Whitepaper

Sustainable AI Agent Design: Building Intelligence We're Willing to Live With

A control-surface approach to resource utilization, autonomy, and bounded risk

Meaghan Reinecke, Director of Product Experience
Nicole Reineke, Chief AI Officer

N-ABLE™

Table of Contents

Introduction

Because of their prevalence and perceived low cost, AI agents are frequently built and deployed as "always-on," embedded into workflows, products, and platforms as a default capability that gives us unnecessary analytics and automatic actions, rather than an as-needed tool. We are forced to interact with AI agents in the products we buy and those we create, and we see their outputs littering our phones and newsfeeds. They are no longer something we experiment with occasionally or deploy only in isolated contexts. As AI agents move from "tool" to "always-on background system," their scale, persistence, and compounding effects change dramatically.

Agents are not invoked and dismissed like traditional automation. They are often set to run continuously, make decisions autonomously, and operate without direct human prompting. In 2026, 97% of 1,900 global IT leaders are exploring agentic strategies, and 49% have graduated their agentic AI projects from pilot to production[1]. This raises two questions: who is teaching these companies good design for AI agents, including the software lifecycle, security, governance, support, and the overall impact on our ability to manage the expanse of AI impact (also referred to as the blast radius)? More controversially, what is the impact of these AI agents on a world of finite resources like water, energy, and compute? These issues are completely intertwined.

The question of intentionally architected AI agents can no longer be treated as an abstract infrastructure concern; it must become a design imperative. The operational footprint of AI is no longer determined solely by how consumptive our models are, but by how often, how broadly, and how carelessly AI agents are invoked and force models to respond.

[1] "The State of AI Development 2026: Navigating the Shift to Agentic AI," OutSystems, 2026. https://www.outsystems.com/1/state-ai-development

The blast radius of AI agent consumption

We often rely on token utilization as a proxy for understanding AI cost. While token usage is closely related to model inference and, therefore, environmental impact, this measure is incomplete. It does not capture infrastructure overhead, orchestration complexity, retries, polling behavior, or the hidden costs of idle but active systems. Over time, cost accrues not just through compute but also through operational burden, future maintenance, and governance overhead. These costs are typically underestimated and difficult to unwind.

With the emergence of global standards for agent safety, security, and reliability[2] (AIUC-1), we have developed foundations for secure AI agent generation, which are exceptional and should be followed. Part of this does state limiting scope. The industry has also introduced the concept of Green AI to refer to a collection of behaviors and outcomes that reduce unnecessary token usage and increase efficient utilization.[3] However, these concepts must be both combined and expanded to create trustworthy AI that is efficient and useful. We must address the impact of AI agents on the software lifecycle, security, governance, support, and the overall impact on our ability to manage the full scope of the blast radius, growing at an exponential rate.

There is little argument on why this issue is critical now due to economics. Total organizational compute expenditure is rising sharply, driven by the volume and frequency of agent execution even as per-unit costs fluctuate[4], and hourly agent execution has increased. Gartner suggests that, by 2030, the cost per resolution for AI-powered customer service will exceed that of offshore human agents, signaling a broader inflection point in AI cost economics[5]. These AI-consumed physical and energetic resources also serve as a proxy for measurable environmental impact[6]. AI model hosting locations, particularly in large cloud data centers, have tangible consequences for electricity demand, water usage, land utilization, and electronic waste. These realities force a new question, not just about what AI can do, but about what it should be doing continuously and where it should be running, as measured against the resource impact.

[2] "The standard for AI agent security, safety and reliability," AIUC-1. https://www.aiuc-1.com/

[3] Referencing Schwartz et al., 2020. Rojahn, M. & Grum, M. (2025) "Green AI: A system review and meta-analysis of its definitions, lifecycle models, hardware and measurement attempts," Arxiv. Computer Science, Cornell University. https://arxiv.org/abs/2511.07090

[4] Noffsigner, J. et al. (2025) "The cost of compute: The $7 trillion race to scale data centers." McKinsey & Company. https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/the-cost-of-compute-a-7-trillion-dollar-race-to-scale-data-centers

[5] *Gartner, Press Release, Gartner Predicts GenAI Cost Per Resolution for Customer Service Will Exceed Offshore Human Agent Costs by 2030, January 2026, https://www.gartner.com/en/newsroom/press-releases/2026-01-26-gartner-predicts-genai-cost-per-resolution-for-customer-service-will-exceed-offshore-human-agent-costs-by-2030

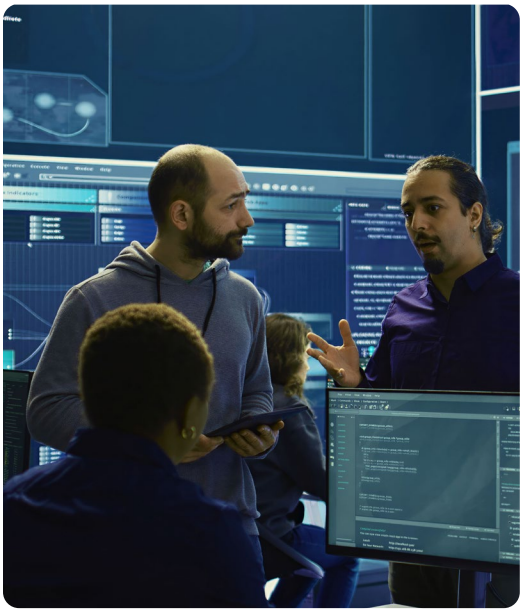
[6] Ord, T. (2025). "Are the costs of AI agents also rising exponentially?" https://www.tobyord.com/writing/hourly-costs-for-ai-agents

The hidden cost problems with AI agents

From a user perspective, classic AI interactions such as those with Claude Code or ChatGPT feel lightweight. The user has an awareness of when it is being called, and typically, it is the user who invokes the action. But AI agents fundamentally change that perception. Agents can be built to create long-running autonomy, often executing even when no one is actively watching, consuming resources continuously rather than episodically. This abstracts the AI invocation and disembodies the act. Now, AI is running and consuming resources, without a person considering its broader world impact.

