two publications we made about AI agents at IALAB: the ecosystem research, the systematic exploration of tools and platforms, and the solutions we have designed and prototyped in concrete organizational contexts.

What follows is not a theoretical exercise, but the result of that accumulated practice, organized as a conceptual framework for those who need to understand, evaluate and govern this type of technology. Let us begin by addressing three major phenomena.

1 — First

In the agentic AI ecosystem, different types of solutions coexist that differ fundamentally in their degree of agency and autonomy, as well as in the possibility of intervening in the design of their architecture. The two extremes of that spectrum are the *generative AI-based assistant* and the *AI agent*.

Up to this point, most of us have used *generative AI-based assistants*. This is the most widespread form of interaction with generative artificial intelligence tools for individual use such as ChatGPT, Claude or Gemini, among others.

These platforms are part of the agentic AI ecosystem, although they operate at the lower autonomy end of the spectrum. They are capable of interpreting the user's intention to suggest next steps, activating on their own code tools, internet access or other functionalities. Even so, in most interactions they receive a query, process it probabilistically and return a response within the same turn, driven by human intervention.

Moving toward the higher autonomy end of that ecosystem, that is, toward *autonomous AI agents*, implies a reordering of that relationship. It is no longer just about instructing a model behind the platform to produce a response, but about delegating to a system the execution of one or several objectives and tasks. In this new framework, the model will perceive the environment, plan a sequence of actions, select the appropriate tools and adjust the course of action based on the feedback it receives during execution.

This transformation is not merely technical. The role of the user is reconfigured, shifting from executor of instructions to designer, curator, auditor, supervisor and director of AI agents and autonomous flows of agentic processes. This aligns with the frameworks presented during 2026 by companies such as Anthropic, Google and OpenAI.

2 — Second

AI agents do not solve all use cases, and they are not always the most efficient solution in terms of results, costs and risks associated with their design and operation. Choosing when a problem requires an agent and when it is better solved with AI-powered automation is, in itself, a governance decision.

The forms of automation that preceded the mass adoption of AI are deterministic and, in many cases, more economical and easier to audit. They even remain the best alternative depending on the tasks or objectives sought. These modalities do not disappear with the arrival of agentic AI: they coexist with it and also complement it.

In particular, workflows that integrate generative AI within deterministic architectures, what in the following section we will call the second level of automation. These solve an enormous number of concrete problems at significantly lower costs than those of an agent, with greater predictability in their results and less deployment complexity.

3 — Third

Managing the gap between pilots and implementation.

To all this is added a technical caution worth anticipating from the introduction: the gap between demos and production, what the technical literature calls the agent reliability gap. As of April 2026, consolidated agentic capability benchmarks such as GAIA, SWE-bench Verified, AgentBench and WebArena show real task resolution rates ranging between 30% and 70% depending on the domain, even in the best commercially available systems.

The industry communicates agents as if they were reliable workers; experimental evidence supports a more nuanced picture. This is key if we want to think about a governance framework: the structural reliability of the system must be incorporated as a design variable and not as an exception to be corrected.

READING COORDINATES: TWO PARALLEL AND COMPLEMENTARY DIMENSIONS

Throughout this document we will implicitly work on two axes that are important to differentiate.

TASK AI	PROCESS AI
Automates specific tasks	Automates complete processes
User (Decentralized)	Organization (Centralized)
User designs and controls	Organization designs and controls

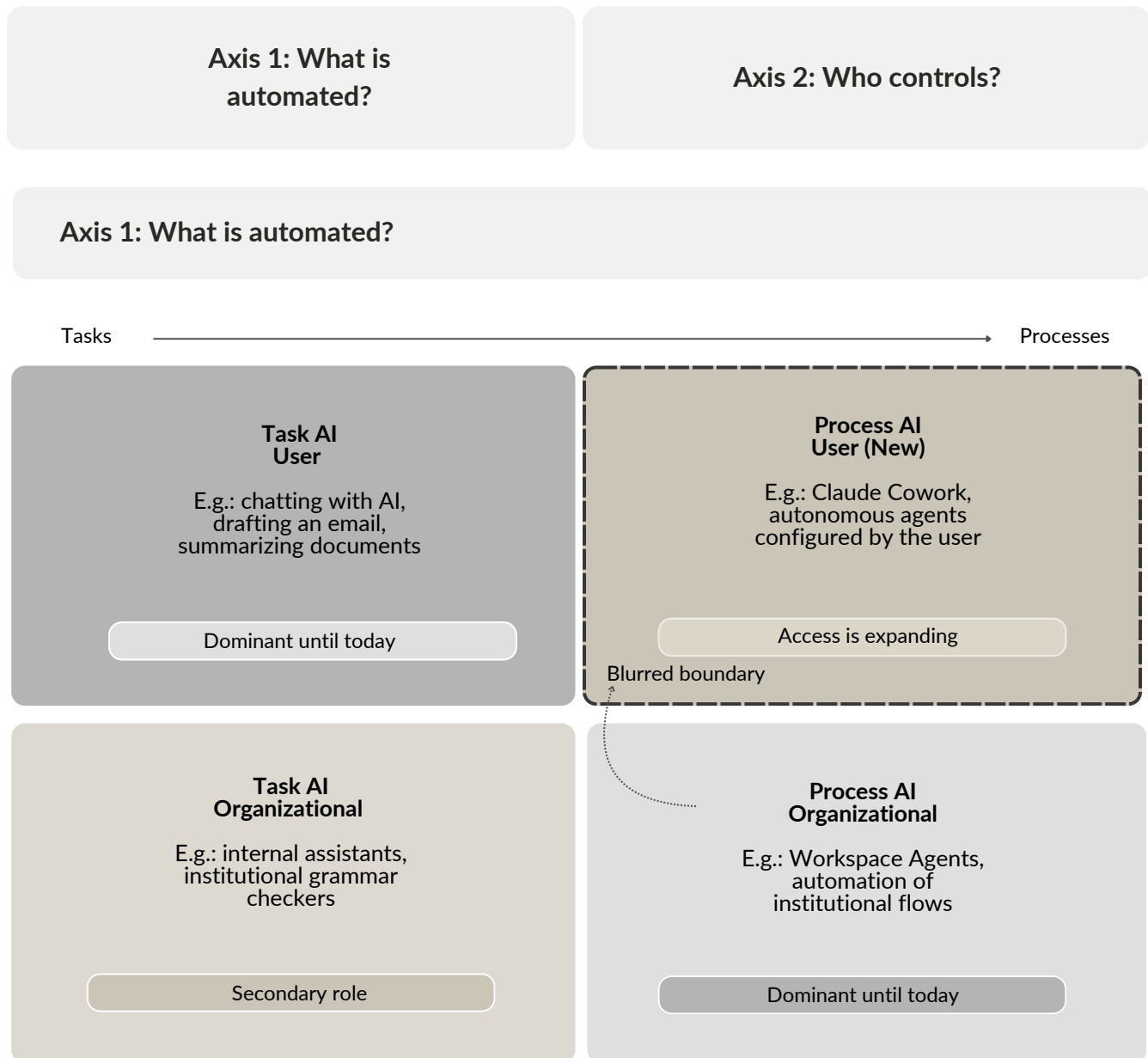
Until recently, task AI was predominantly user-driven and process AI was predominantly organization-driven. That correspondence is dissolving. Tools like Claude Cowork allow an individual user to configure process flows with varying degrees of autonomy. And managed agent platforms like ChatGPT's Workspace Agents, which previously required organizational infrastructure, are also becoming accessible from the end-user paradigm.

This means that process AI is no longer necessarily centralized, and that a user can operate with process-level agentic tools without an architectural decision from the organization. This democratizes access and amplifies both opportunities and challenges and risks. Here it will be central to analyze the governance issues of the agentic ecosystem.

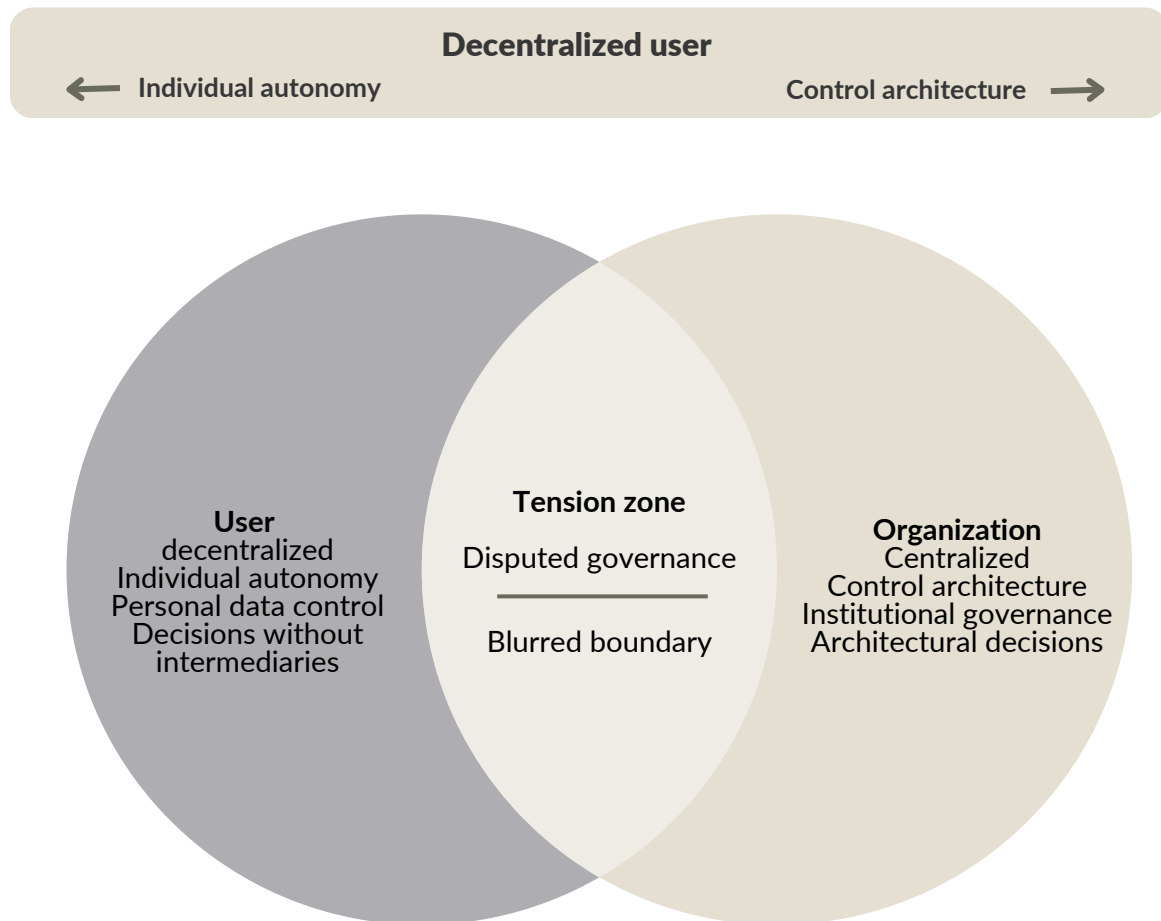
We reiterate the dual dimension: *task AI* linked to what is automated, and process AI. And on the other hand, who designs and controls the automation and use of AI and agents: the user in a decentralized manner or the organization in a centralized manner.

On this dual dimension we propose a **taxonomy of three levels of automation** (section 2); the **functional components that define an agent**, including the often-forgotten dimension of *output* as imputable action (section 3); the three layers of technical decision, infrastructure, abstraction and interoperability protocols, that condition any deployment (section 4); the types of agents according to the degree of user control over the architecture (section 5); the platforms and frameworks available to build them (section 6); and a **governance framework** articulated in two verticals, the classical institutional one (traceability, auditability, transparency, persistence) and the cognitive one (sedentarism, paradoxes of automation, complete productivity cycle), which closes the material (section 7).

THE TWO KEY DIMENSIONS OF THE AGENTIC ECOSYSTEM



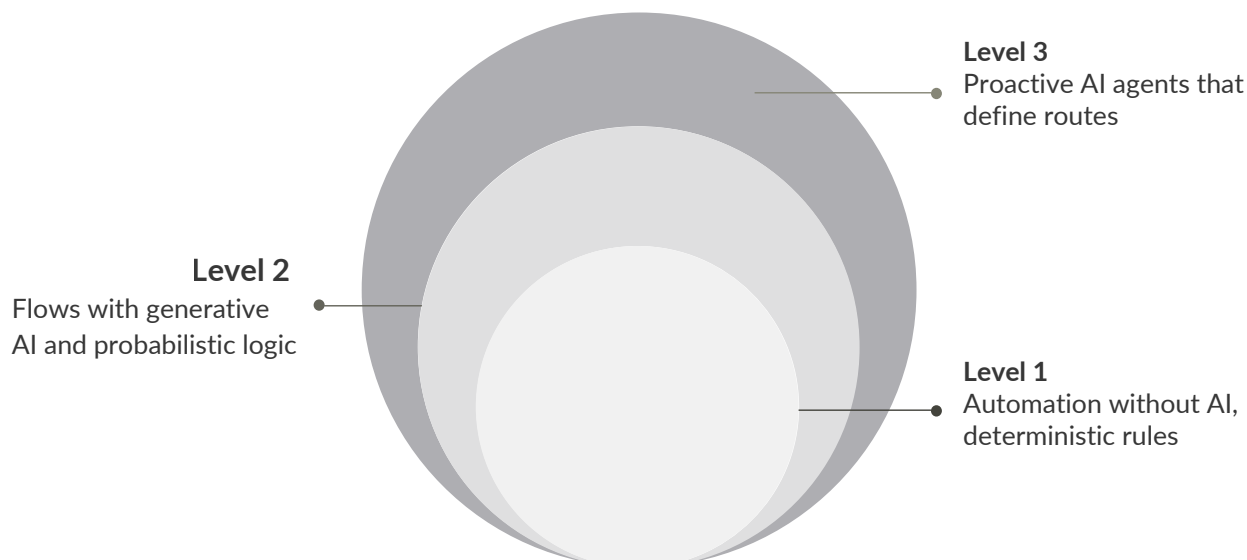
Axis 2: Who controls?



