Sovereign AI Requires Practical Control

By: Christopher Hume, Chief Business Officer, Legion Intelligence

Sovereign AI should give an organization the authority and practical means to govern its information, direct automated work, adapt its applications, and choose the dependencies it accepts. That control should leave the organization more capable of delivering and improving its own operations.

Sovereignty Must Extend to the Work

A procedure changes, and someone has to update the AI application that carries it out.

Consider a fleet maintenance supervisor working to return a vehicle to service. An AI application gathers the technician's fault report and repair history from the maintenance system, retrieves the approved technical procedure, checks parts availability, and prepares a proposed work order. The supervisor reviews the work order, and a qualified maintainer carries out and verifies the repair.

Then a revised procedure adds an inspection before the vehicle can return to service. The application must now require the inspection record and route it for approval. The organization needs to update the workflow, test the new behavior, and preserve a record of which procedure and approvals governed each job.

Can the customer make that change using its own staff or a chosen partner? Can it change the underlying model while retaining the workflow and its controls? At a remote site, which parts of the process can continue when external services are unavailable? These questions make sovereignty concrete: the organization needs the practical ability to govern and adapt the system on which the work now depends.

Market definitions already extend beyond national infrastructure. IBM includes both organizations and nations in its definition of AI sovereignty; Microsoft describes sovereign AI in terms of controls over data, access, governance, infrastructure, and operations. For companies, military organizations, and public agencies, the question is how those controls operate inside an application that people depend on. [1, 2]

We propose the following standard:

Sovereign AI is an organization's practical ability to govern how AI is deployed, used, and changed: controlling how its sensitive information and operational knowledge are handled, what actions AI may take, and which external dependencies it accepts.

For a business, this is enterprise AI sovereignty. Institutional AI sovereignty extends the same principle to military organizations and public agencies. Its purpose is to let organizations improve applications they already depend on and reuse successful work as new requirements emerge.

This standard makes room for commercial software, managed infrastructure, and frontier models. An organization can delegate implementation and operation while retaining authority over the requirements and decisions that matter. The necessary controls will vary with the work, its sensitivity, and the consequences of failure.

The word practical carries much of the definition. A contractual right to make a change has limited value if the customer lacks the tools, documentation, access, or support to exercise it. Sovereign AI should make that authority usable throughout the life of the application.

Control Must Follow Information Into Action

A single request can move information across several systems, a model, and a tool.

Control has to follow information as an AI application brings it together from several systems, passes it to a model, and uses the result to initiate another step. The organization's rules must hold across that entire sequence. Permission to retrieve a record does not automatically confer permission to send it to an external model, disclose it to another user, or act on its contents.

A sovereign AI system should make those boundaries explicit. It should establish which information each person and agent may access, which models may process it, which tools may be invoked, and which actions require approval. When a request falls outside the approved scope, the system needs a defined response: stop, seek authorization, or return the work to a person.

Data control also extends to the material an AI system creates along the way. Prompts, retrieved passages, search indexes, outputs, logs, and support records can contain sensitive context. Oracle's published sovereignty guidance explicitly includes these derived and operational assets. Their location, access, retention, and permitted uses deserve the same deliberate treatment as the source records. [3]

The debate should also be precise about frontier providers. OpenAI and Anthropic state that they do not train on business or commercial customer data by default, subject to

their applicable terms and opt-in arrangements. Training policy, however, answers a different question from where information is processed, how long it is retained, or who can access it. Each matters to the customer's decision. [4, 5]

Auditability makes these decisions inspectable. For a consequential action, the record should identify the information supplied or an access-controlled reference to its relevant version, the model and application configuration used, the tools invoked, the relevant authorization, and the result. That record should support investigation while observing its own access and retention rules.

An execution record cannot establish that a recommendation was correct. Evaluations, checks against the task's requirements, and appropriate human review must address that question. Together, these capabilities let an organization determine what happened, whether it was acceptable, and what needs to change.

Applications Must Remain Adaptable

The procedures behind an AI application will change, and the application has to change with them.

AI applications must remain adaptable because the organization's procedures give them their purpose. Those procedures will change as teams learn, responsibilities move, and new requirements emerge. A sovereign system should let authorized people translate those changes into working applications within established controls.

An agentic system of work can also change which steps and handoffs a procedure needs. If agents reliably assemble vehicle history, check parts availability, and prepare a work order, a maintenance team may combine administrative steps that previously passed between several people. The supervisor can review a prepared work package, with unresolved issues identified for attention.