This leads inexperienced teams to default toward "agent the whole thing" and "run often" instead of designing narrowly scoped, surgical agents. The result is unnecessary compute running in places where simpler, deterministic solutions would suffice. In practice, teams frequently default to always-on configurations, treating continuous execution as the path of least resistance. Once introduced, this infrastructure is rarely removed; it accumulates, compounds, and becomes embedded into how systems operate.

What if we, as designers, made the AI agent creation default to efficiency and trustworthiness through narrow scope and restrained utilization?



A case for expanding and templatizing trustworthy, efficient AI

This is where Green AI provides a necessary reframing. Rather than optimizing solely for performance or capability, Green AI emphasizes transparency across the full AI lifecycle—from design and training through deployment, execution, and retirement[7]. Interestingly, this leads us to a model of AI agent consumption in which we combine both deterministic and nondeterministic behaviors to reduce unnecessary resource consumption, and, in practicality, drives us to better-performing products with a higher degree of accuracy and successful outcomes.

Green AI treats energy, carbon, water usage, and resource consumption as first-class system properties rather than externalities to be addressed later. It also accounts for second-order effects on waste reduction, as well as third-order impacts on behavior and consumption patterns. With a holistic approach to Green AI, we can begin with how we design (and expect outcomes from) AI agents.

By applying these principles consistently across the lifecycle of AI systems, teams can make informed trade-offs rather than accidental ones.

[7] Chouskey, A. et al. (2026). "The green paradox: The climate, environmental, and sustainability implications of artificial intelligence." Global Environmental Change Advances, vol. 6. https://www.sciencedirect.com/science/article/pii/S2950138525000178

From red to green

In this whitepaper, we will use "Red AI" as a conceptual counterpoint to Green AI. It is not an established industry term, but it serves as a useful shorthand for the patterns we seek to move away from.

Traditional AI development—or Red AI—rewards bigger models, higher benchmark scores, and increasingly complex reasoning regardless of cost. This mindset has driven enormous progress, but it has also normalized inefficiency as a sign of advancement.

Green AI proposes a different signal. Instead of maximizing intelligence everywhere, it values proportional, contextual intelligence, using the right level of AI computation for the task at hand.

Green AI Sustainable AI through design choices	Red AI Maximized performance and always-on
Lower resource consumption and carbon emissions	High cost, high carbon footprint
Balances performance with sustainability	Prioritizes performance over sustainability
Values proportional, contextual intelligence	Applies "always-on, all the time" design

In this framing, Green AI is not a constraint or a compromise; it is a signal of design maturity. A system that can achieve its goal with fewer resources is better designed and frequently drives to more successful deterministic behavior. Determinism is also a significant measurement of when to use traditional code instead of AI.

If you need predictable, repeatable outcomes and clearly understand the rules that determine the result, use deterministic approaches such as scripts, workflows, or direct API calls. These systems are cheaper, faster, and easier to validate.

AI is better suited for nondeterministic work. This includes exploratory tasks, open-ended questions, and situations where the right next step depends on context, nuance, or incomplete information. The right pattern is to separate thinking from doing. Use AI to reason, interpret context, and recommend actions. Then, hand execution to deterministic, preapproved mechanisms once the path forward is clear.

In practice, that means:

- The nondeterministic layer, typically an LLM or agent, is responsible for understanding the situation and determining what should happen next.
- Actual execution is performed by deterministic scripts or code that have already been reviewed and approved, such as calling APIs directly.

This approach keeps AI where it adds the most value while preserving reliability, cost control, and safety in execution.

This approach has four key benefits:

1. Eliminates token overhead from having the agent interpret and execute scripts directly
2. Removes write access from the nondeterministic layer, reducing risk
3. Builds user trust, making adoption easier by constraining where unpredictability exists
4. Implements Green AI principles by minimizing unnecessary token usage: AI is only used where reasoning adds value, while routine execution is handled deterministically, reducing compute consumption, cost, and environmental impact

This shift raises an important question: how does diligence around managing AI across an estate encourage—or discourage—Green AI thinking? The way tools are selected, defaulted, and governed can either entrench Red AI behavior or make room for more sustainable patterns to emerge.

How Green AI shows up in design

There are at least five major patterns in AI agent design that we can use to generate high-value outcomes and reduce our impact on finite resources. These include the utilization frequency, the scope of AI use, the choice of execution location and compute routing, lifecycle and governance of the tooling, and our choice of where to run actions and tooling for the outcomes.

Let's examine each in detail.

Execution frequency and scheduling

Most AI costs are exacerbated by unnecessarily calling models to run. Polling loops, short schedules, and always-on monitoring quietly multiply inference, API calls, storage, and noise. That is bad for consumption. It is also detrimental for trust, as users stop believing the agent is intentional.

Many agent systems are built like old monitoring tools. They poll constantly because it is easy.

That creates predictable waste:

- **Continuous cycles:** The agent runs even when nothing has changed
- **Polling storms:** Lots of low-value checks that add up across tenants
- **Operational noise:** Alerts and logs inflate, and users learn to ignore them
- **Peak time waste:** Agents burn cycles during business hours when they are least useful

Let builders define when an agent runs. Prefer event triggers and state changes over polling. Enforce sleep windows so that idle equals off.

Expose a scheduling and trigger model that governs three decisions:

1. **Run cadence:** On-demand, scheduled, or event-triggered
2. **Trigger conditions:** State change rules in lieu of polling
3. **Sleep windows:** Active hours and blackout periods so that idle truly means off

This turns "when does it run" into an intentional design decision instead of a background process that never stops.

