Intelligence Wieldability: Can an AI-Operating System Observe Itself?

Abstract

As we hand more operational authority to AI agents, a question that has received little attention becomes load-bearing: **can the system the agent operates answer questions about itself?** We argue that the practical ceiling on autonomous operation is not the intelligence of the agent but the *wieldability* of the system it acts on — the degree to which the system exposes its own state, mappings, freshness, and processes as first-class, queryable surfaces. When it does not, any intelligence operating it — human, an in-product assistant, or an autonomous agent — is forced to reach *around* the system into raw substrate (container orchestrators, SQL consoles, log greps). We formalise each such reach as a **wieldability defect** with one of four types (DATA, MAPPING, UPDATE-MECHANISM, PROCESS incomplete), and present a field study from ARQERA — a production AI-operations platform (2,688 API endpoints, 805 services, ~653K lines of backend code, a ~40-million-row evidence substrate) — in which a single autonomous engineering session surfaced nine distinct wieldability defects. We show that one defect class (request-path scans of the substrate defeating their own indexes) produced a family of user-visible failures presenting identically as "offline," and that the fixes yielded 19×–300× latency reductions once the system was made observable to itself. We name a deeper, recurring pattern — **the platform declares the user's valid credential dead because of an internal identity/routing mismatch** — and show two independent instances (LLM routing and integration health) that reduce to it. We close with a testable claim: *the marginal value of a more capable agent is bounded above by the wieldability of the system it wields*, and propose wieldability as a measurable, first-class property of AI-operable systems.

1. Introduction

The dominant narrative in applied AI is capability: better models, longer context, more reliable tool use. In production, we found the binding constraint elsewhere. Repeatedly, an agent (or a human operator, or the product's own AI assistant) failed a task not because it lacked the intelligence to reason about it, but because **the system it was operating could not tell it what it needed to know about itself.** The agent knew *how* to fix a broken deploy; it could not get the system to say *which version was running*. It knew how to diagnose a 500; the system could not surface *why*. It knew the concept of a bounced email; the system held no queryable record of *what created the offending account*.

Each of these gaps forced a *workaround* — reaching past the product into the raw substrate: `kubectl` to read a container image tag, `psql` to read an account, `grep` across source to find a code path. Every workaround is diagnostic of a missing surface. We call the property whose absence forces the workaround **wieldability**, and this paper is a field study of its failure modes in a real, large, autonomously-operated system.

Our contributions:

1. **A definition and a defect taxonomy** for wieldability (§2).
2. **A field study** (§3) of nine wieldability defects surfaced in a single autonomous session, each grounded in the workaround it forced.
3. **A quantified root-cause study** (§4) of the platform's most common user-visible failure — everything presenting as "offline" — showing it to be a wieldability defect (the system scanning its own substrate on the request path, blind to its own indexes) and reporting the 19×–300× improvements from making the system observable to itself.
4. **A named cross-cutting pattern** (§5): *the platform declares the user's valid credential dead due to an internal identity/routing mismatch*, with two independent instances.
5. **A testable thesis** (§6): agent value is bounded by system wieldability.

The setting is ARQERA, a governance-first, bring-your-own-key (BYOK) AI-operations platform. Concretely: **2,688 API endpoints across 402 routers, 805 service modules, 170 data models, ~653,700 lines of backend Python and 1,440 frontend modules**, backed by an append-only, hash-chained **evidence substrate of ~40 million rows** (the largest single table, the evidence ledger, is **11.2 M rows / 17 GB**; sibling address ledgers add another ~21 M rows / ~19 GB). The substrate records **252 distinct artifact types** over **174 days** at roughly **64,000 artifacts/day**. The platform is operated by autonomous engineering sessions; the field data below comes from one such session on 23 July 2026.

---

2. Wieldability, defined

> **Wieldability.** A system is *wieldable* by an intelligence to the degree that the intelligence can obtain every fact it needs to act — the system's state, the relationships between its parts, the freshness of those facts, and the ability to perform its operations — *through the system's own first-class surfaces*, without reaching into the substrate the system is built on.

The test is operational and adversarial: **every time an intelligence operating the system must reach *around* it, that reach is a defect.** The reach is observable (a `kubectl`, a `psql`, a `grep`), so wieldability is measurable by counting reaches per unit of work.

