# The instrument problem: measuring an agent you intend to let act

*Faiz Shuja · Founder & CEO, SIRP · August 2026*

*Technical companion to "Accuracy is not calibration — what changes when the SOC agent acts". That piece argues the position; this one shows the arithmetic, the failure modes, and the measurement protocol. It assumes you are building one of these systems, not buying one.*

---

## 0. The claim, stated precisely

An agent that recommends is graded on **accuracy**. An agent that acts is graded on **the risk it retains at the coverage it chooses to operate at**. Those are different quantities, they are optimized by different means, and a system can improve the first while getting worse at the second.

Everything below is an attempt to make that statement operational.

For context, the two Elastic Security Labs results the companion credits, with their published evidence base: triage accuracy moved 60% → 92% by restructuring the context the agent reasons over rather than changing models, measured against analysts' closure reasons. Elastic also analyzed 36,822 conversations across production and QA (about 8 billion tokens). Specialized agents used 3.0–5.7× fewer tokens on the two measured investigation routes. In a separate four-alert macOS test, 57–60% of the single agent's model calls were deliberation without a tool call. **Elastic found that the number of model calls, not prompt size, dominates token cost.** Those are investigation results. What follows is about the step after investigation.

Notation, used throughout: a system produces on each case a decision $\hat{y}$ and a confidence $c \in [0, 1]$. Over $N$ cases, accuracy is $\text{acc} = (1/N)\ \Sigma\ \mathbb{1}[\hat{y} = y]$. An autonomy policy is a threshold $\tau$: act when $c \geq \tau$, defer otherwise.

---

## 1. Calibration is necessary and nowhere near sufficient

The companion piece contrasts two 90%-accurate systems. Let me make them concrete and then show that the standard calibration metric cannot tell them apart.

**System A** — reports $c = 0.90$ on every case. Right on 900.

**System B** — reports $c = 0.96$ on 750 cases (right on 720) and $c = 0.72$ on 250 (right on 180). Right on 900.

Expected Calibration Error, the field's default instrument, bins by confidence and measures the gap between confidence and accuracy in each bin:

$$\text{ECE} = \Sigma_k\ (n_k/N) \cdot |\ \text{acc}(B_k) - \text{conf}(B_k)\ |$$

| | ECE | Brier | reliability | resolution | uncertainty | errors |
|---|---|---|---|---|---|---|
| System A | **0.0000** | 0.0900 | 0.0000 | **0.0000** | 0.0900 | 100 |
| System B | **0.0000** | 0.0792 | 0.0000 | **0.0108** | 0.0900 | 100 |

**Both systems are perfectly calibrated.** ECE is exactly zero for each. System A says 90% and is right 90% of the time; it is not lying, it is not overconfident, and every calibration fix in the literature — temperature scaling, isotonic regression, Platt scaling — would find nothing to correct.

The property that separates them is **resolution**, and it is visible only under Murphy's decomposition of the Brier score:

$$\text{Brier} = \underbrace{\mathbb{E}[(c - \text{acc})^2]}_{\text{reliability} \downarrow} - \underbrace{\mathbb{E}[(\text{acc}_k - \bar{p})^2]}_{\text{resolution} \uparrow} + \underbrace{\bar{p}(1-\bar{p})}_{\text{uncertainty (fixed)}}$$

Reliability is what ECE measures, and both systems score 0. Uncertainty is a property of the problem, not the system — $0.9 \times 0.1 = 0.09$ for both, and no amount of engineering moves it. **Resolution is the entire difference: 0 versus 0.0108.** It is the variance of the system's own confidence bands around the base rate — a direct measure of whether the system can separate its hard cases from its easy ones.

This is the single most important correction I would make to how most teams instrument these systems. *Calibration tells you whether to believe a confidence number. Resolution tells you whether the number was worth producing.* A system with zero resolution has confidence that is honest and useless.

### Why this matters at the threshold

Autonomy is a thresholding decision, and thresholding a zero-resolution system does nothing:

| policy | System A | System B |
|---|---|---|
| $\tau \le 0.90$ | acts on 1000, **100 wrong** | acts on 1000, 100 wrong |
| $\tau = 0.95$ | acts on **0** | acts on 750, **30 wrong**, defers 250 |

System A has exactly two operating points — everything, or nothing. There is no setting at which it acts on its good cases and defers its bad ones, because it does not know which are which. System B, at identical accuracy, gives you a third of the errors and hands 250 cases to a human *because it knew it was on thin ice*.