Controls

Run cadence selection

Let builders choose how and when the agent activates:

- **On-demand**
  - User-initiated or API call
  - Best for investigation, remediation, and interactive workflows
- **Scheduled interval**
  - Every X minutes or hours
  - Use for batch tasks, reporting, or slow-moving states
  - Default should never be continuous
- **Event-triggered (recommended default when available)**
  - Fires on a defined state change
  - Best for compliance, inventory, security signals, and queue-driven work

Trigger conditions, not polling

If the change is infrequent, expose a condition builder that expresses "run when this changes," not "run every 15 minutes."

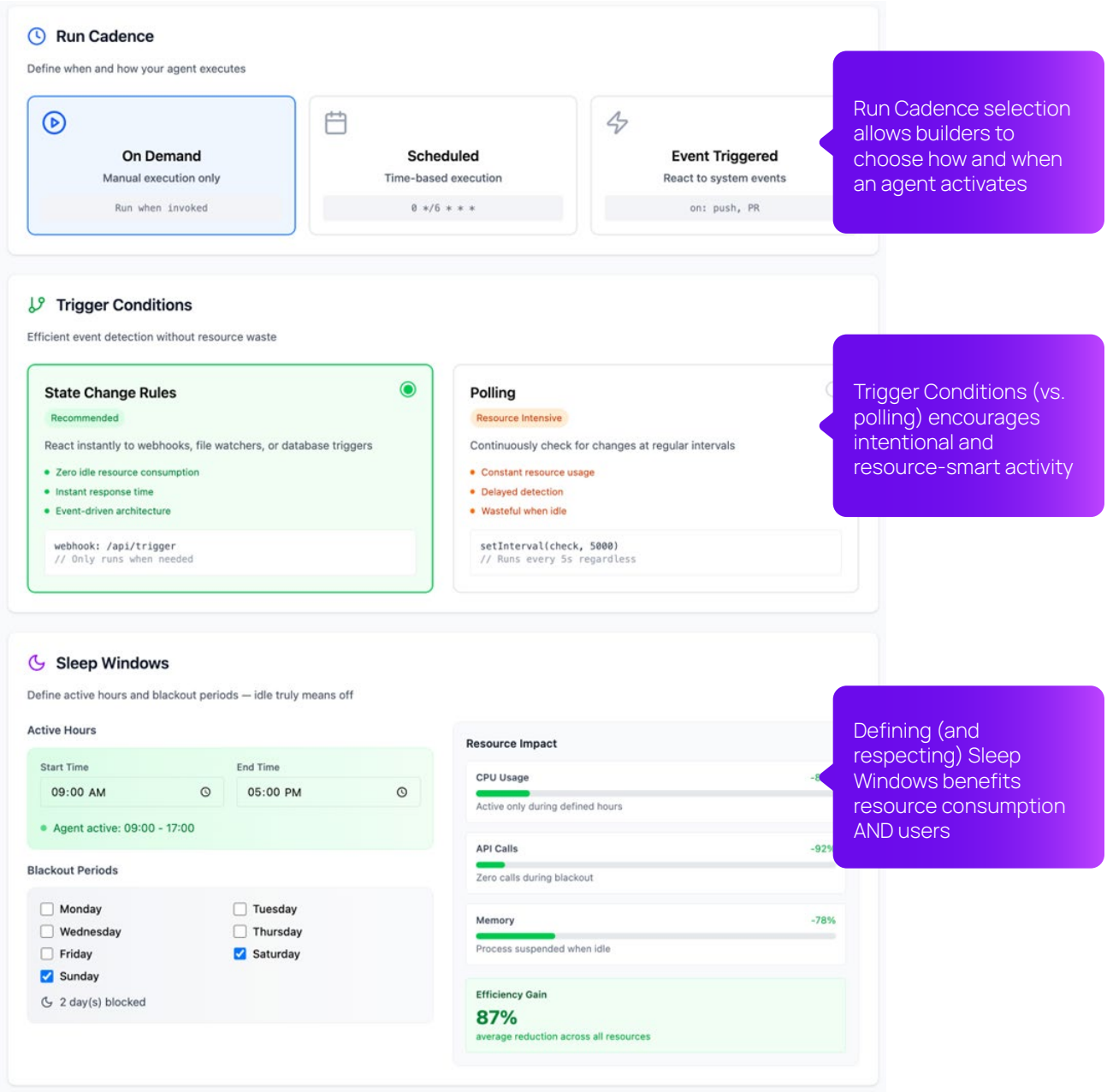
Sleep windows

Let builders define when the agent is allowed to run. Sleep is both a governance and energy control.

- **Active hours**
  - Define time ranges when the agent may run
  - Example: After-hours monitoring agent runs only from 6 p.m. to 6 a.m.
- **Blackout windows**
  - Never run during defined periods
  - Example: Business hours, patch freeze windows, billing close

- **Maintenance windows**
  - Allow actions only during approved times
  - Example: Remediation runs only on Saturdays, 1–5 a.m.
- **Queue behavior while sleeping**
  - Drop, defer, or batch triggers
  - Define max backlog and expiration rules

Consider these types of controls while creating agents (illustrative estimates):



Suggested friction

When designing the AI agent setup, adding a few points of friction can help guide users to default to the desired behavior. For this group, we can:

- Require a justification for continuous schedules or very tight intervals
- Warn when polling is configured but an equivalent trigger exists
- Surface estimated run volume and cost before enabling

Sidebar: Red AI vs. Green AI

Green AI Sustainable AI through design choices	Red AI Maximized performance and always-on
Uses event triggers and state changes	Polls constantly even when nothing changes
Makes always-on an explicit choice	Defaults to tight schedules and continuous loops
Respects active hours and maintenance windows	Runs during business hours without purpose
Uses batching and rate limits	Spams logs and alerts
Logs the reason a run happened and what it produced	Cannot explain why it ran

Key design checklist

✓	I can choose on-demand, scheduled, or event-triggered runs
✓	My default is not continuous
✓	I use trigger conditions based on state changes, not timer polling
✓	I have debounced and de-duped to avoid flapping and repeats
✓	I have rate limits and backoff to prevent storms
✓	I use sleep windows so that idle equals off
✓	I separate analysis windows from remediation windows
✓	I log run reason, run volume, and suppressed triggers for tuning

Scope and context boundaries

AI waste also comes from unbounded calls. Agents with no context limits, scope boundaries, and visible model choice tend to expand. They ingest more data than needed, touch more systems than required, and default to the largest available model because nothing stops them.