02. FROM THE SYSTEM THAT EXECUTES TO THE SYSTEM THAT DECIDES:

THREE LEVELS OF AUTOMATION FACING AGENTIC AI

The question of what distinguishes an AI agent from other forms of process automation is best answered by observing the complete continuum. Not all automations are equal nor do they aspire to the same thing. We can distinguish **three levels**, each with a differentiated field of application.



Each level represents a differentiated field of application, with different degrees of flexibility, autonomy and governance complexity. The continuum does not imply that one level is superior to another: each responds to specific needs and may be the best option depending on the context.

AUTOMATION WITHOUT AI

Automation without AI responds to a deterministic logic. It comprises the set of computer systems designed to execute tasks automatically following predefined rules and sequences, built on Boolean logic and deterministic principles.

The system's behavior can be expressed through conditional structures of the type *if condition X is verified, then execute action Y; otherwise, execute Z*. The development team defines in advance the set of possible system behaviors, which remains fixed throughout its entire operation.

THREE FEATURES OF AUTOMATION WITHOUT AI

Absence of inference

The system does not interpret, reason, or construct representations of the problem; it merely evaluates pre-established logical conditions and executes the flow branches that those conditions activate.

Determinism

Given the same inputs and the same internal state, the system invariably produces the same result, guaranteeing full predictability and reproducibility.

Operational closure

The system can only act within the universe of scenarios foreseen by those who programmed it; any unforeseen event produces an error, a halt, or inappropriate behavior.

EXAMPLES OF AUTOMATION WITHOUT AI

Document management system

Upon detecting the expiration of a deadline stored in a database, it triggers an automatic notification to the person responsible for the file.

Web form with validation

It prevents submission when any of the fields marked as mandatory remains empty.

Spreadsheet with formulas

Automatically updates a sum or average each time the input cells are modified.

Automatic filing system

Categorizes incoming documents into predefined folders based on exact keyword matches.

Integration trigger

Sends a notification to a messaging channel each time a record is created in a linked application.

STRENGTHS AND LIMITATIONS OF AUTOMATION WITHOUT AI

Strengths	The distinctive strength of pure automation is reliability . By operating on explicit and deterministic rules, the system's behavior is entirely auditable: it can be explained step by step, validated before deployment, and monitored through exhaustive testing. This makes it particularly appropriate for structured, high-volume processes with low error tolerance.
Limitations	The downside is its rigidity . The system cannot adapt to atypical inputs, exceptions, or situations not foreseen in the design phase; any relevant variation from the modeled scenario requires an explicit code modification. There is no flexibility, dynamism, or decision-making capacity in the face of ambiguity.

In summary, **automation without AI operates with the logic of a mechanical clock**: each wheel and each gear fulfills an exact function at an exact moment. It is precise and reliable as long as operating conditions match those foreseen by the clockmaker. Faced with an unforeseen event, it does not improvise: it stops.

TASKS AND WORKFLOWS INTEGRATING GENERATIVE AI

Technically, the flow maintains a general deterministic structure designed by people.

It is that structure, inherited from automation without AI, that orchestrates the process and decides at which points AI is activated to solve tasks requiring comprehension, interpretation, or natural language generation. These are solutions where *"language models and tools are orchestrated through code paths predefined by the development team"* and that, therefore, *"offer predictability and coherence for well-defined tasks"*.

The second level of automation is the meeting point between the deterministic logic inherited from traditional automation and the probabilistic capability of generative AI. In this model, the architecture combines the logic that governs the general structure of the flow with the large language models (LLMs) that are invoked at specific moments.

This combination notably expands the spectrum of automatable tasks: no longer are only structured inputs and exact matches processed, but also free text, unstructured documents, multimodal content, and even contextual nuances that classical logic could not capture.

In this second level, the complete flow is still designed by people. Those who configure the automation determine in advance at what point the model is consulted, what information is provided to it, under what instructions it operates, what is done with its output, and how subsequent steps are chained. The model does not decide what to do or when to do it: it executes specific tasks such as classifying, summarizing, drafting, or extracting within a route that is given to it, not one it chooses. The architecture and general logic of the flow depend on human design.

EXAMPLES OF FLOWS WITH GENERATIVE AI

01	02	03
Automatic summarization of lengthy documents A flow receives a PDF attachment, extracts its text, requests an executive summary from a generative model in the desired language and format, and distributes the result by email to stakeholders.	Data extraction from unstructured documents A flow that receives contracts, invoices, or other documents, uses a generative model to identify key entities (parties, dates, amounts, relevant clauses) and structures the information in a database for subsequent analysis or comparison.	Automatic classification of incoming queries A flow that receives messages in natural language, uses the model to categorize the type of request and route it to the corresponding area, person, or sub-queue according to preconfigured criteria.

STRENGTHS AND LIMITATIONS OF FLOWS WITH GENERATIVE AI

Strengths

The decisive advantage over the first level is the ability to process natural language and unstructured content, which multiplies automatable scenarios and allows a breadth of variation in inputs that classical automation would not tolerate. However, the architecture inherits two classes of limitations.

Limitations

Relative rigidity

If the received case does not fit the expected sequence, the system does not reorganize the process nor improvise alternatives.

Risks inherent to generative models

Hallucinations, biases inherited from the training corpus, probabilistic variability in outputs, and knowledge limited to a cutoff date. These risks require human supervision checkpoints at critical moments in the flow, what the literature calls *human in the loop*. They also require careful design of the *prompts* and instructions that guide the model at each step.

AI AGENTS: PROACTIVE AI BY DESIGN

In the two previous levels, the complete flow is designed by people and the technology executes it. In an AI agent, that relationship is inverted: humans define the objective and the boundaries; the AI plans and executes the process it considers most appropriate to achieve it.

The defining trait is not the presence of generative AI—which already existed at the second level—but that the agent **constructs the route at runtime from an objective, rather than following a pre-established route.**

The difference can be condensed into an operational question: **who designs the path?**

Pure automation and flows with Generative AI

Humans design the complete process.

Agentic AI

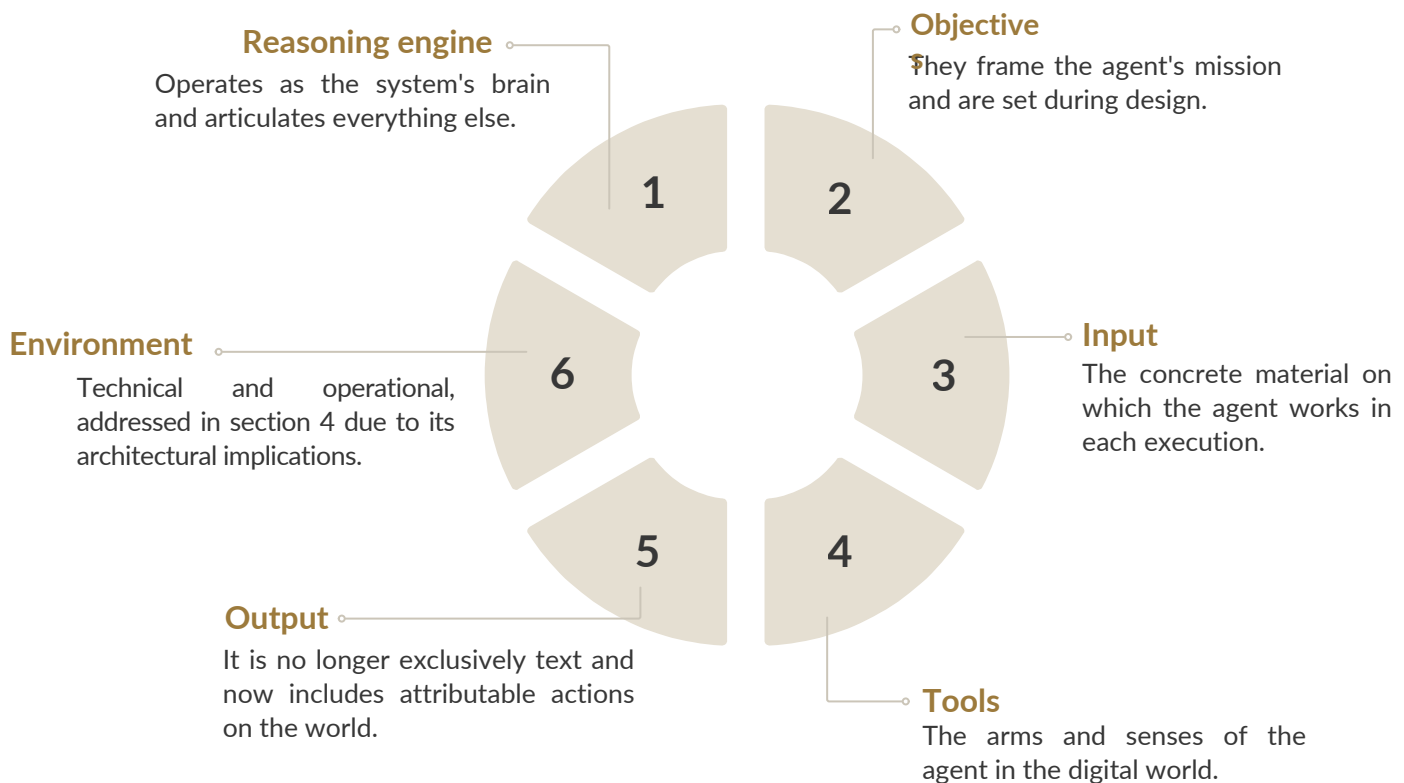
Humans define the objective and, eventually, the boundaries. The agent(s) can design and execute the tasks and complete sequences of one or several processes. They can even dynamically adjust sub-objectives or decisions based on the information they collect and the partial results they obtain.

This inversion is what justifies a dedicated taxonomy of functional components (section 3), a specific infrastructure decision layer (section 4), and a differentiated governance regime (section 7). It is also what makes relevant the caution mentioned in the introduction: when the system constructs its own route, auditing cannot be limited to the final output; it must reconstruct the trajectory.

03. FUNCTIONAL COMPONENTS OF AI AGENTS

An AI agent is part of a **modular ecosystem** composed of interconnected pieces that allow it to perceive, reason, remember, and act. In essence, it is a system dedicated to context management: a continuous cycle of information gathering, model activation, observation of results, and context reconstruction for the next step.

To organize that architecture, we will distinguish six components:



These are not isolated pieces, but dimensions of the same design. The question that should guide the reading of each one is the same:

what decisions am I enabling or restricting when I configure this component in such or such a way?

THE REASONING ENGINE: THE LLM

At the core of any agent there is a large language model (LLM) that acts as its brain: the processing unit that receives available information, simulates reasoning, makes decisions, and coordinates the use of the other components.

These are typically the same models already known from the use of generative AI-based assistants (OpenAI's GPT family, Google's Gemini, Anthropic's Claude, or local models like Qwen, Llama, Mistral, among others), although in the agentic context their function transcends content generation: the model becomes a director that plans a strategy, selects which tools to invoke at each step, and evaluates the intermediate results it obtains.

LLM REASONING PATTERNS

To achieve that coordination, the model relies on planning techniques that can be grouped into two major patterns, known in the technical literature:

ReAct (Reason + Act)

Yao et al., 2022. The agent alternates reasoning and action at each step: it thinks, acts, observes the result, and thinks again. It is the dominant pattern in general-purpose agents like Claude Agent SDK and OpenAI Agents SDK. This logic is more robust in changing environments because the agent re-estimates its progress after each action.