We classify each defect by what is missing:

Type	The system…	Example workaround it forces
DATA	does not hold the fact at all	parse container images to learn the deployed version
MAPPING	holds the fact but cannot relate it to the question	cannot answer "what created this account" though both exist
UPDATE-MECHANISM	holds the fact but lets it go stale	a self-report field permanently reads `"unknown"`
PROCESS	offers no native path to do or observe an operation	diagnose a payment failure only via raw provider logs

This taxonomy is not merely descriptive. It is *prescriptive*: a DATA defect is closed by capturing the fact; a MAPPING defect by adding a relation/query; an UPDATE defect by wiring the refresh; a PROCESS defect by building the surface. Each defect names its own fix.

Relationship to observability. Classical observability (metrics, logs, traces) targets *human operators debugging failures*. Wieldability targets *any intelligence operating the system to accomplish work*, including the system's own agents. The distinction matters: a system can be richly observable to a human with a Grafana dashboard and yet be un-wieldable by its own agent, which has no dashboard and needs the fact as a typed API response. In our study the platform could not answer, through any product surface, "what version of yourself is running?" — a question its own deploy agent must answer on every task.

---

3. Field study: nine defects in one session

The following nine defects were surfaced in a single autonomous engineering session. For each we record the fact the intelligence needed, the workaround the missing surface forced, and the defect type.

#	Fact needed	Workaround forced	Type
1	What backend version is live?	`kubectl` parse of container images (past a sidecar at index 0)	DATA + UPDATE
2	Why is checkout returning 500?	`kubectl logs` + in-pod Python to reproduce	PROCESS
3	Is the payment-plan catalog healthy?	run reconcile scripts by hand; query the payment provider's API	DATA
4	Why did that email bounce?	`grep` source + `psql` + read logs	DATA
5	What created account `d4-uat-…@…`?	grepped the codebase; **could not find it**	MAPPING
6	Does enforcement match what the UI displays?	wrote comparison code against two config sources	PROCESS
7	What is slow, and why?	`pg_stat_activity` by hand (`pg_stat_statements` was not even installed)	DATA
8	Why is the product "offline"?	hand-timed candidate SQL in `psql`	PROCESS
9	What is the status of background job X?	**could not locate it** from any product surface	MAPPING

The pattern is uniform: **the platform runs the world but cannot observe itself.** It governs money, dispatches agents, and emits 64,000 evidence records a day — yet cannot report its own deployed version, its own slow queries, or the provenance of an account it created. Notably, ARQERA already implements a strong version of wieldability *for its customers* — its explicit doctrine is that a customer's authority is carried by their agents, and the platform governs its own infrastructure through that same lens. The nine defects are that doctrine, turned inward, and found missing.

Closing the defects. Each fix is the prescription its type implies:

- Defect 1 (DATA + UPDATE): the deploy now stamps the build tag and git SHA into the running process's environment at the same instant it sets the image; a public `/health` endpoint self-reports `{image_tag, build_sha, build_time}` — turning `curl …/health | jq .build.image_tag` into the answer that previously required container archaeology.
- Defect 7 (DATA): `pg_stat_statements` — already preloaded in the database but never activated — was enabled with a single `CREATE EXTENSION`, immediately surfacing a 238-second-mean query the static analysis had missed. *The capability existed; the surface did not.*
- Defects 2, 6, 8 (PROCESS): a self-observability layer (health, version, schema-drift and catalog gates in the deploy pipeline) replaced the manual reproduction loops.

The empirical lesson: **most wieldability defects are not missing capability but missing surface.** The database could always compute its slow queries; nothing exposed them. The deploy always knew its tag; nothing carried it into the process. Wieldability is cheap to add and expensive to lack.

---

4. Case study: when everything is "offline"

The platform's most frequent user-visible failure was uniform and opaque: pages returned a Cloudflare 524 ("origin did not respond") which the frontend rendered as *"You're offline."* The symptom hid its cause; "offline" is what a wieldability-poor system says when it cannot explain itself.

4.1 Root cause

Grounding in live data (not static reasoning) located it precisely. The evidence ledger is **11.2 M rows / 17 GB**. The chain-verification endpoint fetched "the most recent N" records via `OFFSET (total − N)` — which forces PostgreSQL to **scan and discard ~11 million rows** to reach the offset. Measured on the platform tenant:

Query pattern	Latency	Note
`OFFSET (total − N) LIMIT N` (original)	**16.3 s** (first rows)	scan-and-discard of the whole table
`ORDER BY created_at DESC LIMIT N` (index-backed)	**0.10 s**	bounded index range scan

A **163× difference** from one clause. Worse, the same endpoint opened with a `COUNT(id)` over the tenant's slice, documented in-code as a "cheap indexed query." On the 4.2-million-row platform tenant it measured **44.75 seconds** — comfortably past Cloudflare's 100-second origin ceiling, hence the 524, hence "offline."

