nagarro

# Metrics that matter for Generative AI evaluation

Why good scores mask bad systems and what to measure instead

# Abstract

Generative AI (GenAI) has moved rapidly from research curiosity to production dependency. Customer support bots now resolve live service issues. Clinical summarization tools feed physician workflows. Code generation agents write software that may eventually ship. Yet the measurement frameworks governing these systems often still resemble those used for a different generation of AI: classification and regression models with stable inputs, bounded outputs, and clearer definition of correctness.

This creates a dangerous illusion. Dashboards can look healthy while users report frustration. Safety metrics can stay green while harmful or non-compliant outputs reach production. Engineering teams can optimize a system against a high benchmark score and still struggle to explain why adoption is low, escalation rates are rising, or business stakeholders do not trust the result.

The issue is not simply that GenAI systems are harder to measure. It is that many evaluation programs measure what is easy to compute rather than what determines whether the system works in context. A response can be fluent, factual, and well-formatted, and still fail because it does not help the user complete the task, does not follow the required process, or cannot be trusted in the workflow where it is deployed.

The paper proposes a practical framework for evaluating GenAI systems in production. It organizes quality around five dimensions that matter in deployed systems: Utility, Truth, Safety, Reliability, and Experience. It introduces a layered evaluation stack that connects technical signals to business outcomes and business value. It shows how metric selection changes by architecture and use case, from conversational AI to RAG and multi-agent systems. Finally, it outlines a KPI design method and evaluation flywheel that help organizations move from one-time testing to continuous quality improvement.

The central argument is simple: good scores do not guarantee good systems. GenAI evaluation must start with the user outcome, work backwards to the failure modes that would prevent it, and continuously measure whether the system is delivering value safely, reliably, and in the context where it is meant to operate.

nagarro

## Table of contents

nagarro

## Table of contents

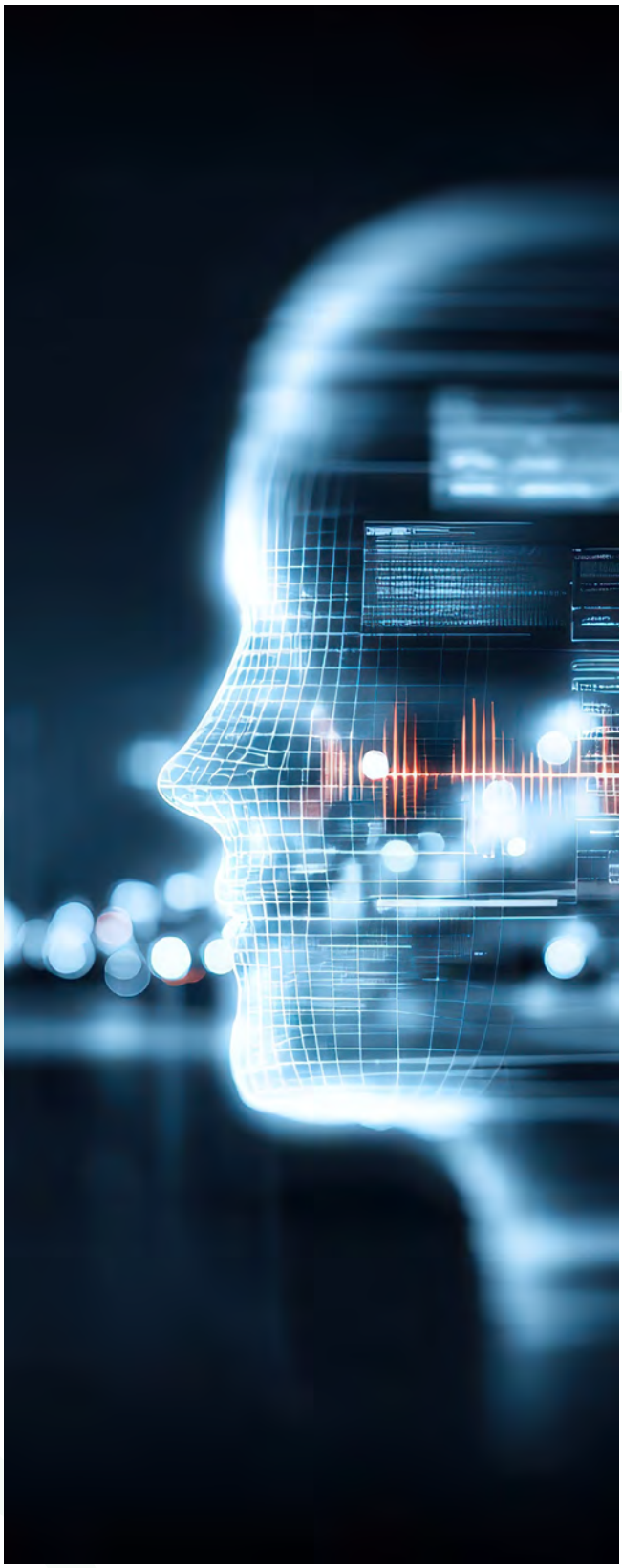
# Why GenAI evaluation needs a new starting point

Most organizations do not fail with GenAI because they forgot to measure. They fail because they measure the wrong things.

A system may pass a benchmark, achieve high faithfulness on a held-out dataset, produce fluent responses, and still disappoint in production. The customer does not care whether the answer was well-formed if the issue was not resolved. The clinician does not care that the summary was grounded if it takes too long to use during a consultation. The compliance officer does not care that the conversation ended successfully if a required action happened in the wrong sequence.

This is the central evaluation challenge for GenAI: system quality is not a property of the model alone. It is a property of the model, the architecture around it, the data it retrieves, the tools it calls, the users it serves, and the business process into which it is inserted.

For that reason, GenAI evaluation must move beyond generic model scores. It must become a design discipline: define what success means for the user, identify the failures that would matter most, select metrics that expose those failures, and connect those metrics to the outcomes the organization actually cares about.



