The Autonomy Boundary

> What separates a directed AI assistant from an autonomous agent — and why autonomous agents are governed, not merely commanded.

This is a living white paper. It is maintained as agent architectures, research, protocols, and operating patterns evolve.

Executive summary

The word "agent" is becoming too broad to be useful by itself. It can refer to a chatbot, an IDE coding assistant, a scheduled model call, or a durable worker that wakes up, observes a system, decides what matters, coordinates with peers, and acts through tools.

Those systems are not the same.

The practical distinction is not whether an LLM is involved. It is not whether the system can call tools. It is not even whether the system can complete a task. The distinction is where initiative lives, where boundaries are defined, and how the system is controlled over time.

Directed agents are commanded. Autonomous agents are governed.

A directed agent works when a human drives it: prompt by prompt, session by session, task by task. The human supplies the immediate intent, watches the work, decides when to continue, and usually remains close to the steering wheel.

An autonomous agent operates under durable governance. Humans still matter, but their role changes. They define the agent's purpose, authority, tools, schedules, constraints, memory policy, escalation paths, and observability. The agent can then wake up through time, events, or messages; inspect its environment; decide whether action is needed; communicate with humans or peers; and act within bounded authority.

This paper argues that autonomy is not a mystical property of intelligence. It is an architectural and operating-model property. At minimum, an autonomous agent needs four surfaces:

- **Addressability.** A human or another agent can reach it through a stable communication interface.
- **Activation.** It can be invoked by something other than an immediate human prompt: time, events, messages, jobs, or webhooks.
- **Discretion.** It can decide what to do within a delegated role and policy boundary.

- **Accountability.** Its actions are constrained, observable, attributable, auditable, and reversible where possible.

Those four surfaces are the short version. The rest of this paper expands them into a more practical stack: how the agent wakes up, what it observes, how it decides, what it can do, how it remembers, how it communicates, and how its authority is governed.

A heartbeat, cron schedule, or webhook is therefore important, but insufficient. Those mechanisms give an agent a pulse. They do not, by themselves, make it autonomous. Autonomy appears when the pulse is connected to goals, state, tools, communication, and governance.

The goal is not to remove humans. The goal is to move human control out of the immediate command loop and into the system around the agent.

The framing is intentionally vendor-neutral. It does not depend on a specific model provider, agent framework, orchestration platform, or protocol. Protocols, runtimes, and tools can support autonomy, but the boundary is the operating model: whether the agent is continuously directed or durably governed.

The confusion

Agent language is overloaded because several different transitions are happening at once.

First, models became capable enough to reason through multi-step tasks. Tool use then gave those models ways to affect systems outside the chat window. Coding environments, browsers, APIs, MCP servers, and workflow systems added richer action surfaces. Finally, teams started wiring agents into persistent runtimes: scheduled jobs, background workers, webhooks, event streams, task queues, memory stores, and agent-to-agent communication.

Each step made agents feel more autonomous. But each step is different.

A model that can reason is not automatically an autonomous agent. A tool-using assistant is not automatically an autonomous agent. A scheduled model call is not automatically an autonomous agent. A multi-agent chat is not automatically an autonomous agent team.

The useful question is narrower:

> Does the system require continuous human command, or can it operate under durable governance?

That question keeps the definition practical. It also avoids several common mistakes.

The first mistake is treating autonomy as a binary. In practice, autonomy is a gradient. An agent may be autonomous for low-risk observation, semi-autonomous for routine remediation, and human-gated for irreversible changes.

The second mistake is treating autonomy as the absence of control. That is backwards. Useful autonomy depends on more control, not less. The control simply moves from constant instruction to durable governance.

A third mistake is treating delegation as autonomy. Delegation can be asynchronous without being autonomous. If a human defines the task, timing, context, success criteria, and next step, the agent may be doing valuable work, but the human is still carrying the operating loop. Autonomy begins when the agent can participate in deciding whether work needs to happen at all.

Older definitions still help

The distinction is not new. Agent research has wrestled with the boundary between an agent and an ordinary program for decades.