Addressing this requires exposing a control surface that governs three related decisions:

1. **How much context can the agent consume per run?**
2. **What data surface is the agent allowed to access?**
3. **Which model is appropriate for this specific task?**

Together, these controls create more deliberate design choices that dramatically reduce cost, energy, and risk. We can limit how much the agent can see, where it can look, and which model it can use, so every inference is intentional and proportional to the task.

Many AI systems quietly inherit the worst defaults:

- **Unlimited context windows** simply because the model supports them
- **Over-broad scopes** like "entire tenant" or "all devices" for narrow tasks
- **Model mismatch** where generative or general-purpose models are used for classification, tagging, or scoring
- **Invisible trade-offs** where builders never see the energy, cost, or efficiency implications of their choices

This creates agents that are powerful but sloppy. They cost more than expected and are harder to govern, audit, and explain.

Controls

Context window budget

Expose a hard cap on how many tokens an agent is allowed to consume per run.

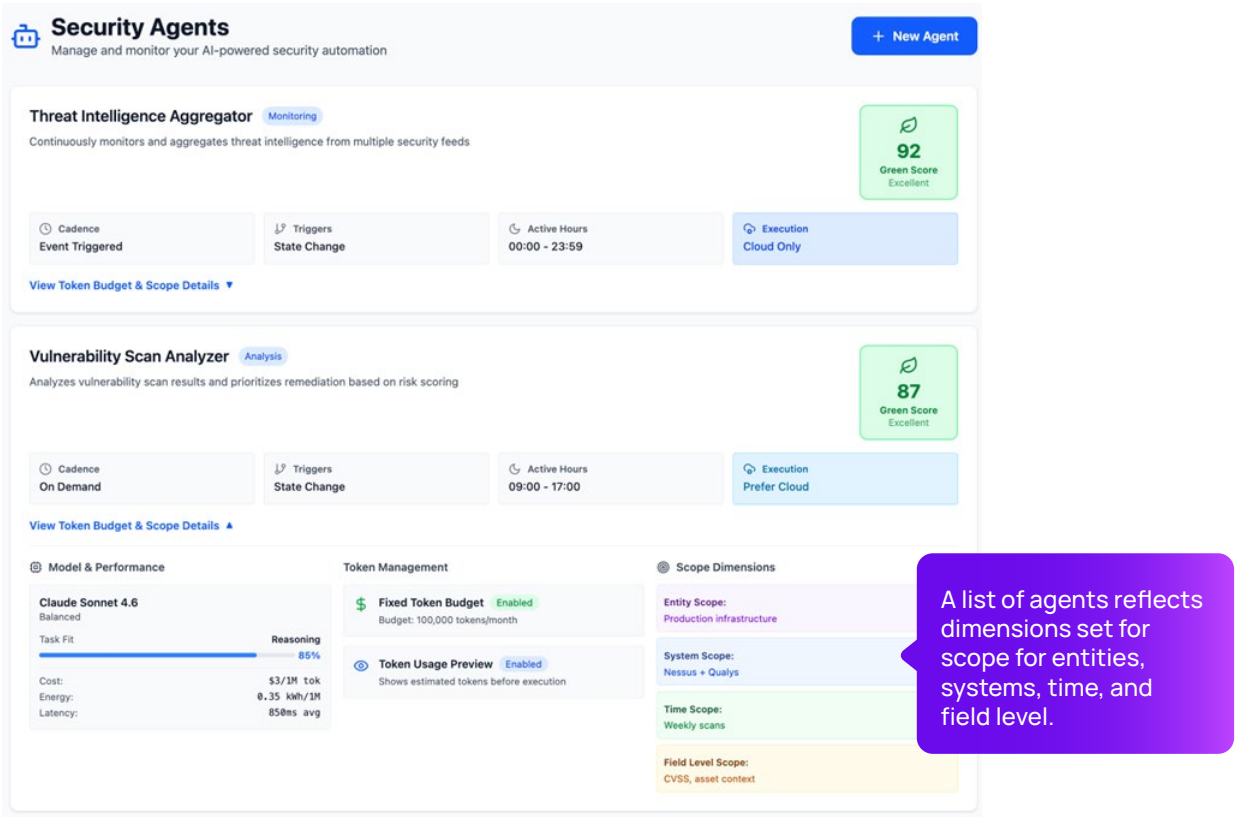
- **Fixed token budget per run**
  - Example: 2,000 tokens max, regardless of model limits
- **Dynamic budget by task type**
  - Classification: Small budget
  - Explanation or synthesis: Larger but capped budget
- **Failure behavior**
  - Truncate oldest context
  - Skip low-priority attachments
  - Abort and log budget exhaustion
- **Preview and estimation**
  - Show estimated token use before enabling
  - Warn when prompts or tools exceed the budget