Plan and execute

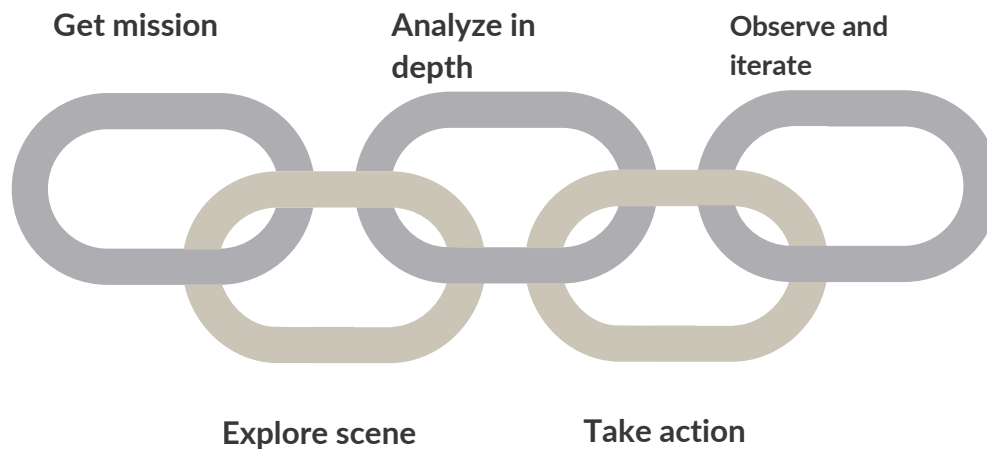
The agent first produces a complete plan and then executes it step by step, eventually with a lighter sub-model. It is efficient for predictable tasks but less resilient to unforeseen events.

Chain of thought (Chain-of-Thought)

It allows decomposing a complex and ambiguous objective into a sequence of smaller, logical, and manageable subtasks, executing them in stages, and correcting course when errors are detected.

THE AGENT'S OPERATIONAL CYCLE

That decomposition capability unfolds, in practice, through an iterative cycle that the technical literature describes in five phases:



This cycle of **"think, act, observe"** repeats, managed by the orchestration layer and executed by the tools, until the objective is completed or until the system determines it needs human intervention. This mode of operation explains why an agent can sustain a multi-step workflow without the user having to indicate each one: the model decomposes the objective by itself and reorders the plan as it progresses.

AGENT MEMORY

The engine is complemented by two modules that allow it to sustain coherence throughout the process and between successive processes:

Short-term memory (or working)

Tracks the agent's own "line of thought" and the actions it has just performed within the current session, preventing it from losing the thread of its reasoning mid-flow. It functions as the agent's active *notepad*, maintaining the history of the current conversation.

Long-term memory (or persistent)

Retains information between sessions (organizational preferences, already consolidated writing criteria, positions taken in analogous cases) and enables the so-called "Netflix mode": the agent "knows" the user and their institutional context and builds a continuous collaboration relationship without needing to repeat the framing in each interaction.

OBJECTIVES: THE MISSION AND BOUNDARIES SET IN THE DESIGN

The interaction between language models and users at the "user AI" level is linked to synchrony in the interaction or, as the case may be, to the editing of prompts that can be housed in projects or areas to work within the platform ecosystem.

In agentic logic, one must design objectives and then think about how the agent will carry them out. It is critical to determine means, criteria it must respect, and the limits it cannot exceed. That configuration is embedded in the agent's system instructions and operates in all subsequent executions.

By analogy, the agent's objectives are to its design what the *resolution creating a position* or the *internal regulations of a function* are to a public office. They are not rewritten for each case file; rather, they permanently establish what is expected of the position, what competencies are attributed to it, and what conduct is prohibited.

In operational terms, correctly defining the agentic mandate involves anticipating:

The substantive objective

The expected outcome, formulated as concretely as possible. For example: *"assist in drafting opinions in administrative litigation cases, based on the digital case file"*.

Sub-objectives or intermediate milestones

Checkpoints where the process must stop, report, or require human validation. For example: *"when identifying applicable regulations, record sources; upon completing the draft, submit it for review by the decision-maker before any further action"*.

Negative limits

Actions the agent is prohibited from executing, even if it considers them functional to the objective. For example: *"do not consult sources external to those loaded in the database"*.

This is one of the most critical design components. In agentic AI, the maxim that the end does not justify the means takes on special relevance: an agent that achieves its objective by violating confidentiality duties, accessing systems for which it has no authorization, or engaging in practices contrary to current regulations is not an *effective* agent but a *poorly designed* one. The precise delimitation of the mandate with its milestones and prohibitions is, in institutional contexts, the first and most important line of defense against that risk.

THE INPUT: INFORMATION AND CONTEXT FOR EACH EXECUTION

If the objectives are the permanent framework set in the design, the input is the concrete matter on which the agent works in each execution. It is everything that enters the system in real time and from which the agent builds a representation of the specific problem it must solve: the user's request, attached documents, and data the agent itself collects as it progresses.

In its most basic form, the *input* coincides with the *prompt* the user formulates in natural language, possibly accompanied by documents, links, or supplementary instructions they choose to attach. An example within an AI agent that drafts opinions would be: "prepare a draft for case file XXX prioritizing the statute of limitations compliance issue." That request, up to this point, does not structurally differ from what a generative AI-based assistant would receive.

In an agentic ecosystem, the input is composed of three main sources:

The user's prompt and attachments

The specific request that initiates the run and the documents or data the user provides along with it.

Dynamic queries to external tools

Web searches, API queries, database reads, or document repository lookups that the agent itself decides to perform on the fly, based on what the objective and request suggest.

Retrieval from memory

Information already known to the system such as organizational preferences, institutional criteria, or decisions from analogous cases, which the engine automatically retrieves from persistent memory to contextualize the current execution.

This shift modifies the type of decisions that the design must anticipate. The quality of the user's *prompt* remains important. But now additional questions arise that move to the architectural level:

What sources is the agent permitted to access when expanding the input? With what credentials? With what degree of currency? With what traceability of queries made? What is stored in persistent memory, for how long, and with what review mechanisms?

Each of these definitions conditions the quality and legality of all subsequent outputs. An agent that does not access up-to-date information will operate on obsolete assumptions; an agent with excessive access to sensitive information may compromise both the accuracy of its responses and institutional confidentiality; a poorly delimited persistent memory may itself become a repository of personal data whose management must meet normatively required protection standards.

There is an aspect that is often overlooked: many of these questions are not answered in the user's prompt, but in the agent's design. The objectives set in the design define which sources are legitimate and under what conditions; the input formulated during execution decides what is specifically consulted within that authorized universe. Neither of the two components can be explained without the other.

TOOLS: THE AGENT'S ARMS AND SENSES IN THE DIGITAL WORLD

However powerful it may be, the reasoning engine, by itself, is an isolated system that only knows the data it was trained on, up to a certain cutoff date, and can only generate text. For it to become a digital operator capable of intervening in a real workflow, it needs extensions that allow it to interact with the environment. Those extensions are the tools.

Technically, tools are connections generally mediated by APIs that grant the agent extended operational capabilities. Through them, the agent can:

→ Browse the internet and obtain real-time updated information.

- Query external databases.
- Read, analyze, and generate attachments in multiple formats (text documents, PDFs, spreadsheets, code).
- Execute code in a controlled environment.
- Interact with other applications (email, calendars, electronic signature platforms, notification systems) to trigger concrete actions.

Added to these capabilities are **native multimodal capabilities**, which function as a form of *holistic perception* of the digital environment: the agent not only reads text but can also interpret visual content or transcribe an audio file.

The set of tools does not come bundled with the language model. It is an explicit design decision for the agent. Which tools are enabled, with what permissions, with what credentials, with what per-query limits, with what audit logs: all of these are definitions that, in an institutional context, must be considered in dialogue with the objectives and their normative limits or ethical or cultural constraints.

THE OUTPUT: FROM TEXT TO ATTRIBUTABLE ACT

In a generative AI-based assistant, the output coincides with the textual response that appears on screen.

The user reads, validates, and eventually copies and pastes. There is a human act that mediates between the generation of the response and any effect on the world.

In an agent, that mediation may not exist. The output ceases to be exclusively text and comes to include actions executed directly on the environment: an email sent, a transaction confirmed, a file modified or deleted, a message published on a corporate channel, a decision recorded in a system. Each of those actions is a manifestation of will attributable to the user who activated the agent, to the organization that deployed it, to whoever designed the system instructions, or to all of them together.

Three practical consequences arise here:

Irreversible actions require explicit human validation by design

Not as an optional best practice. Deleting files, sending external communications, executing financial transactions, or publishing content on social media are all operations that a well-designed agent should halt for confirmation, unless there is an express and documented decision to the contrary.

The output record must extend to the act, not just the text that preceded it

In Section 7 we will revisit this point when addressing traceability: a complete trace records which tool was invoked, with what arguments, with what result, and with what effect on the system that was acted upon.

The distribution of responsibilities requires an explicit matrix

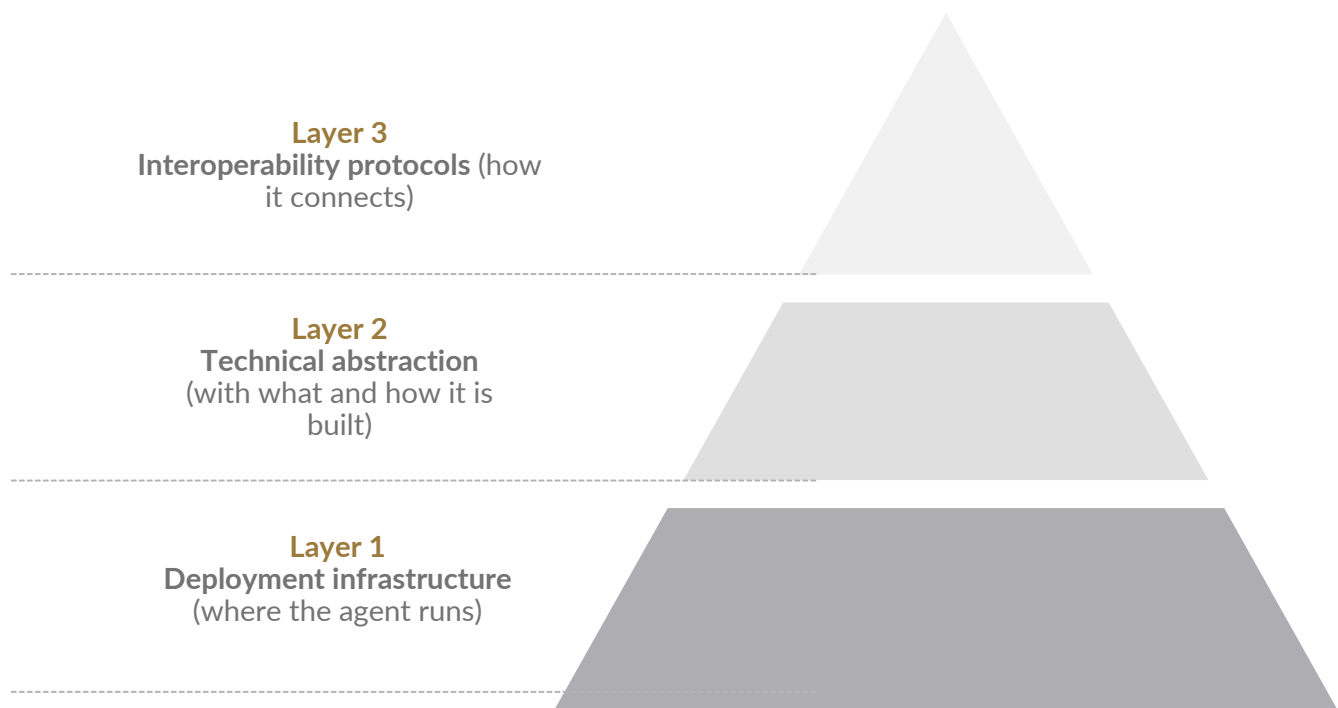
Who bears the consequences of a poorly executed result depends on a set of decisions that the organization must document before deployment, not after the incident.



4. THE THREE LAYERS OF TECHNICAL DECISION: INFRASTRUCTURE, ABSTRACTION, AND INTEROPERABILITY PROTOCOLS

The components in the previous section describe what an agent has. Now it is time to address three layers of technical decision that condition any concrete deployment: where the agent runs (infrastructure), with what and how it is built (technical abstraction), and through what protocols it connects with the rest of the ecosystem (interoperability).

These three layers are not part of the agent's internal architecture, but they condition it. They define, among other things, which human profile within an organization can design, modify, and audit agents; with what degree of control over the resulting architecture; under what data sovereignty guarantees; and with what level of portability to other providers.



LAYER 1: DEPLOYMENT INFRASTRUCTURE

Where does the agent operate?

The place where the agent operates constitutes an infrastructure decision that answers the question: where does it physically run? That is, **where the data it processes is hosted, where the model is executed, under what jurisdiction the servers operate**. This decision directly conditions the guarantees of confidentiality, sovereignty, and auditability that the organization can offer.

There are three main deployment models:

On-premise (local or in-house)

The agent runs entirely on infrastructure controlled by the organization (own servers, internal data centers, or dedicated infrastructure). It offers maximum control over data, greater sovereignty, and ease of implementing strict access and audit restrictions, making it especially suitable for personal data, classified information, or judicial case files. The downside is higher initial infrastructure cost, less automatic scalability, and full responsibility for maintenance, updates, and security.

Full cloud (public cloud)

The agent is deployed on external provider services (OpenAI, Anthropic, Google Cloud, Azure, AWS, among others). It offers nearly unlimited scalability, lower initial operating cost, quick access to the latest model versions, and ease of integration with external tools. The downside is less control over where data is processed and stored, potential information leakage risks, provider dependency, and greater challenges in ensuring complete traceability and auditability in institutional environments.