Wooldridge and Jennings described a widely used "weak notion" of agency: autonomy, social ability, reactivity, and pro-activeness. That framing still maps well to modern LLM agents:

Classic property	Modern runtime question
Autonomy	Can the system control some of its actions and internal state without direct human intervention?
Social ability	Can it communicate with humans or other agents through a shared protocol?
Reactivity	Can it perceive changes in its environment and respond in time?
Pro-activeness	Can it take initiative toward a goal rather than only answering a direct prompt?

Franklin and Graesser offered another useful lens: an autonomous agent is situated in an environment, senses that environment, acts on it over time, and does so in pursuit of an agenda. That definition is broad enough to include many systems, but the key phrase for software agents is **over time**. A one-shot model call can be useful. A session-bound assistant can be powerful. But autonomy becomes much more credible when the system persists across time and can change what it later observes.

Modern LLM-agent literature updates the vocabulary. Surveys of LLM-based autonomous agents tend to emphasize planning, memory, tool use, action, environment interaction, and evaluation. NIST's security work describes AI agent systems as systems capable of planning and taking autonomous actions that affect real systems or environments, and it calls out the importance of tool permissions, monitoring, access patterns, and autonomy as dimensions of agent risk.

The old and new framings converge on the same practical point:

> An autonomous agent is not merely intelligent. It is situated, activated, persistent, communicative, and able to act within some boundary of discretion.

Commanded versus governed

The commanded/governed distinction is the clearest way to separate directed agents from autonomous ones.

Commanded agents

A commanded agent is driven by direct human instruction. It may still be powerful: it can inspect files, call tools, run tests, edit code, browse the web, summarize logs, or reason through a plan. But the human remains the primary scheduler, router, and controller.

The human decides:

- When the agent starts.
- What the immediate task is.
- Which context matters.
- Whether the next action is worth taking.
- When the work should stop.
- Whether another agent should be involved.
- Whether the output is acceptable.

This is the dominant experience of many IDE, CLI, editor, and chat-based agents. The agent can be highly capable, but it is mostly session-bound and human-steered.

This can include asynchronous delegation. A human might ask an agent to fix a bug, generate a report, or draft a design while they do something else. That may be valuable delegation, but it remains commanded when the human still supplies the start signal, scope, and decision to continue.

Governed agents

A governed autonomous agent is not free to do anything it wants. It is authorized to operate inside a defined role.

The human or organization defines:

- The agent's identity and purpose.

- The systems it can observe.
- The tools it can use.
- The schedules or events that activate it.
- The policies that constrain it.
- The memory it can read and write.
- The peers it can contact.
- The actions it can take without approval.
- The actions that require escalation.
- The logs, traces, and evidence it must leave behind.

The human is still in control, but not by typing every command. Control has moved into governance surfaces.

That shift is the autonomy boundary.

Dimension	Commanded / directed agent	Governed / autonomous agent
Start signal	Human prompt	Human prompt, schedule, heartbeat, webhook, event, or peer
Control model	Human steers the immediate loop	Human defines durable policy and boundaries
Scope	Session or task	Role, responsibility, or operating domain
State	Often ephemeral or session-local	Persistent memory, logs, task state, and decision history
Communication	Mostly human-agent	Human-agent and agent-agent
Action	Human closely approves or directs	Agent acts within bounded authority
Accountability	Transcript or local history	Identity, audit trail, trace, policy, and escalation record

This does not mean commanded agents are inferior. Commanded agents are often exactly what a person wants. They are appropriate when work is exploratory, high-risk, ambiguous, or tightly coupled to human taste. The claim is narrower: commanded agents and autonomous agents should not be described as if they have the same operating model.

The pulse is necessary, but not sufficient

A pulse is one of the most visible signs of autonomy.

A heartbeat loop, cron schedule, task runner, or recurring job lets an agent wake up without a human typing a prompt. That matters. It changes the agent from a passive respondent into a durable participant in a system.

But a pulse alone is not autonomy. A shell script can run every five minutes. A CI job can run on a schedule. A database backup can run nightly. Those systems are automated, but not necessarily autonomous in the agentic sense.