Scope limiter

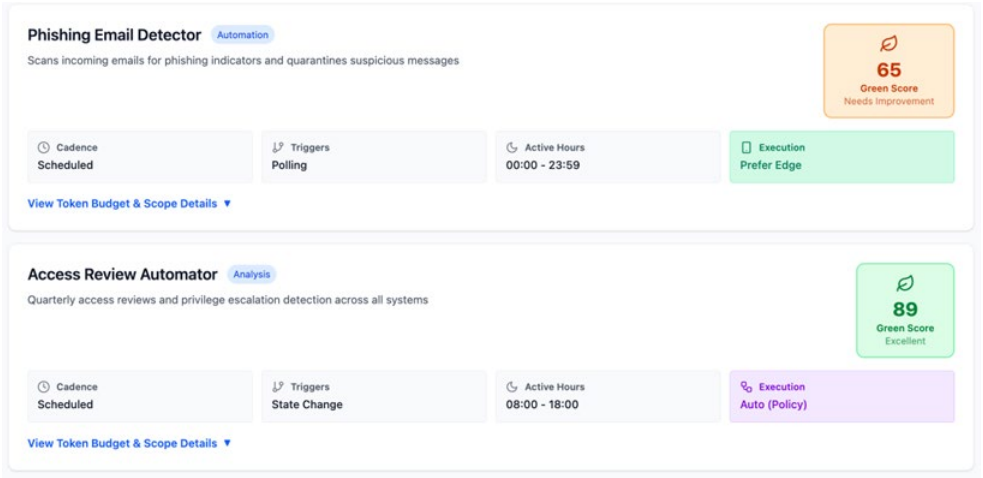
Scope dimensions

Entity scope ◢ One device ◢ One user ◢ One tenant ◢ One site or folder	**System scope** ◢ Specific products or tools only ◢ Read-only vs. read-write
Time scope ◢ Last x-hours or days ◢ Current state only ◢ Explicit historical windows	**Field-level scope** ◢ Allow or deny specific attributes ◢ Example: hardware metadata allowed, user notes denied

Consider, when reviewing a list of agents, we could assess their criteria for resource consumption (illustrative estimates and scores):



A list of agents reflects dimensions set for scope for entities, systems, time, and field level.



Task-specific model selector

Expose model choice as a visible, explicit decision tied to the task, with cost and efficiency indicators.

- **What model selection UI should show**
  - Model name and type
  - Task fit (classification, extraction, generation, reasoning)
  - Relative cost
  - Relative energy use
  - Latency expectations

- **Policy guidance**
  - Classification, tagging, sentiment analysis, anomaly detection
    - Default to fine-tuned or task-specific models
  - Deterministic or low-entropy tasks
    - Prefer nongenerative or encoder-only models
  - Open-ended generation, synthesis, or explanation
    - Require justification for large generative models

Suggested friction

- Require justification for unlimited or cross-tenant scopes
- Warn on large model usage for what looks like small tasks
- Surface estimated cost and carbon impact before enabling

Sidebar: Red AI vs. Green AI

Green AI Sustainable AI through design choices	Red AI Maximized performance and always-on
Enforces hard context budgets	Uses maximum context because it can
Narrows data access deliberately	Pulls broad data “just in case”
Matches model choice to task type	Defaults to large generative models
Makes trade-offs visible	Hides cost and energy trade-offs
Can explain exactly what it saw and why	Is difficult to audit or explain

Key design checklist

✓	I set a hard context window budget per run
✓	I do not default to the model’s maximum context
✓	I define the smallest viable data scope
✓	I restrict scope by entity, system, time, and field
✓	I select models based on task type, not power
✓	I expose cost and efficiency indicators in the UI
✓	I log actual context usage and scope access for tuning

Execution path and compute routing

Not every task deserves a large cloud model. Many steps in an AI-powered workflow are predictable, repeatable, and well understood. Others require contextual reasoning or exploration. Where we have edge devices that can run models, we should help people choose the lowest-cost, lowest-energy execution that still gets the job done.

Most agent frameworks make it easy to call a hosted large model and hard to avoid calling a model. This creates predictable waste:

- **Inference overuse:** Model calls for tasks that are deterministic
- **Cloud by default:** Heavyweight remote compute for routine steps
- **Hidden costs:** Latency, bandwidth, and repeated calls that compound over time

The Green AI approach: Make execution a deliberate product choice with a clear ladder of options:

1. Deterministic execution (rules, scripts, lookups)
2. Local or edge model execution (small models, close to data)
3. Cloud model execution (large models, only when it earns its keep)

Controls

Execution options

Let builders define where agent execution is allowed to run based on cost, latency, privacy, and policy requirements.

- **Edge only**

  Execution is restricted to local or on-device environments. No cloud calls permitted.
- **Cloud only**

  All execution occurs only in approved cloud environments. Edge execution is not allowed.
- **Prefer edge**

  Default to edge execution when available; fall back to cloud only if the required capabilities are missing.
- **Prefer cloud**

  Default to cloud execution; use edge only when cloud execution is unavailable or constrained.
- **Auto, policy-driven**

  Execution location is selected dynamically based on policy signals such as data sensitivity, cost thresholds, latency needs, or regulatory constraints.

Trade-offs and when not to use this pattern

This pattern is powerful, but it comes with real operational and governance costs. It is not the right choice for every workload.

Edge challenges

- **Model updates and version sprawl**

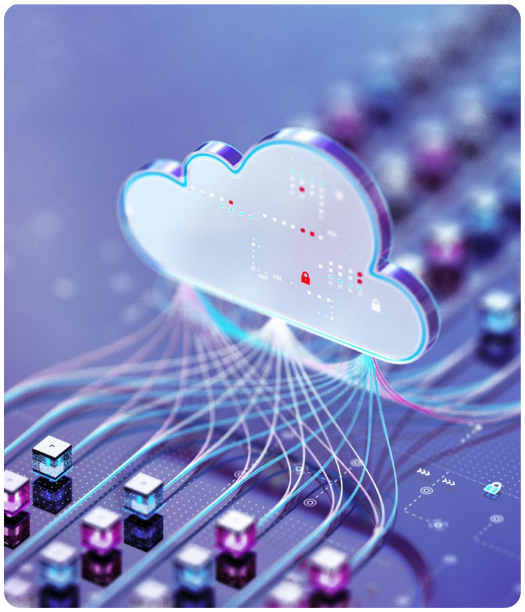
  Keeping models current across many distributed endpoints increases operational overhead and risk of drift.