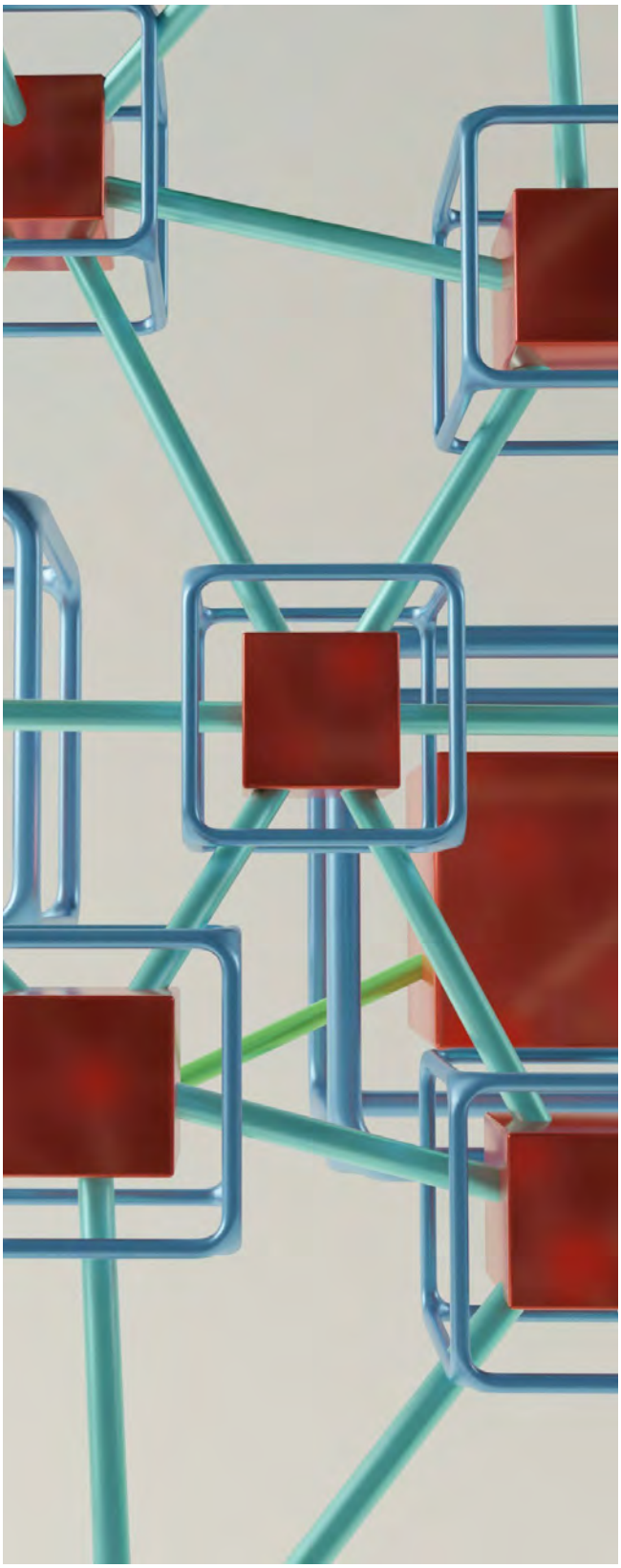
# The ‘measurement’ problem

The history of AI evaluation is, in large part, a history of proxy metrics. Because the things we care about, helpfulness, trustworthiness, safety, and real-world utility, are difficult to measure directly, the field has long relied on proxies: scores that correlate with quality under controlled conditions and are easy to compute at scale.

For classical machine learning, this worked reasonably well. A spam classifier's accuracy on a held-out test set genuinely predicted its performance in production, because the task was stable: the input space was well-defined, the output was a binary label, and the definition of "correct" did not change with context. The proxy and the thing it proxied were tightly coupled.

GenAI breaks this coupling. When the output is free-form text, a customer service response, a clinical note summary, or a piece of generated code, the space of acceptable outputs is vast and context-dependent. A response can be grammatically perfect, factually accurate, tonally appropriate, and still completely useless because it does not address what the user was asking for. The proxy and reality diverge, and evaluation frameworks built on the old assumptions become actively misleading.

This is not a minor calibration problem. It is a category error. Applying accuracy, precision, F1, and BLEU to generative systems is not merely imprecise; it can produce conclusions about system quality that do not reflect real-world performance. When teams optimize these metrics, user satisfaction may not improve at the same rate.



# The production failure pattern

The following examples illustrate how gaps between evaluation design and deployment reality tend to surface. Each scenario reflects a class of failure that evaluation program can be designed to catch.

- **Customer support bot:** Consider a conversational assistant built for a subscription service, evaluated response fluency, tone, and factual accuracy against a product documentation corpus. Pre-launch scores are strong across all three dimensions. The system goes live, and within weeks, customer disputes begin rising. The refund policy had been updated after the evaluation corpus was assembled, and the bot continued citing the old terms. The metric that would have caught this, specifically whether retrieved knowledge reflects the current state of business rules, was not part of the evaluation program. The gap was not in the model; it was what the program chose to measure.
- **Clinical summarization:** A RAG-based summarization tool designed for cardiac consultations is evaluated for faithfulness to source documents and clinical accuracy, both of which it achieves at a high level. What the evaluation does not measure is whether the output is usable within the time constraints of an actual consultation. In practice, the summaries average over 300 words at a reading complexity suited to academic literature, in a setting where a clinician has roughly 90 seconds per patient. Adoption is low and falls further over the first weeks of deployment. The system performs well on the dimensions it was assessed; the workflow dimension was not assessed.
- **Coding agent:** A code generation agent passes evaluation on functional correctness, style compliance, and latency. The evaluation suite does not include a review of caching behavior or token lifecycle management. The generated code uses a caching approach without an expiry policy, meaning authentication tokens remain cached past their intended validity window. A security issue surfaces within days of deployment. The evaluation program had full coverage of what it was designed to cover, and no coverage of this particular failure class.
- **Content moderation agent:** An agent designed to assist support teams with content decisions is tuned to achieve a strong safety score. As the safety threshold is lowered to improve that score, a rising proportion of legitimate queries are declining. Agents find the tool increasingly unhelpful and stop using it. The safety metric improves as measured; the utility cost of that improvement is not tracked alongside it.

Each of these scenarios point to the same underlying dynamic: evaluation programs are designed around anticipated failure modes, and the failures that reach production tend to be the ones that were not anticipated. The practical response is not to design a perfect program upfront, but to build mechanisms for discovering and closing coverage gaps as the system operates.

# What “good” means for GenAI

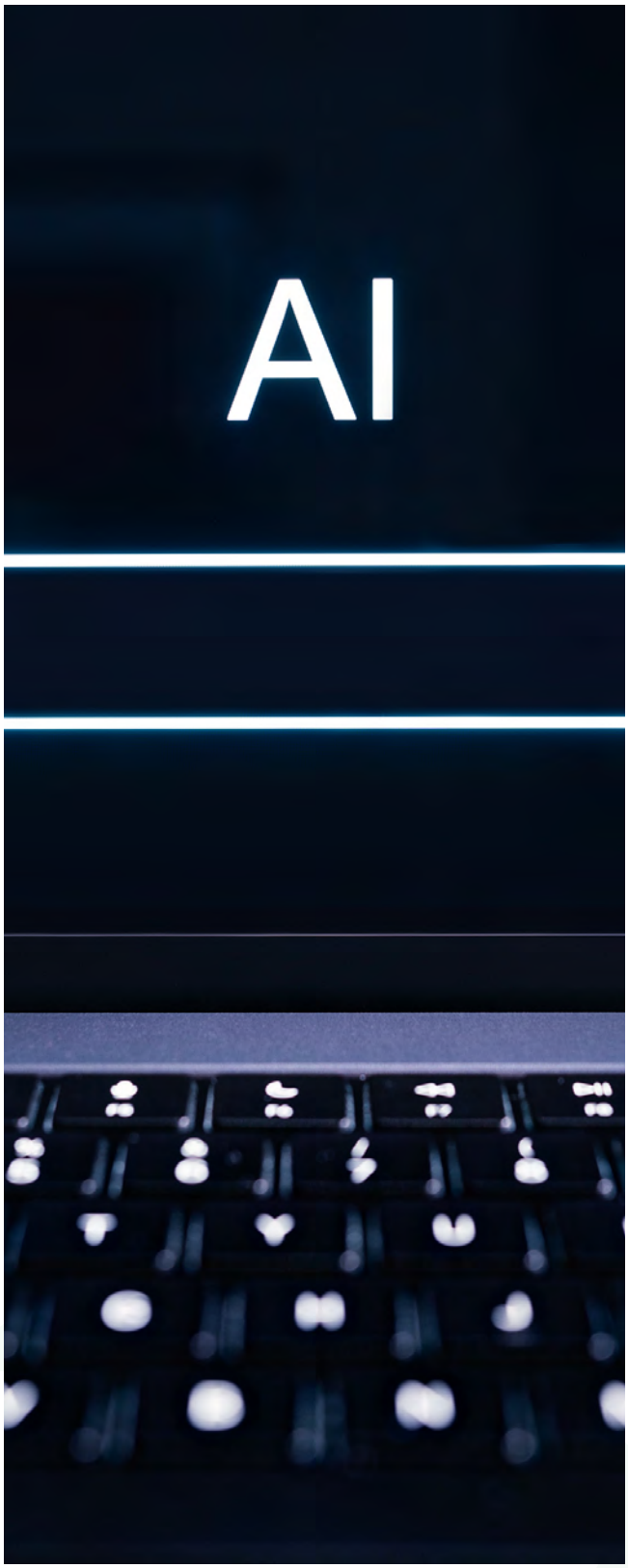
## Challenges with starting from the model, not the user

The most common mistake in GenAI evaluation design is when we begin with the model. Teams ask: "What can we measure about this model's outputs?" They then build a dashboard around the answers, fluency, coherence, accuracy, hallucination rate, and treat that dashboard as the evaluation program. It is not. It is a component of an evaluation program, and not the most important one.