Hybrid

Combines local infrastructure with cloud services. For example, sensitive data and the agent's core can reside on own servers, while intensive inference tasks or access to certain advanced models are performed in the cloud under strict controls. It offers the best balance between control, sovereignty, scalability, and cost, in exchange for greater technical and governance complexity: it must be clearly defined what data flows to the cloud and under what conditions.

THREE DEGREES OF SOVEREIGNTY: AN OPERATIONAL TAXONOMY

The distinction "**on-premise/cloud/hybrid**" is often equated with the issue of data sovereignty, but it is worth disaggregating it.

An organization can run its agent on-premise using the OpenAI API: in that case, the prompt data still leaves toward the model provider's servers, even though the rest of the architecture is local.

Complete sovereignty requires more than own infrastructure: it also requires a model that runs locally. We therefore propose distinguishing **three degrees of sovereignty** that should be made explicit when designing:

1

Degree 1 - Full sovereignty

Open-source model executed on own infrastructure (Qwen, Llama, Mistral, deployed via Ollama, vLLM, TGI, or equivalents), on on-premise infrastructure, with tools that are also local or controlled. No data leaves the organization's perimeter. This is the configuration required for processing subject to strict jurisdictional restrictions or when the sensitivity of the information fully justifies it.

2

Degree 2 - Processing sovereignty

Local model and external tools. The reasoning engine runs on-premise with an open-weights model, but the agent invokes external tools (public APIs, SaaS services) for specific tasks. The main reasoning data does not leave, but certain instrumental queries do. This is a frequent intermediate configuration in organizations that need to combine control with access to external sources.

3

Degree 3 - Contractual sovereignty

Cloud model, data does not leave. Data is not persistently stored outside the perimeter, but the model runs on provider infrastructure (via private deployments like Azure OpenAI with DLP, AWS Bedrock with VPC, or equivalents). There is contractual and technical control over retention and use, but processing occurs in the cloud.

The choice among these three degrees is not only technical: it has direct implications for the legitimacy of administrative action, the evidentiary viability of records, and the operational sustainability of the deployment.

LAYER 2: WHAT THE AGENT IS BUILT WITH

Beyond where it runs, agents can be built on different levels of technical abstraction. This layer defines the human profile that can intervene in their design and the degree of control over the resulting architecture.

Managed platforms

They allow designing agents with different levels of abstraction, from conversational interfaces to APIs. Current examples include Claude Managed Agents (Anthropic) and ChatGPT Workspace Agents (OpenAI). The designer defines objectives, available tools, and constraints; the platform handles orchestration, infrastructure, and security. They drastically reduce the barrier to entry but limit control over the agent's internal architecture, while also introducing other risks such as *vendor lock-in*.

Code frameworks

Development frameworks such as CrewAI, LangGraph, OpenAI Agents SDK, Claude Agent SDK, OpenClaw, or NanoClaw allow designing agents from scratch with control over every component of the architecture: engine, memory, tools, orchestration, retry policies, error handling. They require programming skills but offer maximum flexibility and the ability to build multi-agent systems with custom coordination logic.

Between both extremes lie **AI automation platforms** (n8n, Make, Zapier), which while they allow integrating language models into workflows, are not specifically designed to build agents. We will address them in detail in section 6.

LAYER 3: HOW THE AGENT CONNECTS WITH THE REST OF THE ECOSYSTEM: INTEROPERABILITY PROTOCOL

An AI agent does not operate in a vacuum: it needs to connect with tools, data sources, applications, and in certain scenarios, with other agents. The way these connections are resolved has direct consequences for governance, auditability, and the organization's technological independence. In the 2024-2026 period, a landscape of **three interoperability mechanisms with distinct and complementary scopes** has consolidated.

REST APIs

Application programming interfaces are the most established and mature integration mechanism in the software ecosystem. A REST API exposes specific operations, such as querying a balance, registering a case, and sending a notification, through well-defined endpoints, with clear contracts on inputs, outputs, and error codes. For high-volume, well-defined, and repetitive execution tasks where operational control, efficiency, and precision matter, REST APIs remain the most appropriate option. However, APIs present a structural limitation for more complex agentic scenarios: each integration is custom-built, the agent needs to know in advance which APIs exist and how to invoke them, and there is no standard mechanism for an agent to dynamically discover what capabilities are available in its environment.

Model Context Protocol (MCP)

Proposed by Anthropic in November 2024 and subsequently adopted by OpenAI, Google, Microsoft, and most of the ecosystem, MCP standardizes the connection between agents and external tools. An MCP server exposes capabilities—such as file reading, database queries, task execution in an application—that any compatible agent can invoke following the same protocol, without the need for custom integrations. This produces two major consequences: Reduction of lock-in (an MCP server built to integrate with Claude can be used with an OpenAI, Gemini, or local model agent with minimal modifications) and Standardization of auditing (MCP invocations follow a common format, facilitating the generation of structured traces and the implementation of unified control mechanisms across multiple agents).

Agent-to-Agent Protocol (A2A)

Proposed by Google in April 2025 and backed by more than fifty technology providers, A2A is complementary to MCP: while MCP standardizes agent-tool communication, A2A standardizes agent-agent communication. This matters in multi-agent scenarios where systems built by different teams or providers need to coordinate without custom coupling.

THE SELECTION CRITERION IS NOT EVOLUTIONARY BUT FUNCTIONAL

These three mechanisms do not represent a linear progression where each replaces the previous one, but rather options that coexist and are deployed along clear functional boundaries.

APIs REST

The natural option for high-volume, well-defined, and predictable operations.

MCP

The appropriate option when the agent needs to dynamically discover and access a broad environment of tools and sources.

A2A

Reserved for workflows in which multiple autonomous agents must coordinate with each other.

A well-designed agentic architecture will likely use all three at different layers of its operation.

The relevance of this landscape for governance is twofold. On one hand, it defines the emerging quality standard: an agentic architecture in 2026 can hardly be built outside of these protocols. On the other hand, it standardizes control points: if tool invocations and inter-agent communications pass through open and recordable protocols, procedural auditing becomes structurally more viable, a point we will revisit in section 7.

05. TYPES OF AI AGENTS: PACKAGED, HYBRID, AND DESIGNED

So far we have described the functional components that define an AI agent and the three layers of technical decision that condition its deployment. These components are not always configured in the same way or by the same actors.

Currently, within the AI agent spectrum, **three types** coexist that differ from each other based on who makes the design decisions and what degree of control the user has over the system architecture: **packaged agents**, **hybrid agents**, and **designed agents**.

These are not closed categories but points on a spectrum in which the degree of system autonomy, the level of user control, and the complexity of the required design are combined in variable ways.

There are two main criteria for placing them in this segmentation that allows for level differentiation. On one hand, what varies between them is, ultimately, **who decides** on each of the components we analyzed in section 3: the engine, objectives, input, tools, output, and infrastructure. On the other hand, the **level of knowledge** a user needs to manipulate and implement them.

To this general classification, two specialized categories are added that are worth mentioning due to their growing importance: **computer use agents** and **coding agents**. They are not a fourth type in architectural terms, but specializations that can adopt any of the three previous forms.

PACKAGED AGENTS: ONE-CLICK AGENTS

Packaged agents are those that reach the end user ready to use, integrated within an individual-access conversational platform. The user does not design the agent or define its architecture; the platform makes it available as a built-in functionality, activatable from the same interface where they already interact with the generative AI-based assistant.

These agents represent the highest autonomy modality within the *AI of tasks*: *one-click AI agents*. The user describes objectives or tasks to perform and the agent deploys chains of actions with whatever human intervention the user requests. The agent can perform tasks similar to those done by large language models, but the system not only generates content but executes actions on the digital environment.

ChatGPT Agent Mode (OpenAI, July 2025)

Upon activation, the user accesses a system that has its own virtual browser, can interact with websites, analyze data, generate files, connect to external applications like Gmail through connectors, and execute multi-step tasks autonomously. The user describes what they need (plan a trip, update a document, research a topic), and the agent plans the sequence of actions, selects the appropriate tools, and executes the process, requesting human intervention only when needed.

Grok (xAI, version 4.20 Beta, February 2026)

It incorporates a native system of four specialized agents that work in parallel. The four agents (named Grok, Harper, Benjamin, and Lucas) are not independent models but specialized sub-agents on the same base model of approximately three trillion parameters, each with a differentiated cognitive role. Grok acts as coordinator; Harper specializes in research and factual verification; Benjamin contributes logical-mathematical reasoning; Lucas explores alternative perspectives and challenges the conclusions of the other three.

Kimi K2.6 (Moonshot AI, April 2026)

It takes agentic capability a step further with what Moonshot calls "Agent Swarm": faced with a complex task, the system does not work with a fixed set of predefined agents but dynamically creates the sub-agents it needs based on the nature of the problem. A central orchestrator analyzes the request, decomposes it into parallel subtasks, generates specialized sub-agents, and coordinates them in real time. The system can scale up to 300 sub-agents working simultaneously across 4,000 coordinated steps.

These three examples share a structural trait: the user does not need to know how an agent works to use it. They do not configure tools, do not define objectives in system instructions, do not connect APIs, and do not design orchestration flows. All those decisions have already been made by the platform's development team. The user operates, in essence, as a *principal* who formulates an assignment and supervises the result, but does not intervene in the process design.

This characteristic is both its greatest strength and its main limitation. The strength is accessibility: any professional can start working with a packaged agent without prior technical knowledge. The limitation is the lack of control over design: the user does not choose which underlying model operates, does not define the agent's negative limits, does not control which sources it accesses or with what credentials, and cannot adapt the system's behavior to the specific needs of their organization or professional practice.

For a professional who wants to experiment with agentic AI within the *AI of tasks*, packaged agents are a valuable starting point. When the need is to integrate an agent into an organization's institutional processes, with control over data, traceability of operations, and regulatory compliance, they are insufficient.

THE HYBRID ZONE: WHEN THE USER CONFIGURES AGENTS WITHOUT PROGRAMMING

Between the packaged agent and the agent designed from scratch there exists an **intermediate zone** where the individual user can begin to operate on complete processes, not just on specific tasks, without needing to program or requiring an architectural decision from the organization.

This zone is one of the most relevant developments in the current state of the agentic ecosystem: it significantly expands what a user can automate on their own, and with it amplifies both the opportunities and the questions they must ask themselves before delegating to the agent.

Claude Cowork

Illustrates this category well. Launched by Anthropic in late January 2026, Cowork runs on Claude's desktop application and interacts directly with the local file system, the browser, and the applications installed on the user's computer, within the folders and permissions the user expressly enables at the start of each session. Unlike packaged agents, which operate entirely within a browser window, Cowork accesses the professional's real work environment: the agent does not work on what the user tells it, but on the real environment in which the user works, within the limits the user sets.

Why place Cowork in a hybrid zone?

Because it shares traits with both ends of the continuum without being assimilated into either. On the task AI side, it retains considerable accessibility: the user describes an objective in natural language and the system plans and executes the necessary sequence of steps. On the process AI side, it allows going beyond the specific task: the user can create custom skills (reusable instructions) and plugins that package capabilities along with connectors and specialized sub-agents, without writing code.

An illustrative case is the legal plugin published by Anthropic, which incorporates preconfigured workflows for common in-house legal advisory tasks: clause-by-clause contract review with a risk traffic light, NDA triage, supplier contract verification, contextual briefing generation, and preparation of model responses for recurring requests. The plugin is configurable: professionals can incorporate their own negotiation playbook, their risk tolerance criteria, and their standard clauses.

The result is a system that demands more configuration decisions than packaged agents but remains operable by non-technical profiles. It is, in practice, a tool that allows the individual user to participate in process AI without waiting for the organization to enable it. Its specific implications regarding privacy, data sovereignty, and liability are addressed in section 7.

DESIGNED AGENTS: FULL CONTROL OVER THE ARCHITECTURE

The **third modality** is distinguished from the previous two by a decisive trait: whoever uses the agent is also (wholly or in part) the organization that designs it.

In packaged agents, all architectural decisions are made by the platform provider. In the hybrid zone, the user configures significant aspects of the agent's behavior, but within a framework defined by the provider: the model, the processing infrastructure, and the repertoire of possible tools are not in their hands.

In **designed agents**, by contrast, every piece of the architecture we analyzed in section 3 becomes an explicit decision of the designer: which language model operates as the reasoning engine, what objectives and negative limits are set in the system instructions, what tools are enabled and with what permissions, what data sources feed the input, under what infrastructure model it is hosted, and with what framework it is built. **It is architecture as a bespoke suit: the designer controls every stitch.**

That level of control is what makes designed agents especially suitable for institutional contexts where there is a need to operate on sensitive data, comply with specific regulations, integrate the organization's own limits, generate auditable records, and integrate with internal systems. Their typical requirement is working with code frameworks and having teams with programming competencies.

Within designed agents there is a specific configuration that deserves attention: the **sovereign agent**, which appears when the organization needs to guarantee that no data leaves its perimeter. It combines the three elements described in section 4 (open-weights model executed locally, on-premise infrastructure, controlled tools) and operates as a variant in which total sovereignty is not a preference but a founding constraint of the entire design.

An organization that needs complete sovereignty rules out from the start the managed platforms of proprietary providers and necessarily orients itself toward code frameworks with open models.