---

## 2. The right instrument: risk–coverage, not accuracy

Selective prediction gives the correct frame. For a threshold $\tau$:

$$\text{coverage}(\tau) = |\{i : c_i \ge \tau\}| \,/\, N \qquad \text{risk}(\tau) = \Sigma_{i:\, c_i \ge \tau}\, \mathbb{1}[\hat{y}_i \ne y_i] \,/\, |\{i : c_i \ge \tau\}|$$

Sweeping $\tau$ traces a **risk–coverage curve**, and the area under it (AURC) is a single number that ranks systems by the thing you actually care about. At coverage 0.75, System A retains risk 0.10 — it cannot reach that coverage except by acting on everything — while System B retains **0.04**.

Report this instead of accuracy:

- **risk at fixed coverage** — "at 70% automation, what fraction of actions are wrong?"
- **coverage at fixed risk** — "at a 2% error budget, how much can it automate?" This is the number a security buyer is actually purchasing.
- **AURC**, for tracking a system against itself release over release.

A model change that raises accuracy by two points while flattening the risk–coverage curve has made the system *less* automatable. Accuracy alone will report that as progress.

---

## 3. Asymmetric loss: one threshold is the wrong shape

Section 2 assumes all errors cost the same. In a SOC they emphatically do not, and a single global τ silently prices them as if they did.

Let L(a) be the cost of a wrong action of class a, and B(a) the benefit of a right one. Acting is justified when

$$c \cdot B(a) > (1 - c) \cdot L(a) \iff c > L(a) / (L(a) + B(a)) = \tau^*(a)$$

The threshold is a **property of the action class, not of the model**. Enriching an indicator and isolating a domain controller do not share a break-even point, and they should not share a slider. Concretely: an action whose wrong-case cost is 20× its right-case benefit needs $c > 0.952$; one where the ratio is 2× needs only $c > 0.667$.

Two consequences that follow directly and are easy to get wrong:

1. **Reversibility belongs in L.** An action with a tested undo path has a fraction of the loss of an identical action without one. This is why "can it be undone" is a precondition for automation rather than a nice-to-have — it moves τ*, and it is the only term in the equation that engineering fully controls.
2. **Severity scales L, so it must move the threshold.** The highest-severity band should be pinned to human approval regardless of measured confidence, because at that L the required c exceeds anything the system can honestly claim.

In OmniSense this is the shape of the policy tiers. **Conservative**, **Balanced** and **Autonomous-First** set a per-action-class ceiling on τ*; they do not set one number for the system. Beneath the tier sit rails that no configuration overrides — destructive actions require approval, the maximum automatic scope is artifact-local, and the audit record is immutable. The tier loosens advisory behavior; it never reaches the rails.

One distinction I want to keep sharp, because it is exactly the kind §9 tells a buyer to probe: the **rails** are enforced in code on every execution path today. The **tier ceiling** is enforced on the primary decision plane, and we are still extending its reach to every plane that can execute an action. Those are different maturity levels, and it would be dishonest to let one sentence carry both. Until the ceiling reaches a plane, that plane runs at the default — which is one more reason the rails, not the tier, are what the safety case rests on.

---

## 4. Where the confidence number comes from

Everything above assumes a c worth thresholding. Raw self-reported model confidence is not that, for reasons worth stating plainly:

- **Verbalized confidence is a token distribution, not a frequency estimate.** "I am 95% confident" is generated by the same process that generates the rest of the sentence and inherits its training-distribution priors. It clusters hard on round numbers and rarely goes below 70.
- **Sequence likelihood is not task confidence.** Low perplexity on a fluent wrong answer is common.
- **Confidence conditioned on missing evidence is confidently wrong.** This is the one that bites in practice: an agent that never received the enrichment output will report high confidence about an entity it has no evidence for, because nothing in the prompt signals absence. The fix is architectural, not statistical — mark unverified entities as unverified *in the context*, so the absence is a token the model can condition on.