The right starting point is the user. What does success look like for a real person, attempting a real task, in a real context? This question has a specific answer in every deployment, and that answer determines what must be measured. A customer who phones a support line to dispute a charge has one definition of success: the charge is resolved fairly and quickly. A cardiologist reviewing a patient's summary before a consultation has a different definition: they can identify the relevant history and make a decision in under two minutes. A software engineer using a code generation assistant has a third definition: the generated code works, is secure, and does not require significant rework.

None of these success criteria are fully captured by generic model metrics. Issue resolution rate, clinician decision time, and code security pass rate are the signals that matter most. They are not difficult to reason about; the challenge lies in instrumentation. Surface-level scores that can be computed on a held-out dataset are faster to build and run, which is why they often serve as the starting point.

An important principle follows from this: an output that is technically correct but practically unusable has the same business impact as a wrong output. Evaluation must account for the full path from output to outcome, not just the output itself.



# What “good” means for GenAI

## Inherent tension between quality dimensions

One of the things that makes GenAI evaluation genuinely difficult, beyond the technical complexity, is that the quality dimensions a system must satisfy are often in tension with each other. Optimizing aggressively on one dimension frequently degrades another. An evaluation program that ignores these trade-offs will produce systems that are excellent on the measured dimension and degraded on the unmeasured one.

The tension between safety and utility is the clearest example. A content moderation agent whose safety score is the primary evaluation target will, over time, have its refusal threshold tuned downward to improve that score. The consequence is a rising false-positive rate, where legitimate queries are declined because they superficially resemble policy violations. In the case described earlier, 34% of legitimate support queries were blocked, and agents moved away from the tool within a week.

The tension between faithfulness and fluency operates in the opposite direction. A RAG-based system that is required to ground every claim in retrieved source material will produce outputs that are cautious, heavily qualified, and often dense with citations. This is appropriate for high-stakes domains where unsupported claims carry legal or clinical risk. In lower-stakes conversational settings, the same behavior can produce outputs that users find dense or difficult to read, reducing engagement.

The tension between accuracy and confidence is particularly dangerous. Models that are penalized for expressing uncertainty become systematically overconfident: they produce fluent, assured responses even when the underlying information is ambiguous or absent. This creates hallucination in its most harmful form, not the obviously wrong claim that a reader would catch, but the plausibly stated incorrect claim that goes unchallenged because it is presented with apparent confidence.

Effective evaluation frameworks do not resolve these tensions by picking a winner. They make the trade-offs explicit, quantify the cost of each compromise, and design measurement programs that monitor all the affected dimensions simultaneously. The goal is not to eliminate trade-offs; it is to make them visible and deliberate so they can be managed consciously.

Quality in a generative AI system is not a single score. It is a multi-dimensional balance between competing requirements, and the role of an evaluation program is to make that balance explicit rather than collapsing it into a single number.

# The five-pillar framework for GenAI quality

## Why pillars, not a single score

One natural response to the complexity of GenAI evaluation is to aggregate signals into a single quality score, a weighted composite that can be tracked on a dashboard. While intuitive, this approach has a significant limitation. A composite score obscures the information that matters most: which specific dimension of quality is failing, for whom, and in what context.

A system with a composite quality score of 0.78 could be failing badly on safety while excelling on fluency. It could be highly reliable but deeply unhelpful. It could produce factually grounded outputs that users consistently refuse to engage with. The composite score says 0.78. The user says the system is broken. A composite score makes it difficult to identify which specific dimension needs attention.

The five-pillar framework proposed here maintains dimensional separation throughout the measurement program. Each pillar is measured independently, reported independently, and owned by a specific organizational function. Any aggregation is best done at the level of business decisions, informed by the individual pillar scores.

## Five pillars

The framework decomposes GenAI quality into five mutually necessary dimensions. A system that excels on four and fails on one is a failed system, because quality in all five pillars is a precondition for the system to deliver value to users reliably.

# The five-pillar framework for GenAI quality

## Pillar 1: Utility

Utility asks the most basic question: does the output help the user accomplish what they came to do? It encompasses task completion rate, whether the user's goal is achieved, relevance to intent, and the degree to which the system demonstrates genuine helpfulness rather than merely generating a plausible-sounding response.

The failure mode for utility is outputs that are fluent, safe, and factually grounded but do not address the user's actual need. This is the "accurate but useless" pattern: a system that produces correct information that is irrelevant to the question asked, or that answers a simpler version of the question the user had. Helpfulness is easy to misconstrue as a soft metric. In practice, it is the hardest signal to fake and the most consequential to get right.

## Pillar 2: Truth

Truth encompasses factual accuracy, faithfulness to source material, and grounding, particularly in RAG architectures where the system is expected to generate outputs anchored in retrieved documents. It comprises a system that makes logically correct inferences from correct premises when those inferences go beyond what the source material supports.

The failure mode for truth is hallucination: the generation of confident, plausible, and incorrect claims. What makes hallucination particularly dangerous in production is that it is often undetectable by users without domain expertise.

## Pillar 3: Safety

Safety covers three related but distinct concerns: the presence of harmful, toxic, or discriminatory content; the system's compliance with operator-defined policies and constraints; and the consistency of quality across different user demographics. A system that produces different quality outputs for users of different demographic backgrounds is not merely a fairness problem; it becomes a safety concern because it introduces systematic disadvantage that compounds over millions of interactions.

## Pillar 4: Reliability

Reliability asks whether the system performs consistently across varied inputs, repeated queries, and edge cases. A system that gives correct answers to clean, well-formed queries but degrades under paraphrased inputs, typos, or ambiguous phrasing is not production ready. A system that produces different answers to the same question on consecutive runs, without any change in input, has a robustness concern that will erode user trust over time even if individual outputs are high quality.

For agentic systems, reliability extends to agent stability: does the agent complete multi-step tasks consistently, or does it fail unpredictably at step three of a five-step workflow? Agent instability is particularly significant because failures in the middle of a task carry higher cost than failures at the start, as the user has invested time, and the system may have taken irreversible actions.

## Pillar 5: Experience

Experience measures the qualitative dimensions of interaction that determine whether users choose to continue using the system: fluency and naturalness of language, appropriate tone calibration for the context, session completion rates, and, most tellingly, whether users return. A system that is technically correct on the first four pillars but produces stilted, robotic, or tonally inappropriate output will see adoption rates decline over time as users find