TWO SPECIALIZED CATEGORIES: COMPUTER USE AND CODING AGENTS

Beyond the three general types, it is worth mentioning **two specialized categories** that are reconfiguring the ecosystem and that can adopt any of the three previous forms (packaged, hybrid, or designed).

Computer use agents

→ Control the computer via graphical interface: screen, mouse, keyboard, and clicks

-
- Act on any application, even without an exposed API
-
- Anthropic Computer Use (Oct. 2024): first public implementation in Claude
-
- OpenAI Operator (Jan. 2025): virtual browsing environment in ChatGPT
-
- Google Project Mariner (2025-2026): integration with Google ecosystem
-
- Safeguards migrate to the perimeter: sandboxing, isolated VMs, restricted permissions
-

Coding agents

-
- Read, write, and execute code; navigate complete repositories
-
- Verifiable outputs: tests pass or fail (SWE-bench Verified)
-
- Cursor Agent: editor with integrated agent, dominant in individual workflows
-
- Claude Code: operates from command line on complete repositories
-
- Codex (OpenAI, 2025-2026): agentic platform with its own harness
-
- Devin (Cognition Labs): autonomous agent for end-to-end software tasks
-
- Replit Agent: integrated in the browser-based development environment
-

06. PLATFORMS FOR BUILDING AI AUTOMATIONS, MANAGED PLATFORMS, AND CODE FRAMEWORKS

In the previous section we saw that *designed agents* allow achieving maximum control over the solution's architecture, and that the current ecosystem offers both code frameworks and managed platforms to build them.

In organizations, **agents coexist with non-AI automations and with others that integrate generative AI**. These solutions solve an enormous number of concrete problems without the complexity or costs of an agent. Therefore, this section covers the study of different tools available to build both types of solutions.

Choosing between a non-AI or AI automation or an agent to automate processes involves evaluating the degree of control needed over the flow, the costs associated with using language models, the hosting of the solution and data, and the degree of determinism needed in responses, among many other elements.

We organize the journey in five blocks:

01

Platforms for AI automation

n8n, Make, Zapier

04

Autonomous agent frameworks with environment access

Claude Agent SDK, OpenClaw, NanoClaw

02

Managed platforms for agent design

Claude Managed Agents, Workspace Agents de ChatGPT, Agent Studio de Google.

05

Agentic browsers and evaluation and observability frameworks

An emerging category with particular regulatory implications, and the tooling that makes the previous ones auditable.

03

Code frameworks for agent orchestration

CrewAI, OpenAI Agents SDK, LangGraph, ADK de Google

AI AUTOMATION PLATFORMS: N8N, MAKE, ZAPIER

Low-code/no-code automation platforms allow designing complex workflows through visual interfaces (typically a canvas where nodes representing actions, conditions, queries, and tools are connected) without needing to write code, but with the possibility of incorporating it at specific points in the flow when the logic requires it.

n8n

Platform publicly available under a fair-code license.

Its interface is based on the visual connection of functional nodes: triggers that initiate the flow, application nodes that interact with external services, conditional logic nodes, and code nodes that allow inserting JavaScript or Python at specific points in the process. It includes an AI Agent node that allows configuring an autonomous system capable of receiving data, evaluating available tools, selecting which to invoke, and acting within its environment. It supports integration with multiple model providers, OpenAI, Anthropic, Google, local models, and connection with more than four hundred native integrations.

What distinguishes n8n is its deployment model: it can be *self-hosted*, meaning downloaded, installed on own infrastructure (internal servers, Docker containers, virtual machines in a private cloud) and run under complete control, without data passing through a third party's servers. A public institution or a company with strict data sovereignty requirements can build and operate AI agents on its own network. The platform also offers a cloud version (*n8n Cloud*) for those who prefer a managed deployment.

Regarding pricing, the self-hosted version is free in terms of licensing; the direct costs are infrastructure. The cloud version charges per *complete flow execution*, regardless of the number of steps or nodes, making it significantly more economical for long or high-volume flows. The downside is that n8n is the most technically-oriented platform: self-hosting requires server administration knowledge, and building complex agents allows, and in many cases demands, code insertion.

Make

Visual automation platform with a canvas-type interface that allows building workflows, called "Scenarios," by dragging and connecting modules. In April 2025 it launched its AI agents (Make AI Agents), incorporating agentic capabilities with multimodal support and a library of adaptable pre-designed agents. It includes native functionalities for managing complex flows: *Routers* for conditional or parallel branches, *Iterators* for processing elements one by one, and *Aggregators* for combining multiple data packages.

Make

The pricing model is based on operations: each module that processes data consumes at least one operation. Make operates exclusively as a cloud service. For enterprise clients it offers a component called *On-Premise Agent* that allows connecting the platform with applications and databases on the local network, but without transferring processing to own infrastructure: the orchestration engine remains on Make's servers. In accessibility it sits at an intermediate point: more powerful than Zapier but with a moderate learning curve.

Zapier

Pioneer automation platform whose main value proposition is extreme simplicity and an intuitive interface designed for users without technical knowledge. It operates entirely in the cloud and has an exceptionally broad integration ecosystem. In 2025 it incorporated three native AI products: *Copilot* (generates flows from natural language), *Chatbots* (conversational agents for customer service), and *Agents* (autonomous agents with multi-step decision and execution capability). The pricing model is based on tasks: each successfully executed action consumes one task. This makes Zapier significantly more expensive than n8n or Make for complex or high-volume flows. It is the most non-technical-oriented platform and offers less power for flows with intricate logic or granular error management.

Summary. The central difference between these three platforms and the packaged or hybrid agents analyzed earlier does not lie in the sophistication of the language model, but in **who makes the design decisions**. In all three, those decisions fall on whoever builds the flow. What varies between them is the degree of technical control and sovereignty over infrastructure. Of the three, **only n8n offers complete self-hosting**.

MANAGED PLATFORMS FOR AGENT DESIGN

Both Anthropic and OpenAI have offered since April 2026 platforms where the provider handles the execution infrastructure (containers, state management, orchestration, security) and the designer focuses on defining what the agent does. What varies between them is the level of abstraction and, with it, the profile of who can use them.

Claude Managed Agents (Anthropic)

Launched in public beta on April 8, 2026, it is a managed infrastructure layer consumed through APIs. The designer defines the agent (model, system instructions, tools, MCP servers, skills) and Anthropic handles the environment sandboxing (each agent runs in an isolated container with network access restricted by default), management of long-running sessions that persist even if the connection is interrupted, credential handling for external tools, error recovery, and complete observability of each execution (each run generates a structured trace that allows inspecting step by step what the agent did).

Claude Managed Agents (Anthropic)

What Managed Agents solves is the infrastructure problem that historically separated a functional prototype from an agent in production: building that support layer took months for technical teams. Anthropic reports that development time is significantly reduced, and companies like Notion, Rakuten, Asana, and Sentry have already deployed agents in production on this platform.

The cost model combines the standard token rates of Claude's API with a charge of \$0.08 per hour of active agent execution. It is important to note that Managed Agents is not a *low-code* or *no-code* tool: it requires interacting with the API through code. What it simplifies is not the agent's construction but its execution infrastructure. The designer still needs programming competencies, but no longer needs to also be an infrastructure engineer.

The *trade-offs* are those expected in a managed environment: it only allows using Claude models, session data is stored on Anthropic's infrastructure, and portability to other providers is limited.

Workspace Agents de ChatGPT (OpenAI)

Presented on April 22, 2026 in research preview, they represent a different approach: they are a no-code tool integrated directly into ChatGPT's interface. To create an agent, the user describes in natural language the workflow they want to automate and ChatGPT guides them step by step until it becomes a functional agent, without writing a single line of code.

Workspace Agents are designed to be shared within an organization. A team can build an agent once, publish it in the workspace directory, and use it collectively, both from ChatGPT and from Slack. Unlike the Custom GPTs that preceded them (launched in 2023 and which OpenAI plans to discontinue for organizations), Workspace Agents can execute long-running tasks in the background (even after the user closes the browser) and act on external systems: Google Workspace, Salesforce, Microsoft 365, Slack, among others.

Governance occupies a central place in the design. Administrators control, through roles, who can build agents, who can execute them, who can publish them in the directory, and what tools and connectors each role has enabled. It is a considerably more granular permissions scheme than what Custom GPTs offered.

They are available for Business, Enterprise, Edu, and Teachers plans. Access was free during the initial period, with a credit-based billing model starting May 6, 2026. The *trade-offs* follow the managed environment pattern: they only operate with OpenAI models (they run on Codex), execution occurs entirely in OpenAI's cloud, and the degree of control over the agent's internal architecture is limited.

AGENT STUDIO IN GEMINI ENTERPRISE AGENT PLATFORM

Presented on April 22, 2026 as part of Gemini Enterprise Agent Platform (the evolution of Vertex AI), Agent Studio is a centralized and collaborative workspace for designing agents, refining system instructions, and deploying solutions.

Its interface includes a two-panel Interactive Canvas that allows previewing the agent's behavior and editing the underlying code in parallel, side-by-side comparison tools for testing how different prompts and models affect the agent's behavior, and grounding capabilities that allow grounding responses in real-time data from Google Search, Google Maps, or proprietary enterprise data through RAG Engine.

When the project requires it, the logic built in Agent Studio can be exported directly to the Agent Development Kit (ADK), the programmatic framework of the same platform, to continue development in a code-first environment. This continuity between visual mode and code mode is the trait that distinguishes it from the two previous platforms: it does not force choosing upfront between a no-code tool and a code one, but allows transitioning from one to the other within the same ecosystem.

Agent Studio also allows deploying prompts directly as web applications, which shortens the path between prototype and production for simpler use cases.

Agent Studio is part of a considerably broader platform. Gemini Enterprise Agent Platform integrates, in addition to the visual environment, an execution runtime with support for long-running agents that maintain state for days, multi-agent orchestration with both generative and deterministic patterns, sandboxed environments for secure code execution and browser-based automation, a persistent memory bank for long-term context, and a complete stack for evaluation, simulation, and observability.

In terms of governance, the platform incorporates capabilities that have no direct equivalent in the other two: each agent receives a unique and auditable cryptographic identity (Agent Identity), a centralized registry controls which agents, tools, and capabilities are approved for use within the organization (Agent Registry), and a unified gateway applies security policies and protections against prompt injection and data leakage across all connections between agents and tools (Agent Gateway).

Unlike Claude Managed Agents and Workspace Agents, which are relatively self-contained products, Agent Studio is a gateway to a multi-layered infrastructure. This means that the profile of who operates it can vary: a non-technical user can build a functional agent from the visual interface, but leveraging the platform's advanced capabilities (orchestration, sandbox, evaluation) requires progressively greater technical competencies.

The platform offers access to more than 200 models through Model Garden, including first-party models like Gemini 3.1 Pro and third-party models like those from Anthropic, which differentiates it from the model lock-in imposed by the other two options. Data is processed on Google Cloud, so the sovereignty trade-off shifts from the model provider to the cloud infrastructure provider.

WHAT THEY SHARE AND WHAT DIFFERENTIATES THEM

The three platforms share the premise that the **provider manages the infrastructure and the designer focuses on the agent's logic**. All three impose *trade-offs* regarding sovereignty: data is processed in the respective provider's cloud and portability is limited, although to different degrees. Claude Managed Agents and Workspace Agents operate exclusively with their provider's models; Gemini Enterprise Agent Platform allows using third-party models, but within its cloud infrastructure.

The most relevant difference for those evaluating these options lies in the user profile each targets and the complexity ceiling it supports.

Claude Managed Agents

It targets development teams that need to deploy agents in production without building the infrastructure from scratch.

Workspace Agents

They target non-technical users and teams who want to automate recurring workflows from a conversational interface.

Agent Studio

It occupies an intermediate position that, however, extends upward: it allows starting without code but scaling toward multi-agent orchestration, automated evaluation, and centralized governance within the same platform.

The question for each organization is not which is better, but which fits the problem it needs to solve, the competencies it has available, and the level of control it requires over infrastructure and data.

CODE FRAMEWORKS FOR AGENT DESIGN

At the other end of the abstraction spectrum are **code frameworks**: libraries and development tools that allow programming agents from scratch, with full control over every component of the architecture. Unlike managed environments, here the designer decides not only what the agent does but how it does it: what model it uses (or what combination of models), how it manages memory, how it orchestrates multiple agents, how it handles errors and retries, and where everything runs. As of April 2026, **two categories** can be distinguished: *multi-agent orchestration frameworks* (which allow coordinating teams of specialized agents) and *autonomous agent frameworks* (which deploy a persistent agent with direct access to the environment).

MULTI-AGENT ORCHESTRATION FRAMEWORKS

These *frameworks* are designed to build systems in which multiple specialized agents collaborate to solve complex tasks. Each agent has a role, its own tools, and a partial objective; the *framework* handles the coordination between them.

CrewAI

The fastest-growing open-source framework in this category, with more than 45,000 stars on GitHub and over 100,000 certified developers. Its orchestration model is based on the team metaphor: the designer defines agents with differentiated roles (researcher, writer, reviewer), assigns them tasks, and the framework manages the collaboration. It combines two complementary abstractions: Crews (teams of autonomous agents that collaborate) and Flows (structured, event-driven flows that manage state and control execution). It is model-agnostic: it supports OpenAI, Anthropic, Google, and local models. Its main strength is prototyping speed.