The organization should be able to approve that redesign, assign responsibility for exceptions, and verify that required inspections and approvals still occur. Whether the change comes from a revised maintenance procedure or a new division of work between people and agents, the customer needs to test the resulting application before it enters operation. Customer staff, a chosen partner, or the supplier may implement the change; the organization should decide how that responsibility is assigned.

Cost awareness and efficiency belong in that redesign. The customer should be able to see how proposed changes affect model calls, token consumption, and spending. The team should be able to remove redundant calls, select models suited to each task, and set token and cost budgets for the workflow. Those budgets should govern retries and escalation to more capable models, with authorization required to exceed them. Proposed changes should be evaluated against the quality of the resulting work order,

total resource cost, and human review and rework. When the approved budget is insufficient to complete the task, the application should seek authorization or return the unfinished work to a person.

That requires mature software disciplines: versioned configurations, controlled releases, evaluations, and a way to recover when a change fails. The customer should be able to compare proposed behavior with the currently approved application before putting it into use. Permissions, monitoring, and approval controls should remain consistent as the workflow evolves.

Composability makes this more practical. A connector, review step, tool, or evaluation can be a reusable component with a defined purpose and interface. Teams can assemble and revise applications from those components while preserving the permissions and operational rules each requires.

Reuse still requires judgment. A component approved for one process may need different data access, failure handling, or human supervision in another. The benefit is that teams can examine and adapt an existing capability, with its assumptions visible, instead of reconstructing it without a reliable starting point.

Operational knowledge deserves the same treatment. Approved instructions, process rules, corrected examples, evaluation cases, and application configurations can capture what the organization has learned about doing the work well. Customers should have clear rights and practical means to retain and reuse the artifacts they create, with access governed by their sensitivity.

A deployment should leave behind an application that can evolve and capabilities that authorized teams can use again.

Model Consumption Must Serve the Outcome

Every model call costs something, and the right model depends on the step.

Control over model consumption means deciding how much capability a task needs and which resources it is worth consuming. Different steps in an application may justify different models, or no model at all.

In the maintenance workflow, a fixed rule can check whether a required inspection record is present. A smaller model may be sufficient to extract fault descriptions from technician notes. A more capable model may be justified when the task requires comparing a long repair history with relevant technical guidance. Each choice needs evaluation against the work, its sensitivity, and its acceptance criteria.

Model orchestration should enforce the customer's choices across the workflow, including model selection, consumption limits, retries, and escalation. A fallback must remain inside the information and action boundaries established for the original task. Changes to routing should be tested before release so the customer can assess their effect on both quality and resource use.

Usage should be attributable to the application and task that incurred it, with token counts and spending visible against the relevant budget. A customer needs to distinguish productive use from repeated failed attempts, unnecessary context, or an agent that continues calling tools without completing its assignment. The application should make a budget overrun or incomplete task visible and follow the customer's approved response.

This is how reliance on frontier services can be reduced where alternatives meet the requirements. The customer retains access to stronger models where their additional capability earns its cost. A sovereign approach should preserve both options and make the reasons for using them visible.

Token consumption is useful to measure, but the economic unit that matters is completed work that meets the organization's requirements. A cheaper model may require more attempts or human correction. A more expensive model may be worthwhile if it improves the accepted result or reduces the effort needed to obtain it.

Evaluation should therefore consider output quality, completion rates, review and rework, latency, and the full resources consumed. Local models also incur infrastructure and operating costs. Model substitution should be tested against the same task criteria, with any additional adaptation made explicit.

The standard is the customer's ability to make and enforce an informed decision about capability and consumption, then revise that decision as evidence changes.

Data and Deployment Should Follow the Mission

Where data sits and where work runs should each have an operational reason.

Data placement should follow the mission, so the customer should decide which information remains in existing systems and which needs to be copied, indexed, or transformed for a particular workflow. A sovereign architecture should support working with approved sources where they are, subject to the work's access, performance, and continuity requirements.

This keeps a migration decision tied to an operational reason. Consolidation may serve a particular need. Where existing sources can support the work, adoption should be

able to build on them. The customer should understand the effects of each choice on freshness, retention, and exposure.

Keeping a source record in place does not mean that no information moves. Retrieval, processing, caching, and logging can all create additional copies or representations. A credible account of sovereignty should show those flows and the controls applied to them. It should also preserve the distinction between an authoritative source and a derived result.

Deployment choices should follow the same discipline. Some workloads can meet their requirements in a managed cloud. Others need processing close to sensitive records, physical operations, or users with limited connectivity. The organization should be able to select an operating arrangement suited to the work and understand what would be required to change it.