# The evaluation stack

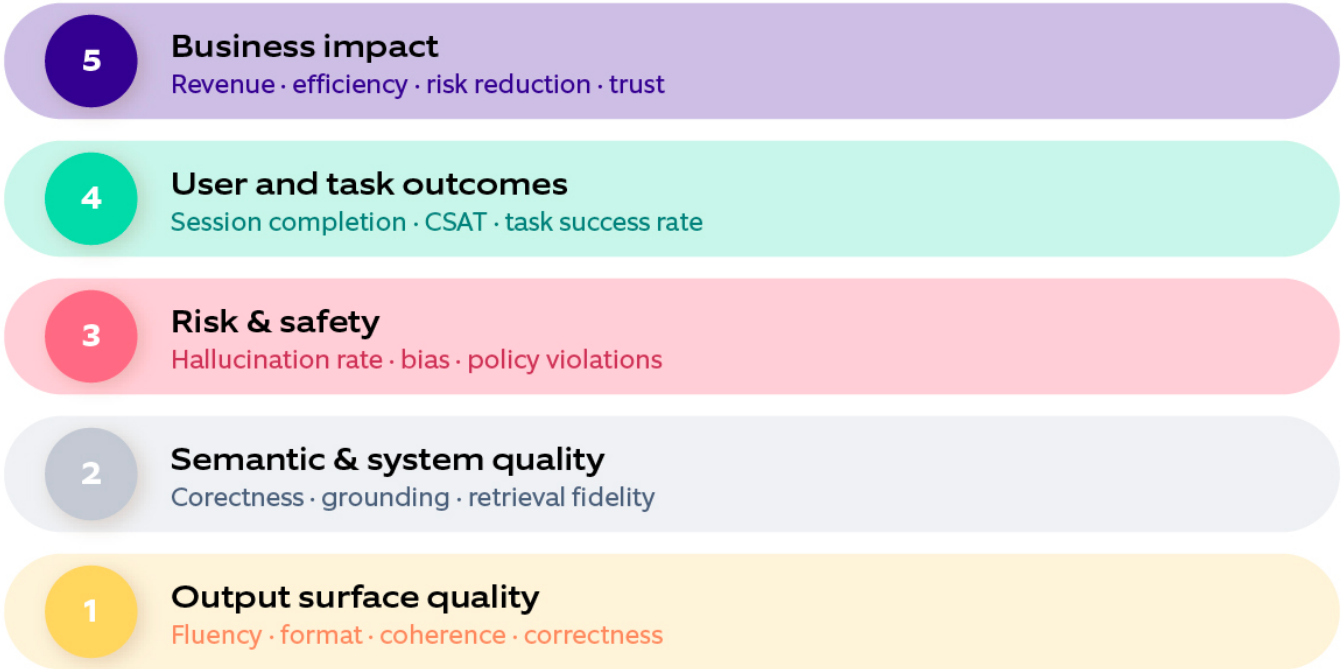
## Why layering matters

Even with the five-pillar framework in place, there is a second structural challenge: the metrics that are easiest to compute are the furthest from the outcomes that matter most. Fluency is easy to measure automatically. Issue resolution rate requires tracking the customer's subsequent behavior. Hallucination rate can be estimated on a held-out dataset. Clinical adoption rate requires deployment in a real hospital workflow. The gap between "measurable now" and "what actually matters" creates a systematic bias toward the measurable, independent of its importance.

The evaluation stack addresses this by organizing metrics into five layers of abstraction, from output surface quality at the base to business impact at the top. The stack complements rather than replaces the five-pillar framework. The pillars define what to measure; the stack defines the relationship between signals at different levels of abstraction and the organizational functions responsible for acting on them.

## The five layers

The framework decomposes GenAI quality into five mutually necessary dimensions. A system that excels on four and fails on one is a failed system, because quality in all five pillars is a precondition for the system to deliver value to users reliably.



# The evaluation stack

## Layer 1 (Output surface quality)

captures what can be measured directly on the text of the output: fluency, grammatical correctness, format compliance, length appropriateness, and completeness. These metrics are reliable, fast, and automatable, which makes them useful for continuous integration testing and quick regression checks. Their limitation is that they measure how the output looks, not what it does. A perfectly fluent, correctly formatted hallucination scores well at Layer 1.

## Layer 2 (Semantic and system quality)

moves inside the output to measure meaning: factual correctness, faithfulness to retrieved context, grounding quality, and logical coherence. For RAG systems, this layer is where retrieval quality metrics live, including contextual precision, contextual recall, and faithfulness. Layer 2 metrics require more sophisticated evaluation methods, typically including LLM-as-Judge or embedding-based approaches, and are more expensive to compute than Layer 1.

## Layer 3 (Risk and safety)

measures the dimensions that can cause the most serious harm if not caught: hallucination rate (distinct from faithfulness in that it measures the proportion of interactions containing any fabricated claim), toxicity, bias across demographic groups, and policy compliance. These metrics are not always easy to interpret in isolation, a hallucination rate of 2% means different things in a children's homework assistant and a clinical decision support tool, but they must be monitored continuously in any production system.

## Layer 4 (User and task outcomes)

is where the signal gets closest to what users actually experience: session completion rate, task success rate, customer satisfaction score, re-open rate, and escalation rate. These metrics require production data and are therefore only available after deployment, which creates a bootstrapping challenge for pre-launch evaluation. Offline proxies can be designed, for example using simulated user sessions or expert panel task completion assessments, but they are not a full substitute for live data.

## Layer 5 (Business impact)

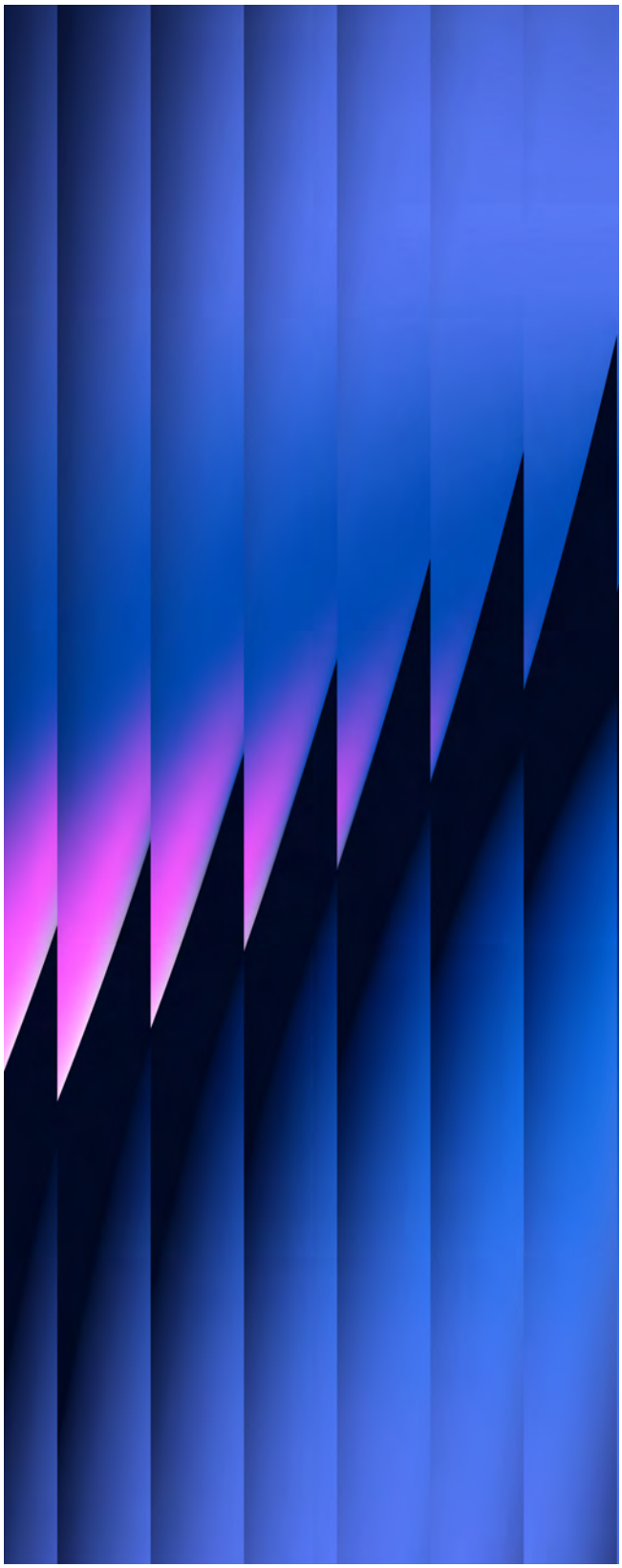
translates the signal from all lower layers into the language of organizational decision-making: revenue impact, cost efficiency, risk reduction, and institutional trust. These are the metrics that determine whether the AI program receives continued investment. They also require collaboration with finance, operations, and risk functions rather than engineering teams working in isolation.