- **Monitoring and observability**

  Gaining consistent visibility into performance, failures, and behavior across edge nodes is significantly harder.
- **Hardware constraints**

  Very small or specialized devices may lack the memory, compute, or acceleration required for reliable inference.
- **Governance complexity**

  Each endpoint effectively becomes its own inference environment, increasing the blast radius for policy and compliance gaps.



Cloud advantages

- **Centralized updates and consistency**

  Models and policies can be updated once and applied uniformly.
- **Greater compute headroom**

  Enables larger models, longer context windows, and more advanced reasoning.
- **Simpler controls and auditing**

  Global rate limiting, logging, and audit trails are easier to enforce and maintain.

When a cloud-first approach is reasonable

- **Long-context workloads**

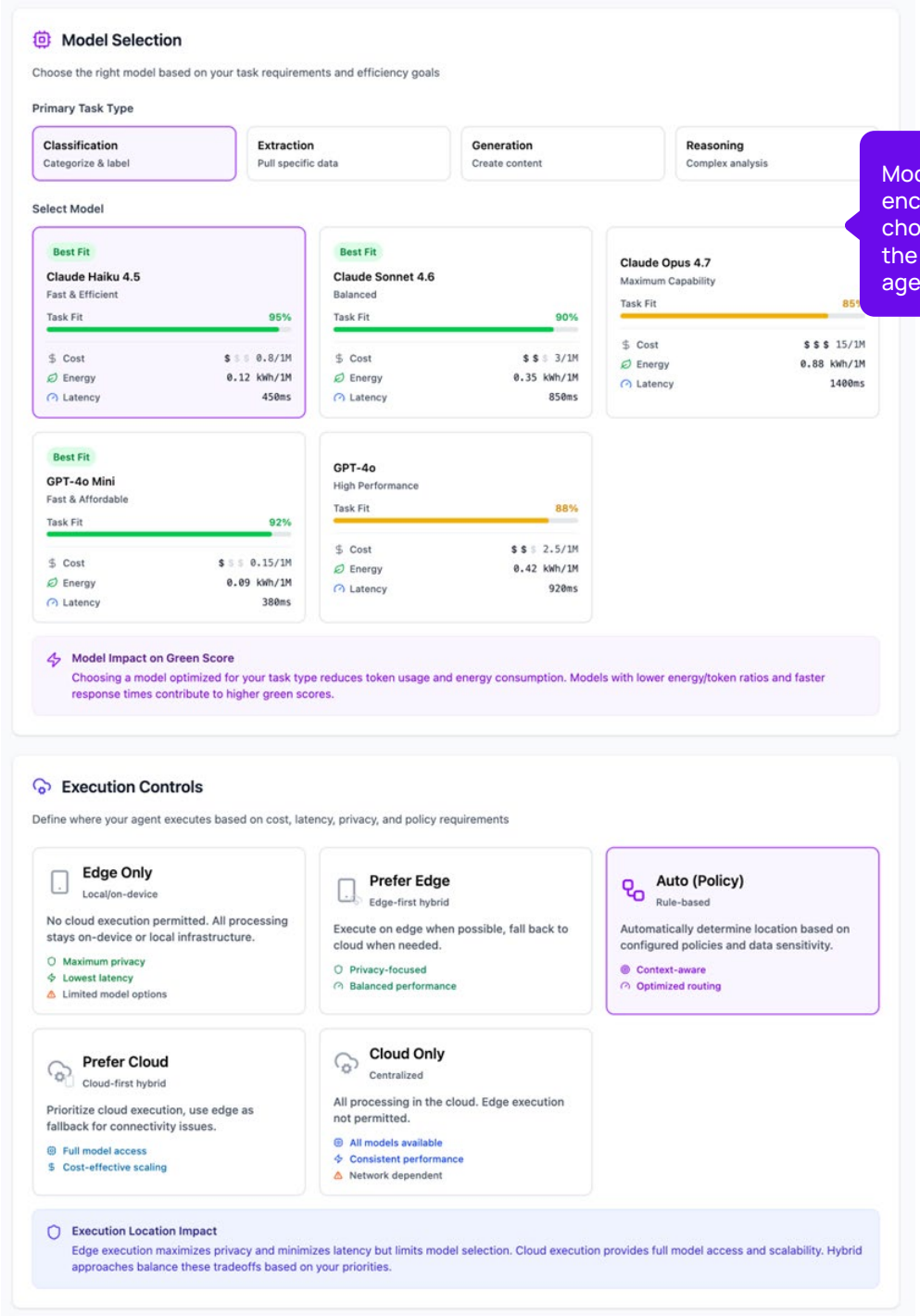
  Tasks that require large context windows or extensive historical data.
- **Deep multisource reasoning**

  Tasks that combine and reason across many systems or datasets.
- **Rich conversational experiences**

  Scenarios where users explicitly want detailed, explanatory, or interactive responses.
- **Constrained or unreliable edge environments**

  Situations where edge devices cannot reliably support inference or execution.

To really support responsible and resource-conscious agent design, consider controls for the model selection and execution controls (cost, energy, and savings are illustrative):



Model selection encourages intentional choices according to the type of task for the agent.

Sidebar: Red AI vs. Green AI

Green AI Sustainable AI through design choices	Red AI Maximized performance and always-on
Edge first for routine tasks	Always call cloud models, even for routine steps
Smallest model for the task	No user-visible placement control

Key design checklist

✓	I can choose edge, cloud, or hybrid per agent step
✓	I have a clear default that favors low energy inference

Lifecycle and governance

AI agents are living systems that need an end date, an audit trail, and a clear stop condition. A lot of AI waste and risk happens after deployment. Agents linger long past their usefulness, and over time, "helpful automation" quietly becomes background load.