The pulse matters when it starts a loop like this:

1. Wake up.
2. Read current state.
3. Compare state to goals, role, policy, and memory.
4. Decide whether action is needed.
5. Choose an action or decide not to act.
6. Use tools within permission boundaries.
7. Record what happened.
8. Communicate if needed.
9. Escalate if policy or uncertainty requires it.

The key is discretion. The agent is not merely executing a predetermined script. It is evaluating a changing environment and selecting actions within a governed role.

This is why heartbeat, cron, scheduled jobs, and webhooks should be treated as **activation surfaces**. They are the nervous system's incoming signals. They create opportunities for autonomy, but they do not define autonomy on their own.

The minimum autonomy stack

A practical autonomous agent needs a stack of capabilities. Not every agent needs every capability at full strength, but the absence of any layer weakens the autonomy claim. The stack below expands the four surfaces from the executive summary into operational design requirements.

1. Addressability

An autonomous agent needs a stable way to be reached.

For human-agent interaction, this can be a chat interface, command-line call, HTTP API, dashboard, or task endpoint. For agent-agent interaction, it should be structured enough that agents can discover capabilities, exchange messages, route work, and manage tasks without sharing private implementation details.

This is where protocols such as A2A matter. A2A is not what makes an agent autonomous, but it provides an important communication substrate: discovery, capability negotiation, task management, and secure information exchange between independent agents.

Without addressability, the agent is not really a participant. It is an implementation detail.

2. Activation

An autonomous agent needs a way to start work that is not limited to an immediate human prompt.

Common activation surfaces include:

- Heartbeats.
- Cron-like schedules.
- Calendar tasks.
- Event streams.
- Webhooks.
- Queue messages.
- A2A messages from humans or peer agents.
- Completion events from previous jobs or continuations.

Activation is the difference between "I can ask it to do something" and "it can notice that work may need to happen."

3. Observation

An autonomous agent needs access to enough state to decide whether action is needed.

Observation may include:

- Repository state.
- Logs.
- Metrics.
- Kubernetes objects.
- CI results.
- Issue or discussion activity.
- Prior decisions.
- Memory files.
- External events.
- Messages from other agents.

Observation should be scoped. An agent does not need omniscience. It needs the state required for its role.

4. Interpretation and discretion

The agent must be able to interpret what it observes and decide what to do.

This is the point where autonomy differs from ordinary automation. A conventional job usually follows a predetermined path. An autonomous agent evaluates the situation, reasons about goals, applies policy, and chooses among possible actions — including doing nothing.

Doing nothing is important. A reliable autonomous agent must be allowed to conclude that no action is warranted. Otherwise it becomes a noisy automation loop that creates work simply because it woke up.

5. Action

An autonomous agent needs tools that let it affect its environment.

The action layer may be read-only, write-capable, or mixed. A reliability observer might inspect service health, logs, metrics, and events, then report anomalies. A release agent might create tags or publish artifacts. A documentation agent might edit markdown and open a change. A remediation agent might restart a service or roll back a deployment.

The important point is that action should be explicit, permissioned, and observable.

Tool access is also where risk rises. NIST's tool-use taxonomy is useful here because it separates functionality, access patterns, risk, reliability, modality, monitoring, and autonomy. A read-only diagnostic tool has a different risk profile than a write-capable tool that can affect production state.

6. Continuity

An autonomous agent needs continuity across activations.

Continuity does not require perfect memory. It does require enough persistent state that the agent can avoid starting from zero every time it wakes up.

Useful continuity surfaces include:

- Conversation logs.
- Session state.
- Memory files.
- Decision logs.
- Task stores.
- Prior tool-call records.
- Escalation history.

- Agent-specific notes.
- Shared team memory.

Without continuity, a scheduled agent is just a recurring amnesiac. It may still be useful, but its autonomy is shallow.

7. Communication

Autonomy is not isolation. A useful autonomous agent must communicate.

At minimum, a human should be able to reach the agent, ask what it is doing, and provide new direction. In multi-agent systems, peer agents also need a way to request work, transfer context, ask for status, raise blockers, and coordinate handoffs.

This is where social ability becomes operational. A single autonomous agent can act within a role. A group of autonomous agents can divide work, cross-check each other, route exceptions, and maintain shared state.