4.2 The wieldability diagnosis

The failure was not load; it was a system unable to read its own substrate efficiently. The fix was to make the system observe itself through the right surfaces:

- **Totals** from a maintained aggregate projection (`evidence_rollup_daily`, 14,484 day-rows) rather than the raw ledger: `COUNT` **44.75 s → 7 ms**.
- **Emptiness** via an index `EXISTS (… LIMIT 1)` rather than `COUNT`: **44.75 s → 1.5 s**.

- **Recent-N** via an index-backed `created_at` cutoff rather than `OFFSET`: **16.3 s → 0.10 s**.

End-to-end, chain verification on the platform tenant went **45 s → 2.3 s (≈19×)**.

4.3 The systemic multiplier: RLS blindness

A sweep found **17** request-path handlers scanning the giant ledger. The worst were *public, unauthenticated* endpoints (a trust page, a public-stats endpoint) running a 44-second `GROUP BY` over 11 million rows **on every anonymous hit** — a permanent 524 and a trivial denial-of-service surface.

A second-order wieldability trap appeared here and is worth isolating. Row-level security (RLS) makes a table's rows invisible unless a tenant context is set. On a *public* endpoint no tenant is set, so PostgreSQL applies the RLS predicate as a *per-row filter* — which **defeats the very index the query would otherwise use**. An `EXPLAIN` confirmed an `Index Scan Backward` whose gains were nullified by a per-row RLS `Filter`, degrading `MIN(created_at)` from O(log n) to a full 11-million-row scan. Public and system reads therefore *must* go through fail-open projections; they can never touch the RLS table directly.

This same RLS blindness bit the *diagnosis itself*. Mid-investigation, a system query reported the membership table "empty — 0 rows across 147 tenants," which would have been a catastrophic Tier-1 finding. It was false: the query ran without a tenant context and was RLS-masked. The true count, via an explicit system bypass, was **156 memberships**. **The instrument was blind for the same reason the system was.** We adopted a discipline — *verify the measurement harness before reporting the measurement* — and it prevented a false bug report. This is wieldability applied reflexively: an intelligence must be able to observe *its own observations*.

4.4 Results

Post-fix, the three worst public endpoints moved from a **60 s+ timeout to 0.1–0.2 s**; twelve more request-path scans were routed to the projection or to `pg_class.reltuples` estimates. "Offline" — the platform's most common failure — was, in the end, a wieldability defect: a system that could not read itself without scanning itself to death.

---

5. A named pattern: the platform disowns the user's valid credential

The most illuminating finding is a single pattern with two independent instances, discovered by operating the platform as its real owner and tracing failures to ground truth:

> **The platform declares the user's valid credential dead because of an internal identity or routing mismatch — not because anything the user holds is actually broken.**

Instance A — LLM intelligence. The product's AI assistant returned, to the user, a bare failure. Tracing revealed the user's *own, working* provider key (Google/Gemini, a valid BYOK credential) was never used. The router sent every request through the platform's own managed AI gateway, whose auth token had expired — so **every** provider routed through it returned `401`. The user's key, which would have hit the provider directly and worked, was never tried. The platform's routing layer had orphaned the user's valid credential. *(Fix: BYOK keys route directly to the provider; only platform-owned keys use the gateway. Verified live — the assistant streamed real tokens where it previously failed.)*

Instance B — integration connections. A freshly-linked integration (linked "one minute ago") showed status `error: expired or revoked`. Ground truth from the integration provider: the connection was **valid**, holding live credentials — but stored under a UUID the provider assigns, while the platform's health probe checked a *deterministic phantom id* (`{tenant}-{user}`) that the provider had never heard of (`404`). The probe read `404`, called the connection dead, and told the user to reconnect a connection that already worked. *(Fix: resolve the phantom id to the provider's real id before declaring failure; deduplicate the identity records. Verified live — the connection flipped from `error` to `healthy`.)*

Two subsystems (LLM routing, integration health), two mechanisms (an expired gateway token, an identity-scheme mismatch), one disease: **a platform-internal identity/routing layer breaks the user's working credential and then blames the user.** This is a *wieldability* failure at the boundary between the platform and the user's assets — the platform cannot correctly relate its internal identity for a credential to the credential's true identity (a MAPPING defect), and the user is told to fix what is not broken.