# Metric selection by architecture and use case

Before considering use-case-specific requirements, the system architecture constrains the appropriate metric set. A plain LLM completion endpoint, a RAG pipeline, and an agentic system with tool use are evaluated differently because they have different failure modes, different internal components that can degrade independently, and different relationships between what the system does and what the user experiences.

The following case studies illustrate how the evaluation principles described in this paper apply to real-world GenAI deployments. Each case presents a representative project context, the challenges encountered when applying standard metrics, and the evaluation adjustments that produced actionable insight.

The examples are anonymized client cases drawn from GenAI evaluation work across different domains and architectures. Client-identifying details have been removed or generalized, but the evaluation challenges, metric gaps, and lessons reflect real production patterns observed in practice.



# 1. Conversational AI: Financial services complaint handling AI

**Customer complaint resolution bot**
**Domain:** Financial Services – Retail Banking
**Architecture:** Conversational AI (LLM with tool access)

## Project context

A large retail bank deployed a conversational AI assistant to handle first-line customer complaints across digital channels. The system was expected to resolve complaints without agent escalation in the majority of cases, reduce average handling time, and maintain compliance with financial conduct regulations that require complaints to be acknowledged and logged within defined timeframes.

The initial evaluation framework borrowed from the bank's existing chatbot program: intent classification accuracy, response time, and a customer satisfaction score collected via post-conversation survey. After six weeks in production, escalation rates were higher than the pre-deployment baseline and CSAT had declined. The model's intent accuracy was 94%. The system was, by its own metrics, performing well.

## Challenge: Metrics that missed the regulatory dimension

Post-incident analysis revealed three failure modes that the evaluation framework had not surfaced.

- First, the system was not consistently logging complaints in the core banking system before attempting resolution, a regulatory requirement, because the tool-call sequence was occasionally out of order. Task completion rate, as measured, checked only whether the conversation ended without escalation, not whether all required actions were performed in sequence.
- Second, the system was resolving some complaint types with responses that were factually accurate but outside the bank's approved communication framework for regulated matters. Factual accuracy was high; policy alignment was not measured.
- Third, CSAT was collected from a self-selected subset of customers who completed the post-conversation flow, systematically excluding customers who had abandoned the conversation in frustration.

## 1. Conversational AI: Financial services complaint handling AI with to

### Evaluation adjustments

- Task completion was redefined to require ordered step validation: complaint logged → investigation initiated → resolution communicated → closure confirmed. Any deviation from this sequence counted as a failure regardless of the conversation outcome.
- Policy alignment scoring was introduced using a rubric-based LLM judge trained on the bank's approved communication templates for regulated complaint categories.
- Abandonment rate was added as a primary metric, tracked from conversation-level session data rather than post-conversation surveys.
- A targeted red-teaming program tested edge cases in regulated complaint categories, specifically, complaints involving product mis-selling, where the consequences of policy non-compliance were most severe.

| Metric | Target | Measured result | Status |
|---|---|---|---|
| **Sequential task completion rate** | > 95% | 89% → 97% (post-fix) | ✓ Met |
| **Policy adherence rate** | > 90% | 78% (baseline); 93% (after prompt revision) | ✓ Met |
| **Conversation abandonment rate** | > 12% | 23% → 11% | ✓ Met |
| **Escalation rate** | > 20% | 31% → 18% | ✓ Met |
| **Compliance violation count** | 0 | 4 (pre) → 0 (post) | ✓ Met |

**Key takeaway:** Conversational AI in regulated industries requires sequence-aware task completion metrics, not binary resolution rates. A conversation that ends without escalation is not the same as a conversation that complied with the required process.

# 2. RAG System: Clinical decision support in secondary care

**Clinical evidence retrieval and summarization**
**Domain:** Healthcare - Secondary care /Hospital Network
**Architecture:** RAG (Retrieval Augmented Generation)

## Project context

A hospital network deployed a RAG-based clinical decision support tool to help physicians access relevant clinical guidelines, drug interaction data, and patient history summaries at the point of care. The system retrieved from a curated knowledge base comprising national clinical guidelines, formulary data, and structured EHR summaries. The intended workflow was for clinicians to query the system during ward rounds and outpatient consultations, with the system surfacing relevant evidence in under thirty seconds.

Initial evaluation focused on faithfulness (whether generated summaries were grounded in retrieved documents) and a physician satisfaction score. Faithfulness averaged 0.91 across a held-out evaluation set. Physician adoption after ninety days was 7% of the eligible clinical population.

## Challenge: High faithfulness, low utility

The adoption failure was investigated through structured interviews with clinical users. Three consistent themes emerged.

- First, retrieved chunks frequently included background sections of guidelines rather than the specific recommendation relevant to the query; contextual precision was poor, and the generator was faithfully summarizing the wrong parts of the right documents.
- Second, the system had no mechanism for flagging when retrieved guidance was out of date relative to the patient's current medication list, meaning clinicians could not trust the output without manual verification, defeating the purpose of the tool.
- Third, query latency under concurrent load exceeded forty seconds, which was incompatible with ward round workflows.

Faithfulness of 0.91 was real but it was simply not the right metric to measure whether the system was useful for clinical decision-making.

## 2. RAG System: Clinical Decision Support in secondary care

### Evaluation adjustments

- Contextual precision was measured for every retrieval event, with a target threshold above 0.75. Queries producing precision below threshold triggered a retrieval strategy review.
- A retrieval recency score was introduced: every retrieved chunk was timestamped against the knowledge base update log, and outputs referencing guidance more than twelve months old were flagged with a staleness indicator in the UI and counted in a separate metric.
- Clinician decision time, the time between a system response and a documented clinical action, was added as a proxy for utility, measured through EHR integration.
- End-to-end latency under concurrent load (P95) was added as an infrastructure KPI with a hard target of under twenty seconds.
- Clinician adoption rate was set as the primary business metric, reviewed monthly against a target trajectory.

| Metric | Target | Measured result | Status |
|---|---|---|---|
| **Faithfulness score** | > 0.88 | 0.91 (stable) | ✓ Met |
| **Contextual precision** | > 0.75 | 0.61 (baseline) → 0.79 | ✓ Met |
| **Stale retrieval rate** | < 5% | 18% → 4% (after KB refresh) | ✓ Met |
| **Time to decision** | < 45s | Not previously measured → 38s | ✓ Met |
| **P95 query latency** | < 20s | 52s → 17s | ✓ Met |
| **Clinician adoption rate** | > 35% | 7% → 41% | ✓ Met |

**Key takeaway:** In RAG systems, faithfulness measures grounding, not usefulness. Contextual precision and temporal freshness are the metrics that determine whether a clinician (or any expert user) can act on the output.

# 3. Multi-agent system: E-commerce Order Lifecycle Management

**Autonomous Order Management and Exception Handling Agent**
**Domain:** Retail & e-commerce
**Architecture: Multi-agent pipeline (Orchestrator + Specialist sub-agents)**

## Project context

A large e-commerce retailer deployed a multi-agent system to manage post-purchase order exceptions autonomously, covering delivery failures, return initiations, payment disputes, and supplier escalations. The system comprised an orchestrating agent that triaged incoming exception cases and four specialist sub-agents: a delivery resolution agent (integrating with carrier APIs), a returns processing agent, a payments agent (integrating with the payment gateway), and a supplier communication agent. Humans remained in the loop for cases above a defined financial threshold and for cases the orchestrator classified as high-complexity.

Success was defined as autonomous resolution of eligible exceptions within four hours, with a customer CSAT maintained above 4.2 out of 5. The initial evaluation framework measured end-to-end resolution rate and overall CSAT. Both looked acceptable in the first two weeks of limited deployment. At full scale, resolution rate degraded and customer complaints about incorrect resolutions increased.

