torq

GROUND TRUTH: WHAT THE AI SOC STANDS ON

Context, Memory, and Judgment Beneath the Agentic SOC

Executive Summary

The security industry spent the last two years proving that large language models (LLMs) can read an alert and produce a credible analysis on day one. That was a real breakthrough. It was also the easy part.

The hard part is everything after. A model that reasons brilliantly but forgets every decision it makes, misses the context of your specific environment, and never learns your team's judgment will produce confident, fast, wrong answers at scale. The SOC's problem was never a shortage of reasoning. The SOC's hardest problem has always been a flood of noise with no shared memory.

A capable model is necessary, but it is no longer the differentiator. The differentiator in the AI SOC is the layer beneath the agents: **the context that grounds every decision, the memory that gives it precedent, and the learning that teaches it your team's judgment.** That layer is the part that's hard to copy and easy to get wrong, and it's where the advantage compounds. This paper breaks down each part, the engineering required to keep it correct, and how the three combine in the Torq AI SOC Platform to make autonomous action something a security team can actually trust.

For security leaders evaluating AI SOC platforms, this paper also offers a practical lens:

- What to demand from the grounding layer
- Why generic retrieval fails in security
- How to tell a platform that learns from one that simply re-runs the same logic faster

Reasoning Is Table Stakes. Grounding Is the Moat.

Even the smartest agent is only as good as what it stands on. Give it solid ground — memory of its own decisions, a model of your environment, your team's judgment — and its reasoning pays off. Leave it ungrounded, and that same reasoning has nothing real to act on.

The numbers describe the environment in which these agents operate. 73% of teams named false positives their single biggest detection challenge.[1] Even with AI cutting the breach lifecycle to a nine-year low of 241 days, that is still roughly eight months of exposure,[2] and most of that gap is the work that comes after detection. Practitioners, meanwhile, consistently rate generative AI tools as their least satisfying category of tooling.[3]

The reason is structural. Most AI SOC tools evaluate each alert in isolation, with no memory of their own decisions, no model of the specific environment they're defending, and no ability to absorb analysts' corrections. Every shift, the AI starts from scratch. The false positive that a team carefully explained on Monday returns looking brand new on Tuesday.

Solving this is an engineering problem, and a bigger model doesn't address it. Everyone now claims their agents are "grounded in context." Far fewer can say what that context costs to keep correct. The rest of this paper describes what good looks like across three layers (i.e., context, memory, and learning) and what it takes to build each one in production.

What Security Leaders Should Demand From the Grounding Layer

Start with the buyer's lens before the architecture. Strip away the vendor language, and good grounding is simple: a living model of your environment that an agent can reason on and that you can audit. Four properties separate context you can trust from context that quietly goes wrong:

1. **Always current.** Context decays. Ownership changes, services get decommissioned, and exceptions expire. A graph that was right at ingestion is wrong by the next alert. The test is whether the data are true at the time the decision is made, not whether they were fresh when you loaded them.
2. **Traceable.** For any recommendation, the system should show what it knew, when it knew it, and where it knew it. This is why 90% of leaders in Torq's 2026 AI SOC Leadership Report said explainable AI decisions matter most.
3. **Decision-capturing.** That a user connects to a device is inventory. Why your senior analyst closes an alert the SOP says to escalate, is judgment, and judgment is what walks out the door when that analyst leaves.
4. **Isolated.** This layer encodes your most sensitive truth: who is privileged, what is exempt, and where the gaps are. It has to stay yours.

The payoff for getting this right is measurable. IBM found that teams that used AI and automation extensively had average breach costs of $3.62 million, compared with $5.52 million[4] for teams that didn't. AI-enabled teams already hold that advantage; grounding is how you keep it as agents start acting autonomously.

Context: The Torq Context Graph

In the Torq AI SOC Platform, the grounding layer is the Context Graph: a live model of your environment every agent reasons on, not a static store each agent re-derives. Torq's acquisition of Jit brought in a context graph already running in production, moving the platform from one that enriches alerts to one that acts on the full story of a case.