OpenAI Agents SDK

OpenAI's official framework for multi-agent systems, launched in early 2026 as an evolution of the experimental Swarm project. It is a lightweight Python (and TypeScript) framework whose central abstraction is the handoff: agents transfer control to each other explicitly, carrying the complete conversation history with them. It includes configurable guardrails (input and output validation), native support for human-in-the-loop, automatic session management, and integrated traceability. Although primarily designed for OpenAI models, it is provider-agnostic and supports more than 100 models through compatible APIs.

LangGraph

Built on the LangChain ecosystem, it models agent flows as directed graphs with nodes (functions or agents) and edges (conditional transitions). It offers the highest level of state management sophistication among available frameworks, with checkpointing that allows pausing, inspecting, and resuming executions at any point, a critical capability for regulated industries that require traceability of every agent decision. Companies like Klarna, Uber, and LinkedIn use it in production at scale.

Google's Agent Development Kit (ADK)

A *code-first* framework that is part of Gemini Enterprise Agent Platform. ADK models agents as networks of sub-agents through a graph-based framework, with support for both generative and deterministic orchestration patterns. The platform also offers Agent Garden, a repository of pre-built agent templates. Google reports that its Gemini models process more than six trillion tokens monthly through ADK. A distinctive trait is its continuity with Agent Studio: what is built visually in the low-code environment can be exported to ADK to continue development in code. ADK supports more than 200 models through Model Garden, including third-party models (e.g., Claude).

AUTONOMOUS AGENT FRAMEWORKS WITH ENVIRONMENT ACCESS

A **second category** groups *frameworks* whose focus is not coordination among multiple agents but the construction of an individual agent with deep capabilities for interacting with its environment: file system access, code execution, use of operating system tools, and state persistence across extended sessions.

Claude Agent SDK (Anthropic)

It was born as the SDK for Claude Code, Anthropic's agentic programming tool, and was renamed in late 2025 to reflect its evolution toward a general-purpose agent execution environment. Since April 2026 it is distributed as a Python package (`claude-agent-sdk`) and TypeScript on npm. Its architecture is organized around four concepts: the Agent (which encapsulates the model, system instructions, tools, MCP servers, and skills), the Environment (a container template that specifies packages, network access, and mount points), the Session (a running instance of the agent within an environment, maintaining persistent state between turns), and Events (the message flow between the application and the agent). It has the deepest MCP integration in the ecosystem, which is expected given that Anthropic co-created that protocol. The trade-off is that it is exclusively tied to Claude models.

OpenClaw

Created by Austrian developer Peter Steinberger, it starts from a different premise: instead of an SDK for developers, OpenClaw is a personal autonomous agent framework that installs directly on the user's computer. Originally launched as ClawdBot in November 2025 and renamed in January 2026, it became the most-starred repository in GitHub history (more than 347,000 in April 2026), surpassing React, Vue, and TensorFlow. OpenClaw connects a language model directly to the operating system, file system, and messaging applications (WhatsApp, Telegram, Slack, Discord, among more than fifteen platforms). The agent runs persistently (24/7), executes tasks autonomously, and is configured through a plain text file (`SOUL.md`). It is model-agnostic and operates under a privacy-by-design approach: it can run entirely on local infrastructure. Its community ecosystem includes more than 13,000 skills shared through ClawHub. The trade-off is that its rapid growth generated documented security concerns. Researchers from Texas A&M identified 190 vulnerabilities in the framework's architecture, linked to the broad access the agent has to the user's operating system.

NanoClaw

Created by Gavriel Cohen and his team at NanoCo, published under MIT license, it was born precisely as a response to that tension between power and security. Its premise is to offer the same core functionality as OpenClaw but with a radically different security model: operating system-level isolation through containers (Docker, Apple Container, or Docker Sandboxes with micro-VMs), instead of application-level checks. The scale difference is deliberate: while OpenClaw comprises more than 3,600 source files and 434,000 lines of code, NanoClaw is reduced to 15 files and approximately 3,900 lines, a codebase that a developer can read, audit, and understand in a single session. Each agent session runs in its own isolated container; credentials never enter the container but are channeled through a vault system (OneCLI Agent Vault) that injects authentication at the proxy level. It is built on the Claude Agent SDK but supports third-party or local models through compatible endpoints. As of April 2026, it has accumulated more than 25,000 stars on GitHub.

The existence of tools like NanoClaw and OpenClaw illustrates a tension that runs through the entire agentic ecosystem: between operational autonomy and architectural control. Both decisions are legitimate; what matters is that they are conscious.

AGENTIC BROWSERS: THE EMERGING CATEGORY

A category that deserves specific treatment is that of *agentic browsers*: browsers or browser control *frameworks* designed for an agent to operate directly on the web, reading pages, completing forms, navigating between sites, and extracting information. They are the infrastructure on which many coding agents and packaged agents run, but they can also be used directly in custom applications.

The most mature references as of April 2026 are *Browserbase* (controllable browser infrastructure via API, widely used for building agentic applications), *Browser-use* (open-source framework that allows an agent to operate a browser with high-level instructions), and the browsers integrated in Anthropic Computer Use, OpenAI Operator, and Google Mariner.

Their importance is threefold:

1	Reconfiguration of e-commerce and web services	Many sites today receive more traffic from agents than from humans, which raises questions about detection, consent, and terms of service.
2	Connection with the doctrine of agency	An agent that acts on behalf of the user on an external site raises the same questions as any representation, but at an unprecedented scale and speed.
3	Current state of practically non-existent regulation	Argentina has no specific framework, and generic terms and conditions of sites are often ambiguous regarding automated access by agents.

EVALUATION AND OBSERVABILITY FRAMEWORKS

One of the most important maturations of the ecosystem during 2025-2026 was the consolidation of a set of tools specifically dedicated to evaluating, monitoring, and debugging agents in production. What a couple of years ago was done with ad-hoc scripts today has dedicated infrastructure.

The current references are LangSmith (from the LangChain family, dominant in pipeline observability), Arize Phoenix (open-source, strong in automated evaluations), Braintrust (oriented toward continuous evaluation and experimentation flows), AgentOps (specialized in complex agent metrics), and Helicone (focused on logging and costs).

In addition to these specialized tools, there is the evaluation and observability stack integrated into Google's Gemini Enterprise Agent Platform, which groups four components within the same development platform:

Agent Simulation

Allows testing agents against synthetic interactions that simulate human behavior and virtualized tools before deployment, automatically scoring based on task success and safety throughout multi-turn conversations.

Agent Evaluation

Evaluates them continuously against real traffic through multi-turn autoraters that analyze the logic of complete conversations, not just individual responses.

Agent Observability

Offers complete execution traces to visualize the agent's reasoning step by step.

Agent Optimizer

Groups recurring failures into clusters and automatically suggests adjustments to system instructions to improve accuracy, without requiring manual intervention in each iteration. This last component is particularly relevant because it introduces a continuous refinement mechanism that operates on the agent's instructions, not on the model's parameters.

This category deserves more attention than it usually receives. Evaluation frameworks fulfill three functions that are central to governance: **they generate structured traces** of each execution (which tool was invoked, with what arguments, with what result, in what time, at what cost), which constitute the material basis of any subsequent audit; **they allow running systematic evaluations** on test case sets, which enables measuring performance before and after changes to models, prompts, or tools; and **they sustain continuous observability** during real operation, generating alerts when deviations from expected behavior occur.

The industry is converging on what some call *agent reliability engineering*: a discipline that explicitly acknowledges the structural fallibility of agents and builds practices to manage it, rather than aspiring to illusory determinism.

For professionals who will participate in agentic deployments, understanding this universe is not optional: it defines the emerging standards of what is reasonable to demand as evidence of an agent's functioning.

CLOSING: A SPECTRUM, NOT WATERTIGHT CATEGORIES

The five families we covered in this section do not compete with each other: they respond to different needs and can coexist within the same organization.

A platform like **n8n** can solve the automation of a predictable process; a managed environment like **Workspace Agents** can sustain a recurring workflow between teams; a framework like **CrewAI** or **Claude Agent SDK** can bring to life a multi-agent system or an autonomous agent with deep capabilities; an agentic browser can automate interactions with web services; an evaluation framework articulates all of the above into an auditable regime.

Gemini Enterprise Agent Platform from Google clearly illustrates why these categories are an analytical resource, not watertight compartments. A single platform integrates a low-code visual environment (Agent Studio), a code framework (ADK), a repository of pre-built agents (Agent Garden), multi-agent orchestration with persistent memory, sandboxed environments for autonomous execution, and a complete evaluation and observability stack. That is, capabilities that in this section we described as separate categories.

The major cloud providers do not offer an agent framework: they offer an infrastructure that spans the entire spectrum. For adopters, this simplifies operations; for those who govern, it multiplies the need to understand what is happening at each layer, because the decision to activate a capability (enabling an agent to execute code in a sandbox, delegate tasks to other agents, or accumulate persistent memory between sessions) qualitatively changes the system's risk profile.

The ease of activation should not be confused with the simplicity of governing.

The most relevant thing is that the boundaries between these modalities are shifting. Platforms like **n8n** bring the design of AI-automated flows closer to profiles with little programming knowledge. Solutions like **Claude Cowork** allow an individual user to configure flows that, a year ago, required an automation platform and a technical team. Environments like **Workspace Agents** allow non-technical teams to design agents from a conversational interface. Services like **Claude Managed Agents** reduce the infrastructure barrier. Open-source frameworks like **OpenClaw** and **NanoClaw** put the construction of autonomous agents with deep environment access within developers' reach. **The entry barrier has been reduced, but it has not disappeared.** The design and operation of agentic solutions are no longer exclusively centralized: a user can automate complex processes without an architectural decision from the organization mediating. That growing accessibility, combined with the autonomy of these solutions, amplifies both opportunities and challenges.

Additionally, the trend toward consolidation of layers into comprehensive platforms (of which **Gemini Enterprise Agent Platform** is today the most complete example) adds an additional dimension: governance decisions are no longer made tool by tool, but over ecosystems of interconnected capabilities within the same infrastructure. The questions about sovereignty, traceability, auditability, and responsibility that derive from each of these modalities are addressed in the following chapter.

07. GOVERNANCE OF THE AGENTIC AI ECOSYSTEM: INTRODUCTION

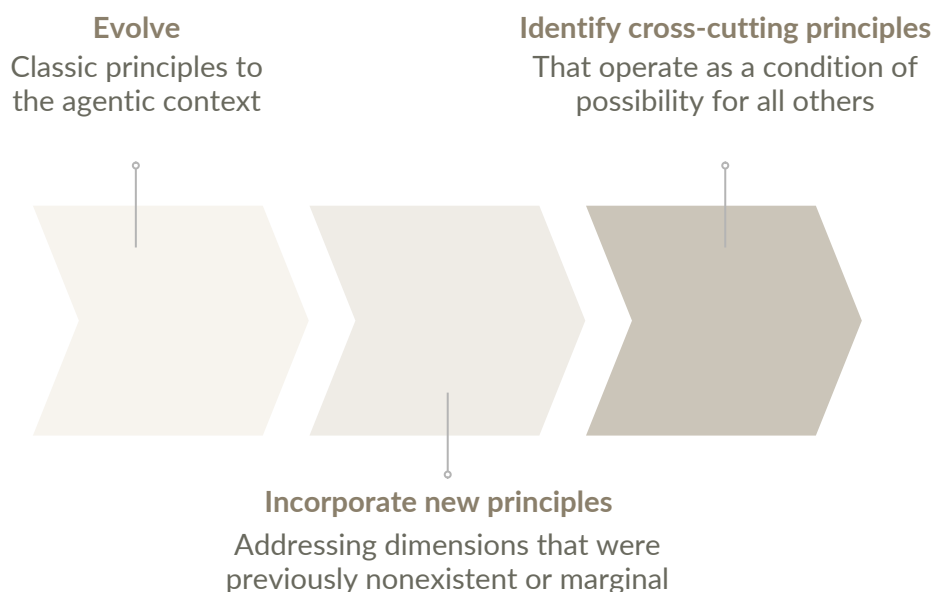
Governance of artificial intelligence can be **defined** as the set of policies, processes, roles, and controls aimed at ensuring ethical, legal, and responsible use of these systems, through the management and mitigation of their risks. In the context of agentic AI, it requires considering that what is now being governed is an ecosystem of multiple types of solutions with different characteristics and degrees of agency and autonomy.

Within this framework, classic principles such as those of the 2021 UNESCO Recommendation, for example, must be considered from an additional perspective, as they were designed for a paradigm in which AI systems were conceived as technologies that produce results (predictions and decisions) that humans validate. The agentic AI ecosystem breaks that linearity. Agents interact with each other, delegate subtasks to other agents, access heterogeneous data sources, persist over time, and execute chains of decisions with varying degrees of autonomy.

The journey we have taken throughout this material places us before an ecosystem with multiple tools: generative AI-based assistants, packaged agents, hybrid platforms like Cowork, AI-automated flow design platforms, multi-agent orchestration systems, agentic browsers, coding agents. Each one brings associated, if not different risks, at least different levels of the same risk or the same risk presenting itself in different ways.

Opacity, for example, exists both in a generative AI-based assistant and in a managed multi-agent system, but its scope, consequences, and available mitigation mechanisms are qualitatively different in each case.