8. Governance

Governance is the layer that makes autonomy safe enough to use.

Governance includes:

- Identity.
- Authentication.
- Authorization.
- Tool permissions.
- Scheduling rules.
- Scope boundaries.
- Human approval gates.
- Escalation rules.
- Audit logs.
- Traceability.
- Rollback paths.
- Cost controls.
- Memory policy.
- Data boundaries.
- Revocation.

This is the part that is easy to understate. A system can be autonomous and reckless. That is not the goal. The goal is governed autonomy: agents that can take initiative without escaping accountability.

A simple autonomy test

A system is meaningfully autonomous when the answer to most of these questions is yes:

1. **Can it be addressed?** Can a human or peer agent reach it through a stable interface?
2. **Can it wake up?** Can time, events, messages, or schedules activate it without a human prompt?
3. **Can it observe?** Can it read enough current state to understand whether action is needed?
4. **Can it choose?** Can it decide among actions, including no action, within a role?
5. **Can it act?** Can it use tools to affect the environment within explicit permissions?
6. **Can it remember?** Can it carry relevant context across activations?
7. **Can it communicate?** Can it report, ask, delegate, escalate, or coordinate?
8. **Can it be governed?** Can its authority be constrained, monitored, audited, and revoked?

A chatbot may satisfy the first question. A coding assistant may satisfy several. A scheduled automation may satisfy the second. A workflow engine may satisfy activation and action but not interpretation. A durable autonomous agent should satisfy all eight at least within its operating domain.

The test does not require broad authority. A read-only observer can be autonomous if it wakes up, inspects state, judges anomalies, records evidence, communicates findings, and escalates issues according to policy. A write-capable remediation agent may be less autonomous in practice if every action is manually commanded.

Autonomy is therefore not the same as write access. It is the capacity to operate under governance.

Examples

The commanded coding assistant

A developer opens an IDE agent and asks it to fix a failing test. The agent reads files, proposes changes, edits code, runs tests, and explains the result.

This is agentic work. It is not autonomy in the sense this paper uses the term.

The human initiated the work, scoped the task, watched the loop, decided which suggestions to accept, and ended the session. The agent had tools and reasoning, but the human remained the scheduler and controller.

The scheduled report generator

A workflow runs every Monday morning. It calls a model, summarizes recent activity, and emails a report.

This has activation, but limited autonomy.

If the workflow simply follows a fixed script, it is automation with model assistance. It becomes more autonomous when it can decide what changed, identify anomalies, ask follow-up questions, select sources, delay publication when confidence is low, and record the reasons for those choices.

The reliability observer

A reliability agent wakes up every few minutes. It inspects service restarts, deployment events, recent logs, metrics, and other operational signals. It compares the current snapshot to prior snapshots. If the pattern looks systemic, it writes an incident note and sends a concise report to a coordinator. If the pattern is benign, it records the observation and takes no action.

This can be autonomous even if it has no write access.

The autonomy comes from durable activation, observation, judgment, continuity, communication, and escalation. Write access may come later, but it is not the essence.

The autonomous team

A group of agents each owns a role: coordination, release, reliability, documentation, implementation, security review, public communication. Each agent has a stable identity, memory, tools, and communication surface. The coordinator can ask peers for work. Peers can report status or blockers. Some agents wake on schedules. Some react to events. Some act only when asked. Humans govern the system through policy, permissions, schedules, budgets, and escalation paths.

This is not merely a collection of agents. It is a governed multi-agent operating model.

The key difference from a multi-agent demo is durability. The team is not assembled for a single prompt and then discarded. It persists across time, retains operational context, and can be governed as an ongoing system.

Why A2A matters, but is not enough

Agent-to-agent communication is a necessary part of many autonomous systems, especially teams. But communication alone is not autonomy.

A2A gives independent agents a shared language for discovery, capability negotiation, task management, and information exchange. That matters because autonomous agents should not need to share internal memory, tools, implementation, or vendor-specific runtime details in order to collaborate.

The minimum communication requirement is simple:

- A human can reach the agent.
- Another agent can reach the agent.
- The caller can discover what the agent is for.
- The agent can accept, reject, clarify, or complete work.
- The interaction can be traced and audited.