We should be asking three questions continuously:

1. Does this agent still deserve to exist?
2. What has this agent been doing and costing over time?
3. When should the agent stop and ask for human help instead of retrying?

Together, these controls prevent infinite life, invisible drift, and runaway inference loops.

When agents have no forced review point and no visible operating record, three things happen:

- **Zombie agents** continue running after the original need is gone
- **Scope creep** increases execution frequency and model usage without notice
- **Low-confidence loops** retry and re-infer instead of escalating to humans

The longer an agent runs unattended, the harder it is to explain or justify.

Controls

Scheduled decommission date

Every agent gets an expiration date at creation.

Capabilities

Required expiration date ◢ Example: 30, 60, or 90 days by default	**Renewal workflow** ◢ Builder must actively extend the agent's life ◢ Renewal requires reviewing usage and outcomes
Ownership requirement ◢ Every agent has a named owner responsible for renewal	**Grace and shutdown behavior** ◢ Warn before expiration ◢ Disable execution after expiration ◢ Optional archival of configuration and history

Run history and cost ledger

Surface a per-agent execution and cost record that is impossible to ignore.

Ledger contents

- Number of executions per day, week, and month
- Estimated token consumption over time
- Relative compute cost by execution path (deterministic, edge, cloud)
- Outcome distribution (success, escalated, stopped by confidence)
- Trigger source (event, schedule, on-demand)

Suggested friction

- Block renewal if the agent has not been reviewed
- Warn when run volume or cost deviates materially from creation estimates
- Escalate repeated low confidence stops to the owner

Sidebar: Red AI vs. Green AI

Green AI Sustainable AI through design choices	Red AI Maximized performance and always-on
Agents expire unless renewed	Agents run indefinitely
Ownership is explicit	No clear owner
Run history and cost are visible	Costs are hidden or aggregated elsewhere
Low confidence stops execution	Low confidence triggers retries or bigger models
Every agent can justify its existence	No clear explanation for why it is still running

Key design checklist

✓	Every agent has the required expiration date
✓	Every agent has a named owner responsible for renewal
✓	I can see the run count and cost per agent over time
✓	I can detect drift early using the ledger
✓	I set confidence thresholds per task
✓	Low-confidence execution stops rather than retries
✓	I log confidence and stop reasons for review

Output & action constraints

Many of the most damaging and expensive agent failures happen after a decision is made. An agent executes too freely, talks too much, or spawns other agents, creating a cascade no one intended and noticed until the bill arrived.

The Green AI approach: Limit action, output, and propagation by default. Make autonomy, verbosity, and recursion explicit choices with clear costs and guardrails.

Expose one control surface that governs these related questions:

1. Is this agent allowed to act, or should it recommend instead?
2. How much output is required to do the job?
3. How far is this agent allowed to fan out into other agents or workflows?

These constraints dramatically reduce blast radius, token waste, and invisible exponential compute growth.

Default agents to recommend, be concise, and stay shallow. Require deliberate approval to act, explain at length, or expand execution chains.

When agents lack output and action constraints, four predictable failure modes appear:

- **Over-autonomy:** Agents execute changes without human oversight
- **Over-verbosity:** Long explanations where a label or flag would suffice
- **Chain explosions:** Agents spawn agents, multiplying inference cost rapidly
- **Invisible risk:** Blast radius grows faster than observability

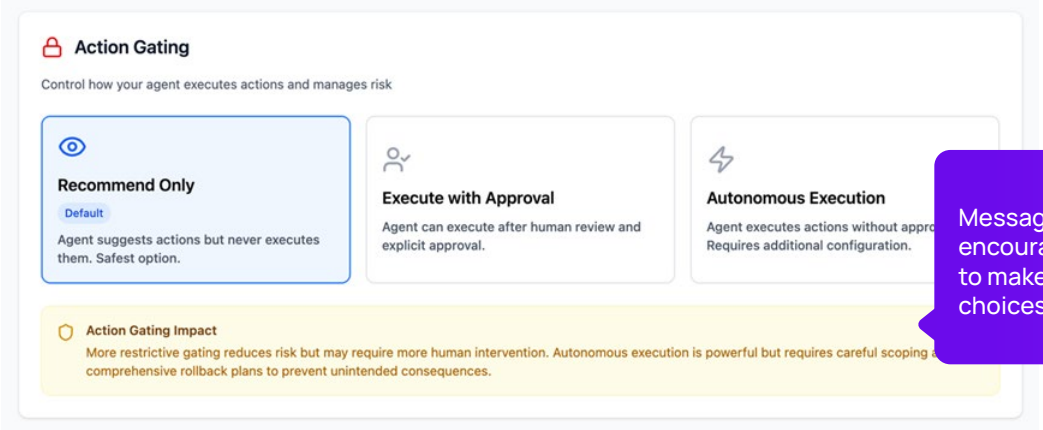
Controls

Action gating: recommend vs. execute

Expose a clear, unavoidable toggle for agent behavior.

Recommend only (default) - Agent analyzes and proposes actions - Human approval required before execution	**Execute with approval** - Agent submits action request to a queue - Execution happens only after sign-off

Autonomous execution
- Agent acts immediately
- Requires explicit unlock and scope definition
	Design rules
	- Autonomous execution is never the default
	- Unlocking autonomy should require:
	  - Named owner
	  - Scoped action set
	  - Explicit rollback or recovery plan



Output length cap

Set a maximum response or output size per run.