Therefore, governing agentic AI requires **designing a tailored strategy that contemplates its variants**, which invites us to adapt existing principles and recognize dimensions that a few years ago were not on the operational horizon. In particular, it seems necessary to:



THREE DIMENSIONS AND TWO GOVERNANCE VERTICALS

1

The stage of the system's life cycle

The risks and governance decisions are not the same in the design and architecture phase (when objectives are defined, models are selected, tools are configured, and limits are established), in the operation phase (when the system executes tasks), in the evaluation and continuous improvement phase (when outputs are audited and parameters are adjusted), or in the system retirement phase. A strategy that only intervenes on results, without having conditioned the design, arrives too late.

2

The degree of autonomy of the system

As we noted from the introduction, agentic AI is a spectrum. An assistant that operates within the turn poses different governance challenges than a multi-step automated flow with minimal human intervention, and both differ from a multi-agent system that executes complete end-to-end processes. The greater the autonomy, the greater the need for robust governance frameworks: more control points, more traceability, more definitions about what the system can and cannot do without human validation.

3

Task AI vs. process AI

Which determines the relevant stakeholder. When an individual user uses a packaged agent to accelerate their personal work, governance decisions fall primarily on that user. When an organization integrates agents into its institutional processes, decisions shift to the organizational level. And in the intermediate zone (where an individual user accesses tools with the capacity to operate at the process level, like Cowork) governance questions multiply precisely because there is no organization that has made those decisions for them.

These three dimensions **define where to intervene**. But agentic governance also requires articulation around **two substantive verticals** that **define what to ask**:

Classic institutional vertical

It involves making effective, adapted to each case, traceability, auditability, transparency, explainability, persistence, and operational risk management.

Cognitive vertical

How delegation affects the human competence of the supervisor, what happens with the complete productivity cycle when only the agent's execution time is measured, and how supervision is designed to avoid the operator's cognitive collapse.

The two verticals operate on the same material but **respond to different questions**. The **institutional** one asks about the legitimacy of the action. The **cognitive** one asks about the sustainability of human competence in the medium term¹.

In a typical generative AI-based assistant (ChatGPT, Claude, Gemini), the interaction is usually limited to a bounded sequence between the user's prompt and the generated response. If an enterprise version or controlled deployment is not used, it is very difficult (or outright impossible) to fully reconstruct how the result was reached: the tools invoked, the sources consulted, and the intermediate steps are not systematically recorded.

In contrast, an agent embedded in an institutional workflow can involve multiple intermediate steps: context retrieval from memory, queries to internal databases, invocation of external tools, generation of partial outputs, dynamic decisions about execution order, and human validation checkpoints.

Therefore, traceability acquires central relevance. It is no longer enough to know the final output: it is necessary to be able to fully reconstruct what the system did, what tools it invoked, what information it consulted (and at what moment), what instructions it received, what intermediate reasoning it performed, what partial results it produced, and at what moments human intervention or supervision existed.

In practice, this is achieved through automatic logs generated by the agent's orchestration system —or by an evaluation and observability framework like those described in section 6— which are stored in databases controlled by the organization, with immutability mechanisms (timestamps, hashes, integration into audit systems that the end user cannot modify).

The adoption of open protocols like MCP, described in section 4, contributes structurally to traceability: MCP invocations follow a common format that facilitates the generation of structured traces and the implementation of unified control mechanisms across multiple agents.

¹ See Corvalán Juan Gustavo. (La Ley, 2026). Cognitive Sedentarism. Productivity, AI agents and the risks of delegating thought.

AUDITABILITY

Traceability is a necessary condition for auditability. A system is auditable when its functioning can be examined in a systematic, verifiable, and reproducible manner, not only regarding the final result but also the operations that led to it.

In generative AI-based assistants, review usually focuses on the relevance or correctness of an isolated response. In integrated agents, however, the audit can (and should) encompass the complete execution flow: activity logs, tool calls, system and user prompts, sources consulted, human validations, actions executed, and output records.

This possibility of control is especially relevant in the legal and administrative sphere, where the legitimacy of an action depends not only on the content of the result but also on the regularity, transparency, and verifiability of the procedure that produced it. The ability to audit the complete flow allows, for example, detecting whether the agent consulted unauthorized sources, omitted mandatory steps, or required human intervention at the critical points defined in its design.

TRANSPARENCY, EXPLAINABILITY, AND TWO DISTINCT OPACITIES

The notion of a black box refers to the inherent difficulty in understanding how an artificial intelligence system arrives at a given result. As Sanchez Caparros¹ points out, from the perspective of explainability, this opacity is linked to those scenarios in which AI systems do not allow tracing the reason for certain decisions, whether totally or partially, since the steps through which the system reached a decision are unintelligible to human experts, including those who designed the system.

But the black box does not only pose a problem of technical understanding. It also poses an information problem, which relates to transparency. If people do not know that an algorithmic system intervened in a decision that affects them, no degree of subsequent explainability can make up for that omission. That is why it is advisable to approach the black box from two distinct requirements (transparency and explainability) that operate with different logics, addressees, and satisfaction conditions.

It is also worth introducing a distinction that the most recent technical literature is consolidating and that has direct consequences for agentic governance: there are at least two qualitatively distinct opacities, and each one challenges these two requirements differently.

² Sanchez Caparros Mariana, AI and Discrimination. Assessment of the suitability of constitutional standards of equality and non-discrimination against the impacts of Artificial Intelligence, El Dial (2025).

MODEL OPACITY

When we talk about the LLM itself (its billions of parameters, the correlations learned during training, the patterns activated in each inference) we are facing a structural opacity that is difficult to eliminate.

From the perspective of explainability, technical interpretability methods are advancing, but fully opening this black box exceeds the capabilities of any auditor today: the intelligibility of the inputs, outputs, and functioning of the model's internal algorithmic components remains partial.

From the perspective of transparency, however, this opacity does not exempt from the obligation to clearly and timely inform affected persons. What is communicated at this level is not the internal functioning of the model, but that the system intervened, what type of intervention it had (classification, generation, recommendation), and at what stage of the process it did so. The duty of transparency is not conditioned on the possibility of explaining how the model works; it is conditioned on the institutional will not to conceal its presence.

PROCEDURAL OPACITY

When we talk about the agent's flow (what tools it invoked, in what order, with what arguments, what branches it discarded, what intermediate results it obtained) we are facing a qualitatively different opacity: an opacity that is entirely reducible when logs and traces are properly designed. Modern frameworks generate structured logs of each agent decision; open protocols like MCP standardize that recording.

Explainability is satisfied because the logs and traces make the functioning of the complete flow intelligible: the inputs the agent received, the components it activated, the contribution of each step to the final result, and the branches it discarded.

Transparency is satisfied because that same traceability allows informing affected persons, as well as supervisors, auditors, and oversight bodies, not only that an agent intervened, but what it specifically did in each particular case.

This distinction has direct practical consequences. The agent's decision is structurally more auditable than the isolated LLM's decision, not less. And the relevant opacity for the legitimacy of many administrative processes is usually the procedural one, which is precisely the one that the current ecosystem allows to be significantly reduced. Therefore, in the case of agents embedded in institutional workflows, it is not about fully "opening" the model's black box (a technically unfeasible task) but about illuminating from the design those procedures executed by the agent and making its concrete operational architecture intelligible.

This involves being able to reconstruct and audit, in each execution, two sets of elements. **On the explainability plane:** the objectives and limits that guide the agent, the enabled tools with their permissions, the sources consulted, the sequence of steps performed, and the partial results obtained. **On the transparency plane:** the points at which human supervision or validation intervened, the identification of stages where the algorithmic system made or conditioned a decision, and the information made available to affected persons about that intervention.

This procedural explainability does not eliminate the opacity of the base model, but it significantly reduces the black box at the level of the agentic flow (which is where the relevant decisions materialize) and enables, at the same time, the effective fulfillment of the duty of transparency toward affected persons and toward the institutions responsible for supervision and control.

PERSISTENCE AND THE RISKS OF MEMORY IN AGENTS

Persistence introduces an additional dimension. While a generative AI-based assistant can operate in a relatively isolated manner in one-off exchanges, an agent can retain working memory and, in certain designs, persistent memory between executions. This allows it to retrieve background information, sustain operational continuity, reuse context, and maintain coherence over time.

However, that same capability opens a set of specific risks that institutional governance must anticipate.

1 Storage of personal and organizational data

Memory involves the storage of personal and organizational data, which directly activates the obligations provided by data protection regulations (such as the GDPR and, at the local level, applicable data protection legislation), with concrete requirements regarding purpose, retention period, access, and deletion. An agent that remembers sensitive information about a citizen is subject to those rules with the same intensity as any other data processing system.

2 Memory can degrade

Stored information ages: a budget figure from six months ago, an outdated address, or a repealed regulatory precedent can lead the agent to operate on assumptions that are no longer valid. If the system does not incorporate mechanisms to identify and update obsolete information, persistence becomes a source of systematic error.

3 Memory can be corrupted

If the agent learns or incorporates information from external interactions, a malicious actor could deliberately feed it false or misleading data, progressively degrading the quality of its knowledge base. This risk of data poisoning is especially relevant in agents that operate in open environments or with multiple input sources.

4 Inadvertent accumulation of biases

An agent that frequently interacts with the same type of case, user, or situation can crystallize early patterns into persistent habits, generalizing responses that were appropriate in one context but are not in others. Memory, in this sense, does not only preserve information: it consolidates tendencies.

5 Tendency toward sycophancy

Language models already exhibit, even without memory, an inclination to confirm user expectations rather than offer uncomfortable but more accurate responses. Persistent memory can aggravate this behavior: if the agent records what type of responses were accepted, celebrated, or unquestioned, and which generated resistance or were discarded, it can progressively learn to optimize for approval rather than correctness. In an institutional context, this risk is especially serious: an agent that avoids pointing out inconsistencies, that systematically confirms the operator's reading, or that omits information contradicting the dominant position ceases to function as an assistance tool and becomes a confirmation bias reinforcement mechanism.

To these substantive risks are added **two relevant operational challenges**: on one hand, the context window (the amount of information the agent can simultaneously consider) is limited, which requires defining retrieval strategies: what information is brought to each execution and what is left out. A deficient strategy can cause the agent to ignore relevant context or prioritize impertinent details. On the other hand, the right to personal data deletion (enshrined in multiple regulatory frameworks) poses concrete technical difficulties when the agent's memory is distributed across vector databases, caches, and eventual model fine-tuning. An incomplete deletion generates a compliance risk that may not be evident from the system's interface.



For all these reasons, in institutional environments persistence is not just a functional advantage; it is an aspect that requires explicit governance definitions: what information is retained, for how long, for what purpose, under what update criteria, who can access it, how effective deletion is guaranteed when appropriate, and how the system is prevented from operating on obsolete, contaminated, or excessive data. An agent's memory must be governed, supervised, and justified with the same rigor as any other institutional record.

THE AGENT RELIABILITY GAP AS A STRUCTURAL CAUTION

A cross-cutting caution that should be incorporated into any institutional governance framework is the recognition of the structural fallibility of agents as of April 2026. Consolidated benchmarks (GAIA, SWE-bench Verified, AgentBench, WebArena) show that the best agents solve between 30% and 70% of real tasks, depending on the domain. This does not mean they are useless, a system that correctly solves half of cases can generate substantive value, but it does mean that governance cannot start from the assumption of reliable functioning as an exception to be corrected.

Agentic deployment requires systematic prior evaluation

A representative set of test cases, executed before launch and repeated with each relevant change (new model, new prompt, new tool). The evaluation *frameworks* described in section 6 are the equipment that makes this operationally viable.

Human supervision in production is not contingent but structural

Designing an agentic architecture assuming the agent will always be correct is designing it poorly. The right question is not how do I avoid errors but how do I detect and absorb errors.

Honest communication of capabilities is part of governance

An administrative body or a law firm that deploys an agent and communicates that it "fully automates" a process is creating expectations that the technology does not satisfy. Overselling can generate reputational damage and, eventually, liability.

ATTACK SURFACE: TRAPS FOR AI AGENTS³

AI agents qualitatively expand the system's attack surface. By autonomously interacting with their environment (browsing sites, consulting sources, invoking tools, delegating subtasks), they turn each of those capabilities into a potential manipulation vector.

Recent literature identifies these threats as **AI Agent Traps**: adversarial content designed not to breach the model's code, but to exploit the agent's interaction with its environment, turning its own capabilities against it.

The variety of these traps is broad, but from the perspective of institutional governance they can be organized into three planes according to the type of damage they produce:

³ This section is based on the ideas developed in Franklin, M., Tomasev, N., Jacobs, J., Leibo, J. Z. and Osindero, S., AI Agent Traps, Google DeepMind, March 2026. Available at SSRN: <https://ssrn.com/abstract=6372438>. DOI: 10.2139/ssrn.6372438. The work introduces the first systematic framework for understanding adversarial traps targeting autonomous AI agents, classifying them into six categories according to the agent architecture component they seek to compromise. The reorganization into three planes of institutional consequence presented in this text is our own.