## **Challenge:** The orchestrator was misrouting and sub-agents were not being evaluated independently

Root cause analysis identified two compounding problems.

- First, the orchestrator was misclassifying exception types under load, specifically, conflating delivery failures with order cancellations, which routed to different sub-agents with different permissions. A delivery failure routed to the cancellation pathway would result in an order being cancelled when the customer wanted it re-delivered. The end-to-end resolution rate did not distinguish between these outcomes because the order was technically 'resolved' in both cases.
- Second, each sub-agent had no independent evaluation coverage. The payments agent was occasionally initiating refunds before confirming that the return had been physically received, a financial control violation that would not surface in resolution rate or CSAT until customers reported receiving refunds for items they had not returned.

# 3. Multi-agent system: E-commerce Order Lifecycle Management

## Evaluation adjustments

- Orchestrator routing accuracy was added as a standalone metric, measured against a ground-truth classification set of 500 labelled exception cases spanning all four routing categories.
- Each sub-agent received an independent evaluation suite: tool call accuracy, action sequence correction, and a use-case-specific outcome metric (delivery re-attempt success rate for the delivery agent; refund-after-receipt confirmation rate for the payments agent).
- Plan adherence was tracked for the orchestrator: cases where the orchestrator's stated routing decision diverged from its executed action sequence were flagged for review.
- Financial process compliance rate was added as a zero-tolerance metric for the payments agent: any refund initiated before return confirmation was a compliance failure regardless of eventual customer satisfaction.
- Escalation precision whether cases escalated to humans genuinely required human judgment was tracked to prevent the system from using escalation as a failure-avoidance mechanism.

| Metric | Target | Measured result | Status |
|---|---|---|---|
| **Orchestrator routing accuracy** | > 97% | 84% → 98% | ✓ Met |
| **Autonomous task completion rate** | > 75% | 61% → 78% | ✓ Met |
| **Payment agent: refund compliance** | 100% | 91% (pre) → 100% | ✓ Met |
| **Average exception resolution time** | < 4hrs | 6.2hrs → 3.1hrs | ✓ Met |
| **Customer CSAT (exception cases)** | > 4.2 | 3.9 → 4.4 | ✓ Met |
| **Escalation precision** | > 90% | 73% → 91% | ✓ Met |

**Key takeaway:** Multi-agent systems require independent evaluation at every agent node, not just at the pipeline output. Orchestrator routing accuracy and sub-agent compliance metrics are often more diagnostic than end-to-end resolution rates, which can mask internal failures that have not yet surfaced as customer complaints.

# What our use cases taught us

Across deployments spanning multiple customers, multiple use cases and multiple domains, one conclusion repeated itself: the model alone does not determine success. Context does. These observations are what drove the frameworks described in this section.

## Frameworks and principles for GenAI evaluation

### Observation 1:

**Success criteria belong to the use case, not the model**

Benchmark scores measure what a model can do. They say nothing about whether those capabilities translate into the outcome a specific deployment was built to deliver. Our clinical summarization work required faithfulness, omission rate, and clinician adoption, none of which appear on standard leaderboards. Our content moderation agent required false-positive rate and legitimate pass rate, because over-refusal was as costly as under-refusal. Each use case surfaced a distinct profile of what good looks like.

The table below captures the evaluation profile that emerged from each deployment context:

| Use case | Primary goal | Priority metrics | Dominant failure mode |
|---|---|---|---|
| **Financial Services Complaint Bot** | Compliance + resolution | Sequential task completion rate, policy alignment score, session abandonment rate, escalation rate | **Out-of-sequence tool calls:** complaint logged after resolution attempt, creating regulatory breach. |
| **Clinical Decision Support (RAG)** | Faithfulness + usability | Faithfulness score, contextual precision, staleness flag rate, clinician decision time, P95 latency, adoption rate | **High faithfulness, low utility:** grounded summaries clinicians could not act on within ward round time constraints. |
| **E-Commerce Order Management (Multi-Agent)** | Faithfulness + usability | Orchestrator routing accuracy, autonomous task completion rate, payments agent refund compliance, resolution time, CSAT, escalation precision | **Orchestrator misrouting under load:** delivery failures resolved as cancellations, with no per-agent evaluation to catch financial control violations. |

# What our use cases taught us

### Observation 2: What you measure must reflect what failure costs

**Failure modes, not features, should drive metric selection:**

In every deployment, the most important design question was not "which metrics exist?" but "which failure would cost us the most?" A hallucination in a recipe chatbot results in a bad meal. The same hallucination in a medication summary could contribute to patient harm. The consequence shapes the metric, the threshold, and the human review cadence, not the other way around.

Every metric in the program should be traceable to a failure mode. If it cannot be, it is measuring something nobody has decided to care about.

### Observation 3: More metrics is not more rigor, it is less signal

A recurring pattern as our evaluation program matured was metric proliferation. Confronted with a long menu of available metrics, teams would add them all, reasoning that more coverage was better than less. The dashboards grew. The signal did not. When three metrics degraded simultaneously, nobody could agree on which one really mattered. When all metrics were green and production quality was visibly declining, nobody trusted the dashboard.

The constraint that resolved this was deliberate: track at most five metrics at any stage. This is not a compromise of rigor. A team that tracks five metrics and acts decisively on every signal consistently outperforms a team that tracks twenty and struggles to diagnose which one reflects a real problem. The discipline of choosing five forces the question that unlocks the right evaluation program: if we can only track five things, which five tell us most honestly whether this system is working for real users?

This observation, combined with the first two, produced the five-step KPI design method, a structured process for building a measurement program that is both comprehensive enough and focused enough to be actionable.

# What our use cases taught us

## Where everything converges: technical signals as business value

The final and most important pattern that emerged across all three deployments was the translation gap. Engineering teams were producing accurate measurement data. Business leaders were making decisions without it. Not because they did not want it, but because the connection between the number on the dashboard and the outcome on the Profit & Loss had never been made explicit.

Consider a clinical RAG deployment where faithfulness has improved meaningfully over a quarter. Reported as a metric, that is interesting. Reported in context, it becomes something different: if each faithfulness failure carries a compliance cost, if reduced failures translate to fewer incidents, and if fewer incidents reduce the burden on clinical review teams, the same improvement can be expressed as a cost avoidance figure, a staffing efficiency gain, or a reduction in liability exposure. The metric did not change. The translation made it actionable.

This translation is not automatic. It requires identifying the causal pathway from the technical signal to the business outcome, estimating the magnitude of the effect, and communicating it with appropriate uncertainty. But it is the step that makes evaluation matter to the organization, not just to the engineering team.

A faithfulness score is a technical measurement. A cost avoidance estimate grounded in that score is a business decision. The evaluation program only earns organizational investment when it learns to speak both languages.