The difference between enrichment and a graph is the difference between one lookup and a chain of them. Take the case where most platforms stop: the same alert fires on two laptops. One belongs to a contractor with read-only marketing access, the other to a finance director who can reach the M&A data room. Same signal, different verdicts. Resolving which is which is a single identity lookup — any tool with an identity feed can do it, and that's exactly why enrichment is table stakes. The Context Graph goes further: it doesn't just inventory what exists, it encodes what it means, chaining that role to the systems each person can reach, the data at stake, and how the case has unfolded so the agent reasons on the full story, not one field.

The questions that decide whether an agent can be trusted to act start one hop later, at the finance director:

- If this identity is compromised, what does it actually reach — the blast radius?
- Which policy governs the M&A data room?
- What else can reach that data, and does any of it sit outside the controls you assume protect it?

None of those is a property of the alert, the user, or the asset on its own. Each is a relationship between them, and the answer shifts as access, ownership, and policy change. Enrichment hands an agent a richer fact and stops. A graph lets the agent traverse the consequences of that fact, which is what acting on the full story of a case requires.

Four engineering choices decide whether a context graph is trustworthy or merely decorative. None of them is solved by a bigger model.

1. **Bitemporal time:** Every fact carries two clocks: when it was true in the world (valid time) and when the graph learned it (transaction time). That is what makes a verdict replayable: an agent reconstructs the graph as of the moment the alert fired and reasons against what was true then, rather than today's view projected backward onto a two-week-old event. Those same two clocks tell the agent how fresh each fact is, so a verdict can weigh what was just confirmed over what hasn't been seen in days, instead of treating every fact as equally current.
2. **Typed meaning:** The edges carry semantics. Not a vague "related to" but "approved by," "governs," "grants access to," "depends on" — the actual operational relationship between two nodes. Policies, access controls, and retention rules live in the graph as queryable nodes too. That lets an agent traverse a real question — Who can approve an exception for this asset? Which policy governs this data? What gets exposed if this identity is compromised? — as a graph query with known semantics, instead of inferring it from free text and hoping.
3. **Decisions as first-class nodes:** This is the part most platforms never build, because it's a data-model problem rather than a feature you can bolt on later. Every verdict, exception, override, and escalation becomes a node in the Context Graph, carrying who made the call, the context at the time, the SOP it followed or broke from, and the outcome. That captures the delta between the written process and what your team actually does: the institutional knowledge that normally lives in senior analysts' heads and Slack threads. Capturing those decisions is the data-model groundwork; what the platform does with them — recalling the right precedent on the next alert and adapting to your team's judgment over time — is the memory and learning the next two sections cover.
4. **Read the graph, and keep it yours:** Re-querying the SIEM, EDR, and IAM from scratch on every alert is slow, costly, and punishing to the systems you depend on. Torq's agents reason from a single, shared, current, normalized layer, so investigations compound rather than restarting at zero. And because that layer encodes your most sensitive operational truth, isolation is an architectural decision; each tenant's graph is its own store, and your data never enters a shared pipeline.

Memory: Torq Recall

Context grounds an agent in the present. Memory grounds it in the past. In most security operations centers, there is none: context disappears the moment a case closes, and when a new alert fires, the investigation starts from zero. Analysts fall back on search interfaces to find similar past cases, but in an agentic SOC, even great search is still manual work. Relevant precedent should be retrieved automatically at triage time.

Without that memory, analysts repeatedly reinvestigate the same benign patterns the organization already dismissed, inflating false-positive noise and burning out teams. And when a senior analyst leaves, their institutional knowledge leaves with them.

Why Generic Retrieval Fails in the SOC

Retrieval-Augmented Generation (RAG) has become a popular way to surface historical data, but standard semantic RAG can't achieve the precision a SOC requires. Generic RAG finds semantically similar information, which works for documents and natural language. It breaks down when the meaning lives inside security entities such as IP addresses, file hashes, URLs, and hostnames. Two hash-like values — *5jshdh2kfw* and *5jshdh2yfw* — look close to a model but are entirely different identifiers. A near match is not evidence, and in a high-stakes environment, "close enough" shatters analyst trust.

Torq Recall rejects semantic matching in favor of structured, deterministic retrieval: exact overlaps of specific observables (e.g., IPs, file hashes, URLs, hostnames, emails) trigger a historical match, so every recommendation is auditable, explainable, and grounded in verifiable evidence from your own environment. Exact matching is only the first step; Torq Recall also weighs signal strength. A match on a rare file hash carries strong precedent. A match on a common binary like explorer.exe or a shared corporate IP is much weaker.