Capabilities

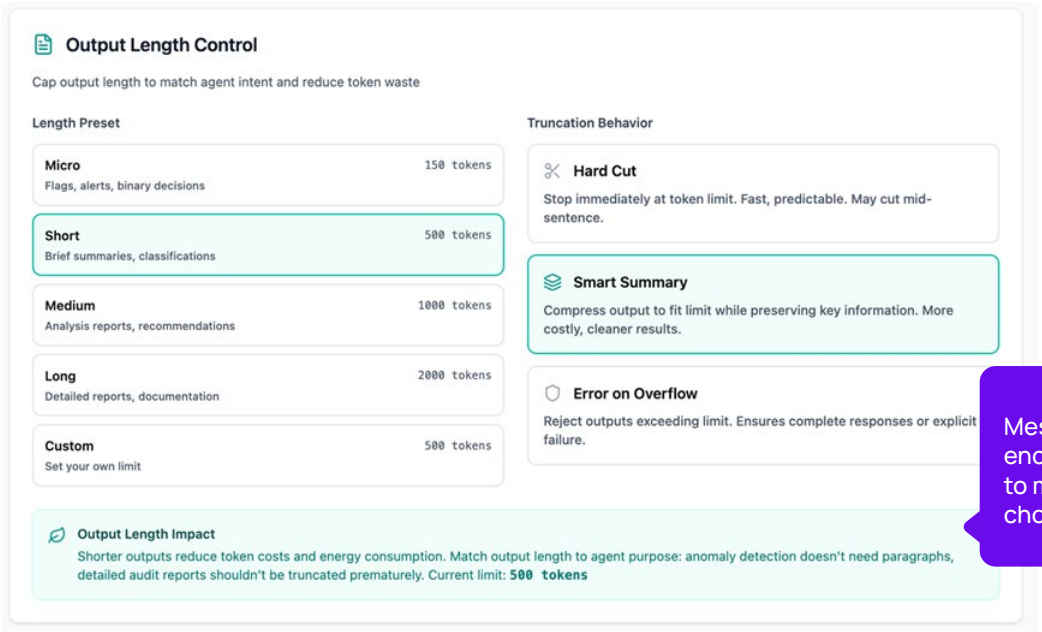
Hard length cap	Task-based presets	Truncation behavior
◢ Tokens or characters per outputrequired before execution	◢ **Classification:** One label ◢ **Detection:** Flag plus identifier ◢ **Recommendation:** Short list only	◢ Summarize instead of spilling context ◢ Drop explanations once the cap is reached

Design principle: Output should match intent.

If the job is to flag an anomaly, the output is a flag and a device ID. Not a paragraph.

Verbose output costs twice: Once to generate and again to process downstream.

Consider these controls when providing recommendations for output length:



Messaging throughout encourages builders to make intentional choices and trade-offs

Dependency chain depth limit

In multi-agent or chained workflows, cap how far execution can fan out.

Capabilities

Maximum call depth ◢ Example: depth of 2 or 3 by default	**Approval required to exceed** ◢ Manual override or governance rule
Chain visibility ◢ Show full invocation path in logs	**Fail-safe behavior** ◢ Stop and flag when depth is reached ◢ No silent continuation

An agent that spawns subagents that spawn subagents compounds compute cost exponentially. This is one of the least visible, most expensive Red AI behaviors because each step looks "reasonable" in isolation.

Suggested friction

- Require justification to enable autonomy
- Warn when output regularly hits the cap
- Alert when chains approach depth limits
- Defaults should actively prevent accidental overreach

Sidebar: Red AI vs. Green AI

Green AI Sustainable AI through design choices	Red AI Maximized performance and always-on
Recommends by default	Executes by default
Keeps outputs minimal	Explains everything
Caps chain depth	Spawns agents without limits
Makes autonomy visible	Hides blast radius
Expands only with intent	Multiplies cost quietly

Key design checklist

✓	My agents default to recommending, not executing
✓	Autonomous execution requires explicit unlock
✓	I cap output length per task
✓	Outputs match job intent, not verbosity preference
✓	I cap dependency chain depth
✓	Deeper chains require approval
✓	I log action mode, output size, and chain depth per run

Optimizing agents is good agent design practice

None of this work is orthogonal to good AI design. It is good AI design. When builders see the cost of their choices before an agent ever runs, they design more carefully. Sustainable agents tend to be legible, bounded, observable, and degradable, which are exactly the properties needed to manage AI safely across a large, distributed estate.

Effective AI agents are designed with the following principles in mind:

- The agent must be auditable
- Humans must be able to understand and explain its actions
- It should be verbose and transparent in how it operates
- Its scope should be very narrow
- Strong guardrails must always be in place

Organizations that consider lifecycle management and agent decommissioning from the outset are better positioned to scale AI responsibly. Narrow scopes enable offloading work into tools, reduce ambiguity around behavior, and make it easier to turn systems off when they no longer serve a purpose. These principles (auditability, explainability, transparency, narrow scope, and strong guardrails) are as much about quality and trust as they are about sustainability.

Conclusion: Sustainability is good architecture

Modern product culture tends to reward adoption: more automation, more intelligence, and more surfaces touched. Green AI asks for something countercultural: restraint. This means letting agents sleep, choosing automation only where it meaningfully improves outcomes, and defaulting to "off" rather than "always on."

Restraint is not a lack of ambition. It is a signal of confidence that we understand our systems well enough to limit them, as well as maturity, enough to prioritize long-term health over short-term velocity. The open question is how we design environments that recognize and reward this restraint as strength rather than hesitation.

A useful closing question is simple yet powerful: what do we hope future teams thank us for? The intelligence we build today will persist long after the teams that created it move on. The sustainability conversation around AI is often framed in terms of harm reduction. Green AI invites a deeper shift. It becomes an act of intention rather than optimization—a balance between ambition and responsibility. Ultimately, it asks us to build intelligence we are genuinely willing to live with.

N-able protects businesses from evolving cyberthreats. Our AI-powered cybersecurity platform delivers business resilience to more than 500,000 organizations worldwide, leveraging advanced end-to-end capabilities, simplified workflows, market-leading integrations, and flexible deployment options to improve efficiency and drive critical security outcomes. Our partner-first approach pairs our technology with experts, training, and peer-led events that empower customers to be secure, resilient, and successful. n-able.com

---