What we found workable is deriving confidence from **evidence structure rather than model introspection**: how many independent evidence sources agree, whether an adversarial pass tried to refute the finding and failed, whether the case matches a pattern with prior confirmed outcomes. This is slower and less elegant than asking the model, and it produces a number with resolution.

A corollary worth internalizing: **confidence is a claim the system makes, not a measurement it reports.** Grade it like any other output.

---

## 5. Measuring quality without fooling yourself

In our own measurement, the absolute-scale reviewer flagged far more responses than it should have. Here is the fuller methodology.

**Absolute scoring fails, and it fails quietly.** A judge asked to rate 1–10 produces numbers that look like data and behave like mood. A detector that flags most cases loses discriminative power, and you cannot distinguish an improvement from a regression when both sides are flagged. The numbers were never wrong exactly. They were uninformative, which is harder to notice.

**Pairwise comparison works.** Regenerate the same request through a second path and have a judge pick a winner. The details are not optional:

- **Randomize position.** Judges have a strong, well-documented preference for one slot. If you do not randomize, position bias silently becomes your result.
- **Allow ties**, and record **"both unacceptable"** as its own outcome. It is one of the most useful signals we collect and it is invisible to a scale that only ranks.
- **Power the experiment.** Detecting a 55% win rate against a 50% null at $\alpha = 0.05$ two-sided with 80% power needs **783 decided comparisons**. Teams routinely declare a winner on 40. At $n = 40$ you cannot resolve anything short of a rout, and most real improvements are not routs.
- **Judge the axis you care about.** A judge asked for "quality" optimizes fluency. Ask for the property under test — factual grounding, absence of unsupported claims, correct disengagement.

**Build the guard into the instrument.** A scorer must be able to say *"I could not read this"* separately from *"this is bad."* We ship an explicit unparseable-rate alongside every quality metric, and a harness-error gate that returns **alone** — suppressing the score entirely rather than reporting a number computed from garbage. It fired correctly on its first live run, and it exists because of §6.

---

## 6. Five ways the instrument lies

This is the section I would most want handed to me two years ago. Each of these is a real failure from our own pipeline, generalized. In every case the system was fine and the measurement was broken, and in every case the measurement was believed for weeks.

**1 — The unreachable denominator.** We graded how well an agent cited its evidence: fraction of available payload fields the answer referenced. The ratio sat near 0.01 against a 0.30 target. Four separate fixes shipped; the ratio did not move. The denominator counted **every** non-empty string field — several hundred — while the answer can only meaningfully cite the handful that carry behavioral signal. **A perfect answer would have scored 0.04.** The target was unreachable by construction, and every "failure" it reported was a property of the metric.

> *Before you fix a number, compute what a correct system would score on it. If a perfect answer cannot hit the target, you are debugging the ruler.*

**2 — The constant that reads as a value.** A diagnostic line ended `→ %d action step(s) (0 ⇒ catalog had no match)`. The parenthetical is a **legend** — a fixed string explaining what a zero would mean — and it prints identically whether the count is 6 or 600. It was read as the count. A bug was filed against the wrong subsystem, with the wrong root cause and a wrong lineage to two unrelated prior defects. The real count was 6.

> *Count the format placeholders, or render the template and diff it against the observed line. It takes one line and it invalidated an entire investigation.*

**3 — The null with no writer.** A field was empty on 65% of records. That count alone supports at least three incompatible stories: a broken write, a broken producer, or a population that legitimately has no value. The cross-tab of *(did the writer run)* × *(is the field present)* × *(which source)* picked exactly one: there were **two** suppliers with different coverage, and the primary writer had not executed on those rows at all.

> *A missing value is only evidence about a writer that actually ran. And a deliberate refusal — a guard, a hard block — writes the same null as a defect does. Read the row's status before counting it as a failure.*

**4 — The zero that was the instrument.** Stated generally, because it recurs: when a metric reads zero, the prior should be that the *measurement* broke, not that the system stopped. Check that the producer emitted, that the reader reads the key the producer wrote, and that the sample window covers the event. We have burned several days on "the feature is dark" that were "the counter is dark."

**5 — The fixture that masked the bug it tested.** A test helper constructed a payload with a nesting one level different from production. The tests passed for a month against a read path that could never work on real data. This is the most dangerous item on the list, because a green suite is the thing you rely on to know the other four have not happened.