Honest AI in a Messy SOC

Real SOC data is rarely clean. Alerts arrive with partial context, telemetry is noisy, and human decisions are inconsistently captured. A system designed in a sterile lab fails on contact with that ambiguity. Torq Recall was built to evaluate the quality and consistency of precedent (not just to point at it), using a subagent architecture that acts like a veteran analyst: weigh the evidence, spot inconsistencies, and know when to ask questions.

Real-World Scenario	What Was Encountered	How Recall Handles It
Conflicting precedent	Half of matching alerts closed as false positives, half escalated as true positives	Refuses to pick arbitrarily; lowers confidence, flags the split, and recommends "investigate further"
Misleading labels	Case labeled "true positive/malicious," but notes say "part of a concluded red team exercise"	Prioritizes the narrative in the notes over the static label and adjusts accordingly
Severity mismatch	Historical cases on a shared IP were critical attacks; the new alert is low-severity informational	Flags the mismatch so it doesn't inherit escalation precedent from a different attack type
Weak signal overlap	Match relies on a single low-fidelity observable, like a common corporate IP	Acknowledges the weak precedent, returns low confidence, declines a definitive call
Inconsistent data entry	Coded "false positive," but notes describe actively blocking malicious activity	Flags the contradiction and lowers confidence to prevent trusting bad data

By turning normal daily actions (e.g., resolving a case and leaving a note)into continuous institutional memory, Recall makes the SOC's own history available at machine speed, without adding a single click to the analyst's workflow.

Learning: Torq Reflex

Context and memory close the gap when the AI is missing information. But in a review of analyst corrections across three customer environments, roughly half the disagreements weren't information gaps at all. The AI had everything the analyst had and still landed somewhere the team wouldn't: contradicting its earlier verdicts, treating a contained threat as resolved when policy called for escalation, or accepting thin evidence the team would have rejected. That gap is about judgment, and no amount of extra data closes it.

The First-Generation Ceiling

First-generation AI triage judges every alert from scratch, and when an analyst corrects it, the correction is added to the prompt rather than the model. Researchers call the result self-inconsistency, an inherent failure mode of LLMs. When we analyzed analyst corrections across enterprise environments, three failure modes kept breaking SOC trust:

1. **Verdict inconsistency.** The same alert is classified as benign on Monday and critical on Tuesday.
2. **Policy divergence.** The same blocked execution is escalated by one team member and closed as benign by another.
3. **Evidentiary threshold disagreement.** The AI and the analyst see the same data, but the AI lacks the judgment to demand more corroborating evidence before calling it malicious.

These aren't information gaps; the AI has the data. They are judgment gaps. And no prompt fixes them: consistency takes a model that actually remembers what it decided, and a team's risk tolerance only becomes visible across dozens of messy edge cases, well beyond what any single written rule can capture.

How Torq Reflex Works

Torq Reflex is a per-tenant model trained on how your team specifically rules, learning more every time an analyst corrects it. Rather than pulling the nearest past case off a shelf, Reflex applies what it has internalized to judge alerts it has never seen.

The flow is simple. An alert arrives; Reflex assesses its confidence; if confidence is high, its verdict shapes the output. If not, the full LLM analysis runs, and the case goes to an analyst, exactly as it would without Reflex. Whatever the analyst decides flows back in and retrains the model. Because Reflex lives inside the Torq AI SOC Platform, a confident verdict doesn't just label an alert and stop. It feeds Auto Triage and case creation, then Socrates™ orchestrates the response across Torq HyperAgents™, so the judgment your team teaches Reflex makes every downstream action across the full threat lifecycle more trustworthy.

That calibrated confidence score is the advantage of training a model instead of running a search. A lookup can tell you an alert resembles something a human once fixed; Reflex gives a real probability, which is what makes the arrangement safe. It weighs in when sure and holds back when uncertain, routing anything ambiguous to the pipeline you already trust. The same math lets you set your tolerance for the two kinds of error: by default, Reflex treats a missed-malicious verdict as the costlier mistake, and you can tune that weighting to your organization's risk posture.