*The table below maps the technical signals from our deployments to the business outcomes they ultimately predicted:*

| **Technical signal** (improving) | **What it indicates** | **Business outcome** |
|---|---|---|
| **Hallucination rate** ↓ | Outputs can be trusted without systematic review | Reduced liability, lower human review cost, stronger user trust |
| **Faithfulness score** ↑ | Claims grounded in authoritative source | Higher adoption in expert domains; fewer clinician or analyst overrides |
| **Task success rate** ↑ | Users achieve their goals through the system | Revenue preservation, efficiency gains, improved CSAT |
| **Safety / refusal quality** ↑ | Legitimate queries pass; policy violations blocked | Risk reduction, brand protection, regulatory compliance |
| **Robustness score** ↑ | Quality consistent under varied inputs | Reduced support ticket volume; stable trust at scale |
| **Step efficiency** ↑ (agents) | Tasks complete with fewer actionsinputs | Lower compute cost per query; faster time-to-outcome for users |
| **Return rate** ↑ | Users voluntarily return to the system | Adoption growth without additional acquisition cost; product stickiness |

# From evaluation framework to production practice

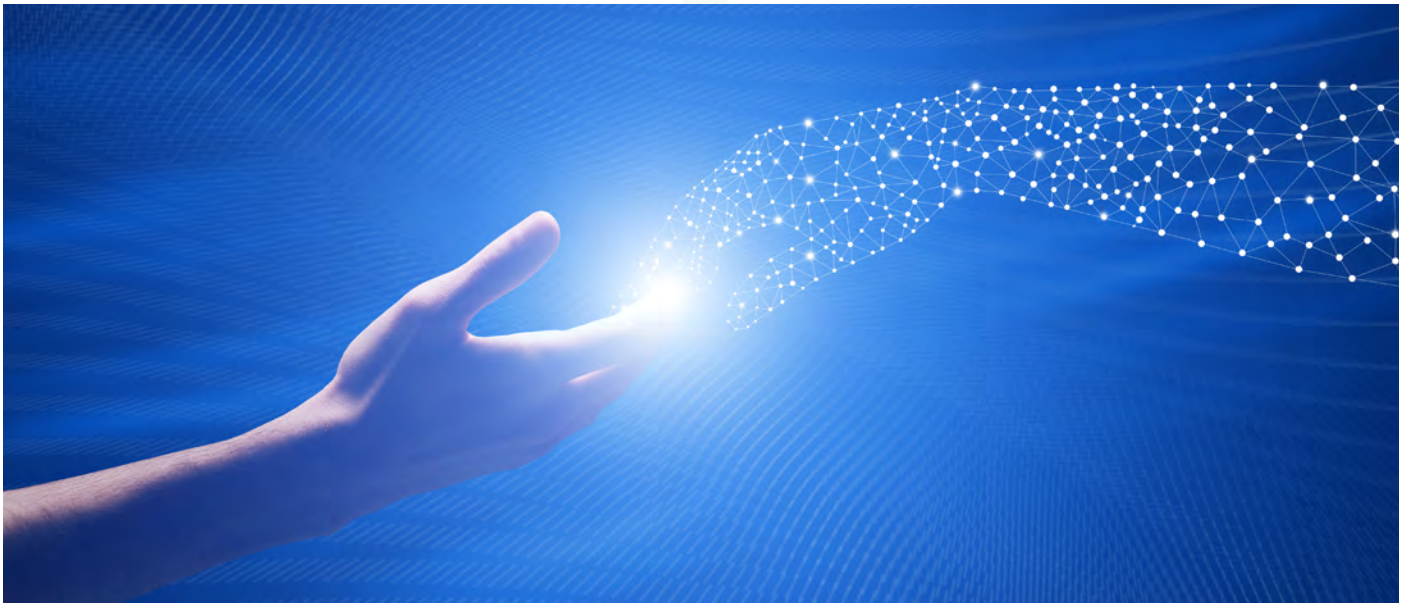
For Nagarro, the practical question is not whether GenAI can produce impressive outputs. It is whether those outputs can be trusted, governed, improved, and scaled inside real business processes.

This is where evaluation becomes an engineering discipline. A useful GenAI evaluation program does not sit at the end of a project as a launch checklist. It is designed into the solution from the beginning: in the architecture, the data pipelines, the prompt and retrieval strategy, the agent workflows, the monitoring layer, and the business KPIs used to judge whether the system is creating value.

In client work, this means helping teams translate ambition into operating discipline. For a RAG system, evaluation may require measuring retrieval precision, source freshness, faithfulness, latency, and adoption together rather than treating answer quality as a single score. For an agentic system, it may require independent evaluation of the orchestrator, the specialist agents, the tool calls, and the action sequence. For a regulated conversational AI use case, it may require policy alignment, auditability, process adherence, and escalation behavior to be measured alongside customer satisfaction.

The value of this approach is that it connects technical quality to business confidence. Product teams get clearer signals on what to improve. Risk and compliance teams get evidence that critical failure modes are being monitored. Business leaders get a more honest view of whether GenAI is reducing cost, improving speed, strengthening trust, or creating new risk.

GenAI systems will not remain static after deployment. User behavior changes, source knowledge changes, policies change, and models change. Evaluation therefore has to become a continuous capability, not a one-time assessment. Organizations that build this capability early will be better positioned to scale GenAI from pilots to production systems that are useful, safe, reliable, and aligned with business outcomes.



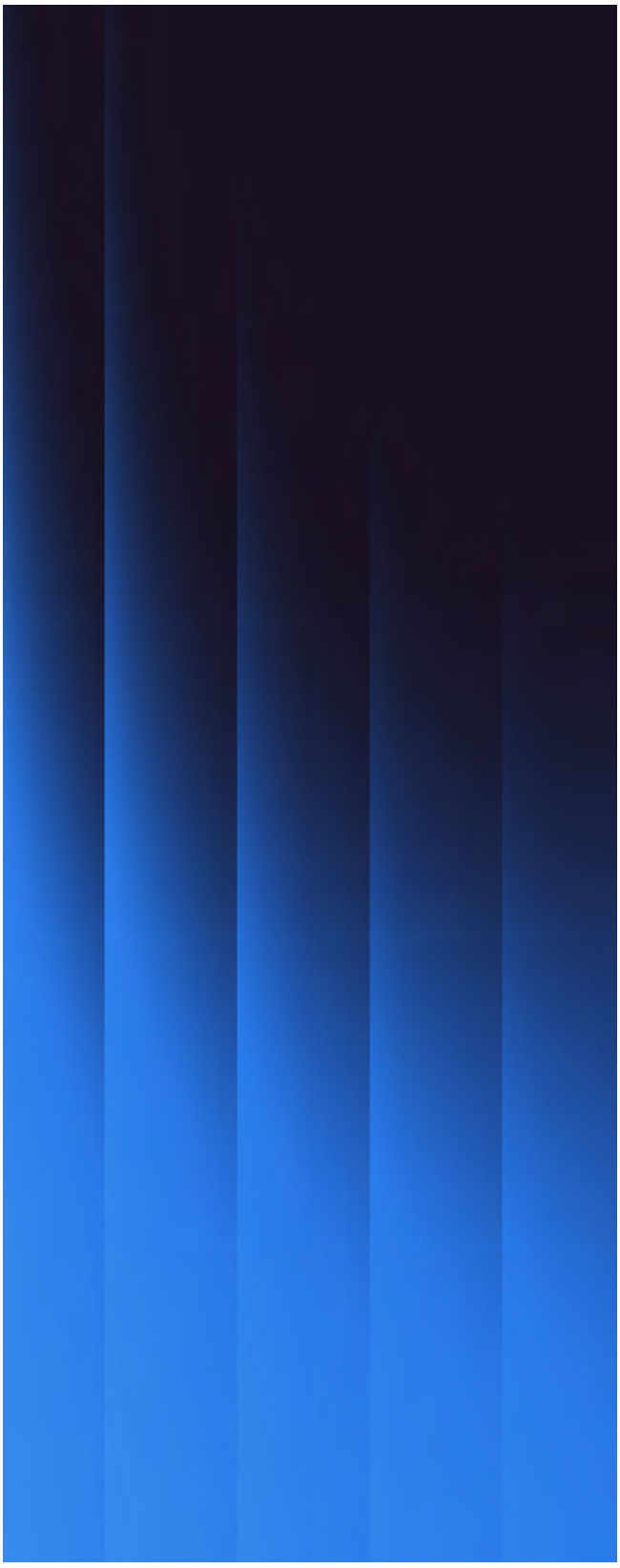
# Use case and the KPI design method

## Same model, different success criteria

One of the more counterintuitive realities of generative AI evaluation is that the model itself does not determine what success looks like. Deploying the same underlying model in two different contexts benefits from a distinct evaluation program, with different metrics, thresholds, methods, and organizational owners.