That gives the agent a social surface. It does not give the agent a pulse, memory, tools, or governance. Those have to come from the runtime around it.

In other words: A2A makes an agent reachable. Governance and activation make it operationally autonomous.

Why MCP matters, but is not enough

MCP belongs in the autonomy discussion because tools are how agents act beyond text.

A model that can only produce text may still be useful, but its effect on the world is mediated by a human. A model that can call tools can inspect files, query systems, run tests, create tickets, post messages, execute code, or change infrastructure. That tool layer is what turns reasoning into action.

MCP is valuable because it standardizes how AI applications connect to external systems, tools, data sources, and workflows. But, again, the protocol is not the autonomy boundary. A manually commanded assistant can use MCP. An autonomous agent can use MCP. A dangerous agent can use MCP. A governed agent can use MCP.

The autonomy question is not "Does it have MCP?" The autonomy question is:

> Under what conditions may the agent decide to use the tool, what may it do with the result, and who can observe or revoke that authority?

Tool use makes autonomy consequential. Governance makes it usable.

Autonomy changes the risk model

Autonomy is powerful because it moves work out of the immediate human command loop. That is also why it changes the risk model.

A commanded agent can still make mistakes, but the human is usually near the loop. An autonomous agent may act at night, act repeatedly, act after reading adversarial input, act through write-capable tools, or coordinate with other agents. The system therefore needs stronger controls around identity, authority, observation, and escalation.

NIST's agent security work is useful because it does not treat agents as merely chat interfaces. It focuses on the risks that arise when AI model outputs are combined with software functionality. Its tool-use taxonomy also highlights questions every autonomous-agent deployment should ask:

- What action does the tool enable?
- Does the tool have read or write permissions?
- Is the environment trusted or untrusted?
- Are the effects reversible?
- Can the action be monitored?
- How much discretion does the agent have in using the tool?

Those questions are governance questions. They are also autonomy questions. The more discretion the agent has, the more important the surrounding governance becomes.

A safe autonomy model does not ask, "How do we let the agent do everything?" It asks:

> What decisions can this agent safely make without immediate human command, and what evidence must it leave behind?

The autonomy gradient

Autonomy is easier to reason about as a gradient than as a yes/no label.

Level	Name	Description
0	Tool	A function, API, script, or model call that does not decide when to run.
1	Commanded assistant	A human drives a model or agent through a session.
2	Delegated task agent	A human assigns bounded work and the agent completes it asynchronously.
3	Event-activated agent	The agent wakes on schedules, events, messages, or webhooks.
4	Governed autonomous agent	The agent observes, decides, acts, remembers, communicates, and escalates.
5	Governed autonomous team	Multiple autonomous agents coordinate through roles, protocols, and shared state.

The important leap is not from level 0 to level 1. Many teams already use powerful commanded assistants. The important leap is from direct command to governed operation.

That leap requires architecture: activation surfaces, communication protocols, tool boundaries, memory, identity, permissions, logs, and policies.

It also requires organizational maturity. Someone has to decide what the agent is allowed to decide, what evidence it must leave behind, and where human judgment must re-enter the loop.

Design principles for governed autonomy

Any system that hosts autonomous agents should optimize for governed discretion, not raw freedom.

1. Make activation explicit

Every autonomous wake-up path should be visible: heartbeat, cron, task schedule, webhook, queue, A2A request, or continuation. If the agent can wake up, operators should know why.

2. Separate observation from authority

Reading the world and changing the world should be distinct permissions. Many useful autonomous agents should start as read-only observers.

3. Treat doing nothing as a valid action

An autonomous agent that acts every time it wakes up is likely to create noise. A mature agent records why it chose not to act.

4. Scope tools to roles

Tools should match responsibilities. A documentation agent does not need production write access. A release agent does not need unrestricted cloud-admin authority. A reliability observer may need broad read access before it needs any write access.

5. Make communication protocol-based

Humans and agents should be able to reach an agent through stable interfaces. Peer agents should not need private implementation knowledge to collaborate.