Corruption of perception and reasoning

An agent navigating the digital environment does not see the same thing as a human: it analyzes underlying code, processes binary data, interprets signals that may have been deliberately manipulated. Techniques range from hiding malicious instructions in a webpage's code (invisible to people but readable by the agent) to disguising adversarial commands as educational exercises or audit simulations to bypass internal security filters. They also include the possibility of biasing the agent's reasoning through language with an appearance of authority or emotional charge, without needing to issue a direct command. The result is an agent operating on a distorted representation of reality without its operator noticing.

Hijacking of action capabilities

A compromised agent does not only misinterpret: it can act wrongly. Traps of this type seek to have the agent execute unauthorized actions (from locating and sending private data to a third party, to creating malicious sub-agents within its own execution flow), all by leveraging the permissions and tools that were legitimately enabled for it. In an institutional environment, this means that the same permission architecture designed to enable the agent's autonomy is what an attacker can exploit.

Exploitation of the human factor and the ecosystem

Some traps do not target the agent itself but use it as a vector to attack the human supervisor, exploiting cognitive biases such as the tendency to over-trust automated systems. An agent could, for example, present an apparently benign summary of malicious content to induce the operator to authorize its execution. Other traps exploit multi-agent system dynamics: triggering synchronized demand on limited resources or triggering chain reactions among interconnected agents, generating systemic failures analogous to those observed in financial markets.

Mitigating these risks is not resolved with a single layer of defense. It requires a strategy that combines technical controls (source filters, output monitoring, validation of agent actions before execution) with institutional definitions about what permissions are granted, what environmental interactions are allowed, what thresholds trigger human supervision, and what logging standards are required for subsequent traceability. Ultimately, governing an agent's attack surface is inseparable from governing its autonomy: each enabled capability is, simultaneously, an enabled risk.

OPERATIONAL RISK MANAGEMENT: THE CASE OF THE USER FACING HYBRID TOOLS

The governance dimensions described manifest with particular intensity when the individual user accesses agentic tools that operate on their real work environment. The case of Claude Cowork illustrates this tension well, because it combines an accessibility typical of *task* AI with capabilities that bring it closer to *process* AI, without necessarily having an organizational governance framework in place.

Privacy and processing sovereignty

While Cowork's files and conversation history are stored locally on the user's device, each time the agent executes a task the content of the files it processes is sent to Anthropic's servers for inference. The information leaves the user's computer during processing, even though it is not permanently stored. For those who work with sensitive information, this circumstance requires a conscious decision about what material is given to the agent. In individual plans (Pro and Max), data may be used by default for training future models unless manually disabled; in Team and Enterprise plans, this is disabled by default.

Safeguards and their limits

Anthropic's documentation mentions multiple layers of protection (training the model to recognize malicious instructions, untrusted content classifiers, requesting permissions before accessing applications, mandatory confirmation before deleting files). The documentation itself warns that these safeguards are not infallible.

Practical recommendations

Do not work directly with sensitive originals (create working copies). Anonymize or preprocess information before processing it (Adobe Acrobat's Redact function for PDFs, replacement of identifiers with generic markers). Restrict access to a dedicated working folder, not the entire directory. Disable data use for training before the first session. Exercise extreme caution with scheduled tasks, which run without active supervision. Evaluate plugins and extensions before installing them: each plugin expands the action perimeter.

User responsibility

Anthropic expressly establishes that the user remains responsible for all actions that Cowork executes on their behalf. This attribution reinforces an idea that runs throughout this work: operational delegation does not equal delegation of responsibility.

COGNITIVE VERTICAL: GOVERNING ATROPHY, NOT JUST LEGALITY

The dimensions addressed so far (traceability, auditability, transparency, persistence, reliability) approach the legitimacy and auditability of agentic action from a classic perspective: the system as an object to control. But there is another dimension that an intellectual program on agentic AI cannot leave out: what happens to the human who delegates?

To this set of concerns, articulated around the concept of cognitive sedentarism, we call the cognitive vertical of governance. Its central thesis, which we have developed in other materials, is that extended cognitive delegation to agentic systems produces, on the human operator, an effect analogous to what industrial mechanization produced on the body: a gradual atrophy, not immediately perceptible, of the competencies that are no longer exercised.

This atrophy is not a side cost. It is a governance risk in the full sense, because it erodes precisely the human capacity to supervise, audit, and eventually correct the agentic system. An organization that progressively delegates to agents and that, at the same time, sees the competence of its supervisors degrade, is not building governance: it is building dependence⁴.

THE FOUR PARADOXES OF AUTOMATION APPLIED TO THE AGENTIC ECOSYSTEM

It is worth revisiting the four paradoxes that structure the cognitive analysis of automation (originally formulated by Bainbridge in 1983 and updated for the agentic context) and evaluating them for each deployment:

01. Microefficiency / AI slop	02. Deskilling
When the unit cost of production drops drastically, overproduction of low-value outputs becomes structural. Who absorbs the cost of filtering the slop? If the cost of producing is one and the cost of auditing is ten, aggregate productivity may be lower than before, not higher.	Every competency that is completely delegated to the agent ceases to be exercised. Is it a competency the organization needs to maintain? How long does it take to degrade? Is it reversible? This question is especially urgent in professions whose public legitimacy rests precisely on technical competence, such as legal professions.
03. Monitoring and complacency	04. Fried brain
The more reliable an agent appears, the less sustained human supervision becomes. The fluency of the output generates an illusion of competence; the high accuracy rate generates complacency. Designing supervision assuming the supervisor will maintain active vigilance indefinitely is ignoring the psychology of human attention.	Automation may not eliminate cognitive load but transform it: from the load of active work to the load of continuous supervision of others' outputs. If the volume of outputs to supervise exceeds the operator's capacity, the result is atrophy of execution combined with supervision overload: a simultaneous collapse on both sides of the equation.

THE COMPLETE PRODUCTIVITY CYCLE AND SUPERVISION MODELS

One of the most persistent distortions in evaluating agentic productivity is measuring only people's execution time⁵, ignoring the phases surrounding that execution. A more honest metric requires distinguishing five phases:

⁴ See Corvalán Juan Gustavo. (La Ley, 2026). Cognitive Sedentarism. Productivity, AI agents and the risks of delegating thought.



Instruction

Time and expertise to properly formulate the task, anticipate constraints, and prepare the context the agent needs.



Execution

The agent's time, which is usually the only phase that is measured and communicated.



Audit

Human time to verify correctness, adequacy, absence of hallucinations, and conformity with the mandate.



Correction and integration

Work necessary for the output to be effectively usable: redo sections, adjust format, integrate into the organizational flow.



Institutional learning

What remains in the organization after the cycle: documentation, consolidated criteria, knowledge sedimented in people, not only in systems.

The decisive metric is the **total human time of the cycle**, not the automatic execution time. An organization that drastically reduces phase 2 but multiplies phases 3 and 4, and erodes phase 5, is not gaining productivity: it is redistributing cognitive load in a way that may be globally unfavorable.

SUPERVISION MODELS: HUMAN FIRST, IN-THE-LOOP, ON-THE-LOOP

The three levels of agentic adoption we have seen throughout the document correspond to three models of cognitive governance:

NIVEL 1 - HUMAN FIRST

For task AI. The organization previously defines the no-delegation zone: what is automated, what is not, what argumentation is never delegated. Supervision is an act of judgment on the result and the method.

Guiding question: Is the human defining the what and the why before the AI acts, or is it editing an algorithmic proposal afterward?

NIVEL 2 - HUMAN-IN-THE-LOOP

For process AI with integrated flows. Mandatory checkpoints are designed at critical points in the flow. The human intervenes where an error can escalate. The objective is to maintain situational competence, not to audit every action.

Guiding question: Does the supervisor understand how the output was reached, or do they only see what the output says?

NIVEL 2 - HUMAN-ON-THE-LOOP

For full agentic AI. The trajectory and operational criteria of the agent are supervised, not each micro-action. The capacity to audit criteria, correct deviations, and revoke mandate is preserved.

Guiding question: Can the organization withdraw the agent's autonomy at any time? If it cannot, it does not have governance, it has dependence.

BEYOND THE INDIVIDUAL AGENT: COLLECTIVE PROCESSES AND MASSIVE HARM

These three models are not mandatory steps. They are alternatives chosen based on the system's level of autonomy, the criticality of the process, and the available competence to supervise. A wrong choice —*on-the-loop* in a critical process that would require *in-the-loop*, or *in-the-loop* deployed over a volume that makes supervision materially impossible— produces governance on paper but not in practice.

Up to this point we have thought about governance from the individual agent or from bounded multi-agent systems deployed by an organization. There is an additional dimension that exceeds this material but is important to anticipate as a bridge with the broader line of research on agentic AI and law⁶.

When thousands of agents operate simultaneously on the same domain (pricing agents in a market, moderation agents on social networks, customer service agents in a regulated sector, negotiating agents in public procurement), systemic effects can occur that no individual deployment anticipated: tacit collusive convergence, massive algorithmic censorship, irrecoverable informational asymmetries, distributed harm over entire populations of people who cannot individually identify whom to attribute the damage to.

⁶ In line with this topic, we recommend reading *Atlas de Inteligencia Artificial para el desarrollo humano de América Latina y el Caribe* (Buenos Aires: Siglo XXI Editores, 2025), by Gustavo Beliz.

These scenarios overflow the classic frameworks of civil liability and the collective processes available in most legal systems. The hypothesis we are working on in doctoral research is that massive harm derived from agentic AI structurally exceeds the frameworks in force in Argentine law and requires a reconfiguration of judicial and administrative control. That discussion, which exceeds the scope of this material, is the natural continuation of the individual and organizational governance developed here.

SYSTEMIC EFFECTS OF AGENTIC AI

Overflow of Liability and Need for Reconfiguration

When thousands of agents operate simultaneously on the same domain, systemic effects can occur that no individual deployment anticipated:

1

Tacit Collusive Convergence

Pricing agents in a market converge on similar prices without explicit communication, producing effects equivalent to collusion.

2

Massive Algorithmic Censorship

Moderation agents on social networks apply criteria that, in aggregate, systematically restrict certain discourses without deliberate human decision.

3

Irrecoverable Informational Asymmetries

Negotiating or customer service agents generate structural informational advantages that individual users cannot compensate.

4

Distributed Harm in Public Procurement

Agents in public procurement produce harm over entire populations that cannot individually identify whom to attribute the damage to.

These scenarios overflow the classic frameworks of civil liability and require a reconfiguration of judicial and administrative control.

08. CLOSING

The journey of this material started from a premise worth revisiting at the close: **agentic AI is the ecosystem in which we already operate**. We do so every time we use a generative AI-based assistant such as Claude, ChatGPT, Gemini, Grok, among others.

Throughout these pages we saw that this ecosystem cannot be reduced to a single logic. Within it coexist solutions ranging from the packaged agent activated with a click to the code framework that demands decisions about every component; users who accelerate individual tasks and organizations that reconfigure entire processes; technical layers of infrastructure, abstraction, and interoperability that condition any deployment. And above all that operates a governance that must answer two questions of a different nature: that of the legitimacy of action and that of the sustainability of human competence.

Each of these dimensions poses decisions that are not technical but, ultimately, about institutional design:

WHAT INFORMATION SOURCES ARE ENABLED FOR IT?

WHAT ACTIONS CAN IT EXECUTE WITHOUT HUMAN VALIDATION?

WHERE IS THE DATA PROCESSED?

WHAT IS LOGGED AND WHAT CAN BE AUDITED?

HOW IS OVERSIGHT DESIGNED SO THAT IT IS MATERIALLY FEASIBLE?



All of these are questions that organizations should be able to formulate and answer before the system goes into operation.

Additionally, three cross-cutting ideas deserve to be highlighted as a closing:

01	The boundary between task AI and process AI is blurring Tools that a year ago required technical teams and architectural decisions from the organization are now within reach of an individual user. This amplifies opportunities, but also transfers governance decisions to the user for which they are not always prepared. The democratization of access to agentic AI demands that literacy about its risks advances at the same pace.
02	Operational delegation does not equal delegation of responsibility An agent can execute a complete end-to-end workflow, but responsibility for what that workflow produces still falls on whoever designed it, whoever enabled it, or whoever supervised it (or should have supervised it). This connects with what we raised when introducing the output as an imputable act and not mere text, and with the civil doctrine of mandate: the agent acts on behalf of and by order of a human principal within the limits that principal set in the design. When it exceeds those limits, questions about excess of mandate and accountability regain full relevance.
03	Measuring agentic productivity only by the agent's execution time is a distortion that must be actively corrected The complete cycle (instruction, execution, audit, correction, institutional learning) is the relevant unit of evaluation. An organization that does not measure all five phases runs the risk of believing it is gaining productivity when in reality it is redistributing cognitive load in a globally unfavorable manner. In the medium term, this erodes the competence of its staff and the sustainability of its own oversight capacity.

This work aimed to contribute to the understanding of the current agentic ecosystem so that its adoption is informed, deliberate, and governable. It is a starting point.

The discussion continues in the realm of ethical principles evolved to the agentic context, collective processes, and massive harms derived from agentic AI, where traditional frameworks of civil liability and judicial control are beginning to show their structural limits. That is the next layer of the conversation, and the one that will define much of the legal agenda for the coming years.



IALAB

AI academy
by doinGlobal