At the edge, that means evaluating the complete workflow. A local model may still depend on remote data, identity services, tools, or approvals. If the mission requires disconnected operation, the necessary models, permitted data, authentication, application state, and supporting services must be available within that environment.

At a remote maintenance site, the team may still prepare a work order using approved local manuals and vehicle records, while a parts reservation may need to wait for access to the inventory system. The expected behavior during disruption should be explicit. Users need to know when data is stale, which actions remain authorized, and how queued changes will be reconciled when connectivity returns.

Continuity should be demonstrated under the conditions the workload is expected to face.

The Standard Must Apply to Every Supplier

The software that promises control is itself a dependency.

The standard has to apply to every supplier, because an organization can gain model choice while becoming more dependent on the application platform that coordinates those models. Sovereignty therefore requires examining each material dependency, including the software that promises to provide control.

The customer should understand which decisions it can make directly, which require supplier assistance, and which are constrained by licensing or architecture. Documentation, administrative access, supported interfaces, and usable exports should make those boundaries concrete.

Being able to export data is only one part of that assessment. The organization also needs to know what happens to its application configurations, workflow logic, evaluation cases, and operating history. Can it reuse the relevant artifacts? What software is required to interpret them? Which functions would need to be rebuilt if the relationship changed?

Commercial platforms can create substantial value through integrated engineering and support. The customer can knowingly accept those dependencies when the benefits justify them. A credible sovereignty claim should make transition costs and limits visible, including what continued operation would require under the applicable licenses.

The customer's ability to retain its operational knowledge can coexist with the provider's ownership of its underlying software. Contracts, documentation, and export formats should make the distinction usable. A transition plan should identify the work and assistance needed to move the relevant capability.

This standard should apply to every provider that describes its AI as sovereign, including Legion Intelligence. The customer should be able to judge the supplier by the capability it delivers and the authority the customer retains.

Practical Control Should Be Demonstrated

Any supplier can describe its controls. A demonstration on a real workflow shows whether they hold.

Practical control is best demonstrated by exercising it on a workflow the organization recognizes. The demonstration should include ordinary operation, an authorized change, and a condition that puts a claimed control under pressure.

In the maintenance workflow, that could mean adding the new inspection requirement, combining administrative handoffs into one reviewed work package, evaluating an alternative model, or interrupting the inventory connection. The exercise should show whether the customer can change both the process and the application, enforce its resource budget, and preserve the required controls, with any supplier assistance recorded.

The following expectations turn the standard into observable evidence.

What the organization should control	What a demonstration should establish
Information and actions	Attempts to retrieve restricted information, disclose it to an unapproved service, or act without authorization are prevented and recorded.

What the organization should control	What a demonstration should establish
Investigation and correction	Authorized reviewers can inspect the execution record and assess the result against the task's requirements.
Application change and reuse	An authorized team can revise the application and its underlying process, reuse a relevant component, and verify that required controls still hold, with effort and supplier assistance recorded.
Models and consumption	A model or routing change is evaluated for quality, cost, latency, and review effort, and token and spending limits are enforced.
Deployment and continuity	Required functions behave as specified during an interruption, with dependencies and recovery behavior visible.
Customer artifacts and supplier dependence	Relevant artifacts can be retained or exported, and their practical reuse and operating requirements are demonstrated.

A well-engineered platform should make this evidence available through normal operation. Its identity controls, deployment process, evaluations, monitoring, and recovery mechanisms should help the organization sustain control as applications change. The customer should know who is responsible for maintaining those capabilities and what skills or support are required.

Organizations should be able to adopt better models, redesign procedures as agents take on additional work, and manage the resources those applications consume while preserving authority over the work. Their architecture, commercial terms, and delivery arrangements should make that ability demonstrable throughout the application's life.

Sovereign AI should leave an organization more capable of running and improving its operations.

Sources

- IBM. What is AI sovereignty. Updated September 11 2026.
- Microsoft. Sovereign AI: Accelerate innovation with control and choice. October 5 2026.
- Oracle. Sovereign AI: Building AI Where Data, Infrastructure, and Control Stay Aligned. July 16 2026.
- OpenAI. Enterprise privacy at OpenAI. Updated January 8 2026.

Continue the series

Follow Legion Intelligence on LinkedIn: linkedin.com/company/legion-intel

Explore all Command Papers: legionintel.com/command-papers

Ready to talk: legionintel.com/contact