> *Build fixtures from captured production payloads, not from your mental model of the schema. The two disagree far more often than anyone expects.*

The pattern beneath all five: **the measurement and the system fail in ways that look identical from the outside.** Any team automating consequential actions needs a habit of asking, before "why is the system doing this," the cheaper question — *"am I reading what I think I am reading?"*

---

## 7. Rollout as an evaluation protocol

Shadow mode is usually described as a safety measure. It is more valuable as an **instrument**, and the difference changes how you run it.

Every consequential capability ships through `off → shadow → enforce`. In shadow it computes its decision, records it, and changes nothing. That yields the only dataset that matters for the threshold question: **paired decisions on identical inputs**, one from the incumbent path and one from the candidate, with the outcome eventually observable on both.

Three disciplines make it real, each learned the hard way:

- **Declare the rung and verify it live.** A flag-gated capability that is never compared does not exist. Read the deployed configuration rather than the repository default — they diverge, and the divergence is invisible in code review.
- **Ship the reader with the writer.** A shadow decision written to a store nothing queries is not shadow mode, it is a silent write. We have shipped an instrument whose own attribution field was unreadable, then spent days puzzled that three fixes had not moved the ratio.
- **Diff at the reason level, not the flip level.** Counting how often candidate and incumbent disagree tells you almost nothing. Comparing *why* each decided is what surfaces the systematic difference, and it is what distinguishes a real regression from sampling noise.

Our adversarial verification layer — a pass that actively attempts to refute a finding before it stands — runs in shadow today. It observes and logs. It does not gate live decisions. A technical audience should be able to tell which parts of an architecture are load-bearing and which are still being measured.

---

## 8. Earned autonomy, formally

If §3 is right that τ* is per-action-class, and §1 is right that resolution is what makes thresholding meaningful, then a global autonomy switch is a category error. Autonomy is earned **per pattern**, with the tier as a ceiling rather than a grant:

- **Tier bounds; memory gates.** The policy tier sets the maximum autonomy for an action class. Whether a *specific* pattern reaches that ceiling depends on its confirmed-outcome history. In the same spirit as the shadow disclosure: the outcome feed that deepens this history is the newest part of the loop and is still being wired end-to-end. Until it flows at volume, patterns do not graduate — **a gate with no evidence stream holds at the floor and asks**, which is the correct direction to fail in. An earned-autonomy model that estimated its way past missing evidence would be the confident guess this document exists to prevent.
- **Negative evidence weighs more.** A disconfirmation counts roughly double a confirmation. The asymmetry is deliberate: the cost of a wrong action exceeds the benefit of a right one, so the evidence bar should be asymmetric too. It falls straight out of the L/(L + B) term.
- **Undo is a precondition, not a mitigation.** No action class should auto-fire before a tested reversal path exists. See §3.1.
- **Destructive actions require approval at every tier.**
- **Silence is never approval.** An expired approval window should escalate.

The property that makes this auditable is an immutable audit log of every decision. If you cannot reconstruct *why* the system acted six months later, you do not have an autonomous system — you have an unattributable one.

## 9. What to ask any vendor, including us

None of this is checkable from a datasheet, which is why we publish the SOC Autonomy Framework — an open, vendor-neutral classifier and benchmark suite. Concretely, the questions that separate a measured system from a described one:

1. **Show the risk–coverage curve**, not the accuracy number. What is the retained error rate at the coverage you are recommending?
2. **Report resolution, not just calibration.** A perfect ECE with zero resolution is a system that cannot be safely thresholded — and it will pass every calibration check you run.
3. *What is τ per action class, and how was L estimated?** If there is one slider for the whole system, §3 has not been done.
4. **Which components are enforcing and which are shadow?** Today, in production, not on the roadmap.
5. **What does it do when it disengages?** A clean stop with a decision-ready handoff is a feature. A confident guess is the failure mode this entire document exists to prevent.

If you run the framework against OmniSense and find our level lower than our marketing implies, that result is more useful to both of us than a number on a slide. The industry does not need more adjectives. It needs published methodology and instruments that fail loudly.

*SIRP builds OmniSense, a Self-Driving SOC. The SOC Autonomy Framework, its classifier and its benchmark suite are published openly at github.com/SIRP-Labs.*