Results

Standard AI agents are non-deterministic. They can look at the exact same alert twice and reach different conclusions based on the "temperature" of the model. In a security environment, that is a liability. We benchmarked Torq Reflex against standard LLM-based triage to measure operational reliability:

- **100% repeatability.** When the same alert recurs, Reflex returns the same verdict every time. Standard LLM agents drifted on more than 10% of alerts even at low-temperature settings, and dropped to roughly 55% consistency at default settings.
- **Precision under pressure.** On the highest-stakes call — confirming a real threat — Reflex gained about 22 points of accuracy over the baseline.

Reflex achieves this through tenant-specific adaptation, so the judgment your team teaches the model is stored as a persistent, durable rule rather than a transient prompt instruction. It reaches this in roughly two weeks of normal analyst feedback, then sharpens with every retraining pass. Most AI triage systems perform the same on day 100 as on day one. Reflex draws a curve you can put in front of a board.

Built for Oversight and Isolation

The model is yours, not anyone else's. Torq does not pool training data across customers or share parameters between them. Because Reflex learns through lightweight, tenant-specific adapters, each tenant maintains its own compact judgment layer that remains entirely isolated, enabling the model to learn from your analysts' behavior without the risk of parameter sharing or data leakage.

That data discipline is also what makes genuine human-on-the-loop operation possible: because Reflex produces a calibrated confidence score grounded in your team's own calls, the platform can act on the cases it's sure about and route the uncertain ones to a person, with an auditable record of what the model decided and why. This aligns with the EU AI Act's high-risk oversight and transparency requirements, which take effect in August 2026.

The Layer, Together

Context, memory, and learning are not three features. They are one layer with three jobs:

- **Context** grounds the agents in your environment as it is right now.
- **Memory** gives them precedent from how your SOC has already ruled.
- **Learning** teaches them the judgment your analysts apply when the rules run out.

Underneath the Torq AI SOC Platform, the Context Graph, Recall, and Reflex form this shared substrate beneath every agent. Auto Triage, Socrates™, and Torq HyperAgents™ all reason from the same grounded, remembered, and learned foundation, enabling the platform to run the full threat lifecycle — triage, investigation, response, and remediation — autonomously while keeping analysts on the loop for the judgment calls that genuinely need them.

This is the part most platforms skip, because it's hard to build and harder to keep correct. You can swap the reasoning engine next quarter. What compounds over time is the layer that knows your environment, remembers your decisions, and learns your judgment.

What This Means for Your SOC

The next model release will not fix grounding, memory, or judgment, because that knowledge isn't in the weights. It's in your environment, your history, and your analysts. When you evaluate an AI SOC platform, look past the demo and ask the layer-beneath questions:

- Does it maintain a current, traceable, isolated model of our environment, or reason from a generic one?
- Does it retrieve our own past resolutions deterministically, or rely on fuzzy similarity that can match the wrong evidence?
- Does it actually learn our team's judgment in a model that remembers, or just rewrite prompts and look up similar cases?
- Can it show its reasoning, route uncertain cases to a human, and keep our data ours?

A platform that answers yes turns adoption into compounding value. One that doesn't give you faster wrong answers.

Conclusion

Reasoning over an alert was the breakthrough of the last two years, and it is now table stakes. What separates an AI SOC you can trust from one that produces fast, confident, wrong answers is the layer underneath: context that grounds a decision, memory that gives it precedent, and learning that bends it toward your team's judgment.

That layer is an engineering discipline, not a model feature, which is exactly why it is hard to copy and why it compounds. Anyone can point a capable model at an alert. Far fewer have built the bitemporal graph and captured the decisions, and the advantage widens with every shift. Torq built that foundation in production: the Context Graph for grounding, Recall for memory, and Reflex for learning. The result is an AI SOC that gets sharper with every alert prioritized, every case closed, and every correction your team makes.

So the question to put to any platform is not whose agent reasons best on day one. Its whose foundation is still learning on day 100.

References

[1] SANS 2025 Detection and Response Survey

[2] IBM, Cost of a Data Breach Report 2025

[3] Torq, The 2026 AI SOC Leadership Report