This is counterintuitive because the intuitive frame for AI evaluation comes from the model evaluation tradition: you run the model on a benchmark and get a score that applies to the model, regardless of deployment context. That frame is appropriate for comparing models. It is not appropriate for evaluating deployed systems, because a deployed system's performance is not a property of the model, it is a property of the model, the system architecture around it, the data it operates on, the users it serves, and the context in which it is deployed.

Consider a concrete example. GPT-4-class models deployed as a customer service bot for a consumer electronics company and as a clinical decision support tool for a cardiac intensive care unit are, in one sense, "the same model." In every sense that matters for evaluation, they are different systems. The consumer electronics bot must be measured on issue resolution rate, CSAT, re-open rate, and escalation rate. Hallucination is important, but the consequence of hallucination is a customer complaint, not a clinical incident. The cardiac tool must be measured on faithfulness to source, omission rate, clinician decision time, and adoption rate. A single hallucination, such as a fabricated medication dosage or a missed contraindication, could contribute to patient harm. The thresholds, the review cadence, the human-in-the-loop requirements, and the remediation urgency are entirely different, despite the shared underlying model.



# Use case and the KPI design method

## A five-step KPI design method

Designing a use-case-appropriate evaluation program is a structured process, not an art. The following five-step method produces a measurement program that covers the primary user outcome, the highest-cost failure modes, and the most likely architectural risks, while keeping the total metric count to a manageable number.

1. **Write the user outcome sentence:** Write a single sentence describing what success looks like for the user, focused on the user experience rather than system performance. "The customer gets the correct answer and does not need to call back." "The cardiologist can identify the clinically relevant history within 90 seconds." This sentence is the evaluation anchor. Every subsequent metric must trace back to it.
2. **Identify and rank failure modes:** List the specific ways this system can fail, and rank them by cost. A hallucination in a medical context carries different consequences than one in a recipe chatbot, and the ranking of failure modes should reflect those consequences. The two highest-cost failure modes each require at least one dedicated metric.
3. **Map signals to outcomes and risks:** Attach one concrete, measurable signal to each user outcome and each ranked failure mode. Prefer a mix of automated metrics (for scale) and human evaluation (for ground truth on the highest-stakes failure modes). If a signal cannot be computed automatically in the time available, identify the human process that substitutes for it.
4. **Select evaluation methods:** Automated metrics for continuous coverage; LLM-as-Judge for nuanced qualitative assessment at scale; human evaluation for calibration and launch gates. No single method is sufficient. For high-stakes domains, the human evaluation cadence should be more frequent than the standard quarterly cycle.
5. **Connect to business outcomes:** Every metric should trace to at least one named business outcome: revenue, cost, risk, or trust. If it cannot, it should either be reframed or dropped. The executive dashboard should show business outcomes; the engineering dashboard shows the technical signals that predict those outcomes.

# Use case and the KPI design method

## The evaluation flywheel

The second shift that proved durable across all three contexts was treating evaluation as a continuous process rather than a pre-launch event. In practice, this means building an evaluation flywheel: a structured feedback loop that runs continuously in production and compounds system quality over time.

**The flywheel moves through five stages:**

1. **Deploy and observe:** Every production interaction is logged: inputs, outputs, retrieved context for RAG systems, tool calls for agents, and downstream user behavior such as re-queries, escalations, and session abandonment. This logging is not just for debugging. It is the raw material of the evaluation program.
2. **Measure and flag:** Logged interactions are run through the active evaluation suite automatically. Interactions that score below threshold on any active metric are flagged for review. The suite runs on a rolling sample of live production traffic, not only on a held-out offline dataset, because the production distribution reflects what the system must handle.
3. **Analyze failures:** Flagged interactions are examined to identify root cause: retrieval failure, prompt misalignment, a model knowledge gap, or a policy violation the safety filter missed. Root cause analysis determines not just how to fix the specific case, but whether the evaluation program needs to be extended to catch the class of failure that produced it.
4. **Improve the system:** Improvements are made to the appropriate component: prompt engineering, retrieval parameters, safety filters, or system architecture. The discipline here is to address the class of failure rather than the specific case. Fixing individual cases without addressing the underlying pattern creates the illusion of progress while leaving the vulnerability intact.
5. **Validate offline:** Before re-deployment, the improved system is run against the full offline evaluation suite, including the cases that triggered the improvement. Regression testing is critical: a fix that addresses the target failure mode while degrading a related capability is a net loss. Offline validation before re-deployment is what prevents the flywheel from becoming a cycle of whack-a-mole.

Each rotation builds the dataset, refines the metrics, and improves the system. The compounding effect of running this loop consistently is the primary mechanism by which GenAI systems improve reliably over time.

# Conclusion: Measure what matters, not what is easy

Across every engagement documented in this paper, the problem was never the model. The model performed exactly as its metrics said it would. The metrics were simply not measuring the right things.

A system is not good just because its dashboard is green. A system is good because (and only when) the people who depend on it can do their work better, faster, and more safely.

We began this work believing that better evaluation was a measurement problem: Find the right metrics, instrument them well, and quality would follow. What we learned is that it is a design problem first. The question is not how to measure a generative AI system. It is what success looks like for a person attempting a task and whether the evaluation program is honest enough to surface the answer.

The five-pillar framework emerged because no single score can represent the multi-dimensional reality of a deployed system. A system that is fluent, faithful, and safe but does not help users accomplish their goals is not a quality system. The evaluation stack emerged because the metrics that are easiest to compute are structurally the furthest from the outcomes that matter and left unchecked, teams will optimize toward what is measurable rather than what is meaningful.

Good scores on the wrong dimensions do not produce good systems. They produce convincing dashboards.

The five-step KPI design method and the evaluation flywheel are the residues of repeated production failures: Signals missed, thresholds mis-calibrated, metrics without owners that degraded quietly for months before anyone noticed. Every principle in this paper has a failure behind it that has convinced us why the principle is necessary.

The discipline of asking 'what would have to be true for this system to have failed?', before launch, not after, is the single most valuable shift an evaluation program can make.

Generative AI is not going to become simpler to evaluate as it matures. Agentic systems will take more actions with more consequence. RAG pipelines will draw from larger, less curated knowledge bases. Models will be deployed in higher-stakes domains by teams with less ML experience. The gap between what can be measured automatically and what matters will not close on its own.

What closes that gap is the habit of starting from the user outcome, working backwards to the failure modes that would prevent it, and building a measurement program that monitors those failure modes continuously, not just at launch, not just when something breaks, but as a permanent and evolving function of how the organization operates its AI systems.

Evaluation is not a phase of an AI project. It is the heartbeat of a production AI system. When it stops, the system does not fail immediately. It drifts, quietly, plausibly, and often invisibly, until the gap between the dashboard and reality becomes impossible to ignore.

**Reference**

https://www.confident-ai.com/blog/llm-evaluation-metrics-everything-you-need-for-llm-evaluation

# About the authors

Anamika is a QA Consultant with over 13 years of experience in software testing and automation and leads Nagarro's Emerging and Special Tech Practice. Passionate about inclusivity and equal access to technology, she explores AI-driven innovations to advance testing practices. Her focus is on delivering sustainable, accessible, and high-quality solutions that empower diverse users across an evolving digital landscape.

**Anamika Mukhopadhyay**

Deepshikha leads Nagarro's AI in Testing practice and brings over 13 years of experience at the intersection of software quality and emerging technologies. She specializes in designing AI-driven testing strategies that enhance reliability, accelerate delivery, and help organizations build systems that are not only efficient but also trustworthy and resilient when it matters most.

**Deepshikha**