The generalisable lesson for autonomous, multi-tenant AI platforms: **every layer that proxies or re-identifies a user's credential is a place the platform can silently disown it.** The BYOK doctrine — *the user's own key, direct to the provider, never platform-funded and never platform-proxied* — is not only an economic and trust choice; it removes an entire class of these mappings, and with them an entire class of failures.

---

6. Discussion: the wieldability bound

We state the thesis the field data supports:

> **The marginal value of a more capable agent operating a system is bounded above by the wieldability of that system.**

The intuition is direct. An agent's competence converts to outcomes only through the facts and operations the system exposes. If the system cannot report its own version, a perfectly capable deploy agent still cannot verify a deploy without reaching into the orchestrator. If the system scans itself to death, a perfectly capable diagnostician still meets a timeout. Capability past the wieldability ceiling is stranded: the agent knows what to do and cannot get the system to let it.

This reframes a good deal of "agent reliability" work. Much of the observed unreliability of agents-operating-real-systems is not the agent hallucinating or mis-reasoning; it is the agent *correctly* concluding it cannot proceed because the surface it needs does not exist — and then either working around it (introducing substrate-level risk) or failing. Improving the agent does not move this ceiling; improving the system's wieldability does.

Three testable predictions follow:

1. **Reaches-per-task should fall as wieldability rises** and correlate with task success — a measurable, model-independent metric.
2. **The highest-leverage investments for autonomous operation are surfaces, not scale** — because §3 showed most defects were missing surfaces over existing capability (a database that could always compute its slow queries; a deploy that always knew its tag).
3. **Wieldability defects cluster at re-identification boundaries** (§5) — anywhere the platform proxies, caches, or re-keys a user asset.

Limitations

This is a single-system field study, not a controlled experiment; the wieldability bound (§6) is argued and illustrated, not proven. The nine defects are those a single session surfaced, not an exhaustive census; the true defect count is a lower bound. "Reaches per task" is proposed as a metric but not yet instrumented longitudinally. The performance figures (§4) are measured on production data but on a single tenant distribution; they demonstrate the *mechanism* (scan-and-discard, RLS-defeated indexes) rather than a population average. And the platform is operated by AI sessions including the author's own — the observer is inside the system, which is a source of both unusual access and potential bias. We have tried to counter the latter by grounding every claim in a live measurement and by treating the measurement harness itself as suspect (§4.3).

Related directions

Wieldability sits between three literatures it does not reduce to: **observability** (aimed at human debuggers, not operating intelligences), **self-adaptive / autonomic systems** (which assume the monitoring surfaces wieldability treats as the missing thing), and **agent tool-use / environment design** (which studies the agent's side of the interface; wieldability studies the system's). We think it is the systems-side complement to agent research: as we ask agents to operate real infrastructure, the property that most limits them is how well that infrastructure can be *asked*.

---

7. Conclusion

We operated a large, real, autonomously-run AI platform and found its ceiling was not intelligence but *wieldability* — the system's ability to be asked about itself. Nine defects in one session, a family of "offline" failures rooted in a system that scanned itself to death (fixed for 19×–300× gains by making it observe itself), and a named pattern in which the platform disowns the user's valid credential — all reduce to the same thing: **surfaces the operating intelligence needed and could not get.** As we grant agents more authority over real systems, we should measure and build for wieldability directly. The agent is only ever as good as the questions the system can answer about itself.

Appendix A — Headline figures

Figure	Value
API endpoints / routers / services / models	2,688 / 402 / 805 / 170
Backend Python / frontend modules	~653,700 LOC / 1,440 files
Evidence substrate (largest table)	11.2 M rows / 17 GB
Total top-table substrate	~40 M rows / ~40 GB
Distinct evidence artifact types / span / rate	252 / 174 days / ~64,000 per day
Commits / fix-to-feat ratio	4,742 / **0.88 : 1** (down from a historical 2.1 : 1)
Wieldability defects (one session)	9
Request-path substrate scans found	17 (worst: public + unauthenticated)
verify_chain (platform tenant)	45 s → **2.3 s** (≈19×)
`OFFSET` vs index cutoff	16.3 s → **0.10 s** (163×)
Public endpoints	60 s+ → **0.1–0.2 s** (≈300×)
`COUNT` via projection	44.75 s → **7 ms**
Membership table (RLS-masked vs true)	"0" → **156**

All figures measured live on the production platform, 23 July 2026. Latency figures are single-run measurements demonstrating the mechanism, not population averages.