6. Persist operational memory

If the agent can wake up later, it needs to know what happened earlier. Memory should be useful, inspectable, and governed.

7. Require evidence

Autonomous action should leave a trail: what triggered the agent, what it observed, what policy applied, what it did, what tools it used, and why escalation was or was not required.

8. Design for revocation

An autonomous agent's permissions, schedule, tools, memory access, and communication surfaces should be easy to pause, reduce, or remove.

What autonomy is not

Autonomy is not unlimited authority.

Autonomy is not a model quality score.

Autonomy is not the presence of a tool call.

Autonomy is not a cron job.

Autonomy is not a webhook.

Autonomy is not A2A.

Autonomy is not MCP.

Each of those can support autonomy. None of them defines it.

The defining move is the shift from immediate command to governed operation over time.

Evidence and boundaries

This paper is a synthesis, not a formal standard. It combines older agent theory, modern LLM-agent research, emerging agent protocol work, and security guidance around tool use and autonomy.

The central claim is intentionally narrow: autonomy is best understood as an operating-model property. A system becomes meaningfully autonomous when it can be addressed, activated, situated, persistent, communicative, discretionary, and governed within a role. Model quality, tool access, schedules, webhooks, A2A, and MCP can all contribute to that model, but none of them is sufficient alone.

The paper does not claim that every autonomous agent should have write access, that humans should disappear from the loop, or that autonomy is always better than commanded assistance. Many useful agents should remain commanded, read-only, human-gated, or narrow. The stronger claim is that systems should be honest about which side of the boundary they occupy, because commanded agents and governed autonomous agents need different design, risk, and trust models.

Conclusion

The useful boundary in agentic systems is no longer "agent versus non-agent." That boundary has become too blurry. The more useful boundary is **directed versus autonomous**.

A directed agent is commanded. It can be powerful, creative, and valuable, but the human remains close to the execution loop.

An autonomous agent is governed. It can wake up, observe, decide, act, remember, communicate, and escalate within a durable role. Humans still own the system, but they control it through identity, tools, policy, schedules, memory, observability, and revocation rather than through constant prompting.

That distinction matters because it changes how we design, evaluate, and trust agents. If an agent is commanded, the interface and task experience matter most. If an agent is autonomous, the operating model matters most: activation, scope, identity, authority, memory, observability, escalation, and revocation.

The future of useful autonomous agents is not unbounded machine freedom. It is governed discretion.

Sources and further reading

This paper is an interpretation of current agent research, standards work, and emerging practice. The sources below support pieces of the argument, but the commanded-versus-governed distinction is the paper's own synthesis.

- Michael Wooldridge and Nicholas R. Jennings, 1995. **Intelligent Agents: Theory and Practice**. Classic framing of agents around autonomy, social ability, reactivity, and pro-activeness.
- Stan Franklin and Art Graesser, 1996. **Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents**. Useful for the distinction between ordinary programs and situated agents that sense, act, and persist over time.
- Lei Wang et al., 2023. **A Survey on Large Language Model based Autonomous Agents**. Modern survey of LLM-based autonomous agents, including planning, memory, action, applications, and evaluation.
- NIST, 2026. **CAISI Issues Request for Information About Securing AI Agent Systems**. Useful for the security framing around planning, autonomous actions, real-world impact, and deployment controls.
- NIST, 2025. **Lessons Learned from the Consortium: Tool Use in Agent Systems**. Useful for tool-use taxonomies that include functionality, access patterns, risk, reliability, monitoring, and autonomy.
- NIST, 2025. **Adversarial Machine Learning: A Taxonomy and Terminology of Attacks and Mitigations**. Includes a section on the security of agents and the risks created when models plan and execute tool-mediated actions.
- Agent2Agent Protocol Project. **A2A Protocol Specification**. Useful for grounding agent-to-agent communication in discovery, capability negotiation, task management, and secure information exchange.
- Model Context Protocol. **What is the Model Context Protocol?**. Useful for grounding tool and external-system connectivity as a protocol layer rather than as the definition of autonomy.
- OpenAI. **Agents SDK: Agents**. Useful for current implementation vocabulary around agents, tools, guardrails, handoffs, sessions, and orchestration.