When Data Becomes Context

Why AI Systems Fail Even When the Data Is Correct

The Assumption AI Breaks

For two decades, enterprises built data infrastructure around a stable assumption: data would be prepared upstream, organized once, and consumed repeatedly by humans or deterministic applications. Data warehouses stored it. Catalogs tagged it. Pipelines moved it. Quality teams monitored it. Security teams gated access to it.

The assumption held because consumers were predictable. A BI dashboard queried the same tables the same way. An application retrieved records by known keys. A human read a document from beginning to end. The relationship between data and its use was stable enough that preparing data in advance made sense.

Artificial intelligence breaks this assumption.

Modern AI systems do not retrieve data. They assemble context. Large language models, retrieval systems, and agents combine fragments from documents, databases, emails, tickets, logs, and APIs at the moment a question is asked. The output is not a static artifact, it is a transient construction that exists only long enough to influence a decision.

> **Example: The Research Report**
>
> A pharmaceutical researcher asks an AI assistant about dosing protocols. The AI retrieves fragments from a 2024 clinical study, a 2022 safety review, internal meeting notes from three different teams, and an email thread discussing a protocol amendment. Each fragment is accurate. Each is authorized. But the 2022 safety review references a protocol version that was superseded. The AI has no way to know this, it sees five relevant fragments, not a coherent timeline.

This is not a failure of data quality. The data is correct. It is not a failure of access control. The researcher is authorized to see all of it. It is a failure of data use, the context required to interpret these fragments correctly does not exist at the moment the AI assembles them.

What Data Use Failure Looks Like

A data use failure occurs when content that has been copied, overshared, duplicated, and spread across systems is used by AI without the context required to interpret it correctly at decision time.

The underlying bytes may be accurate. Authorization may be valid at every source. The failure arises because the same content exists in many copies, while the context that explains how to use it, which version is current, which is authoritative, which applies to this situation, is scattered across different systems and never attached to the content itself.

Example: The Contradictory Answer

A financial analyst asks an AI assistant for the Q3 revenue forecast. The AI retrieves fragments from the board presentation (showing $42M), the sales team's pipeline spreadsheet (showing $38M), and an email from the CFO discussing a revised estimate ($40M). It returns an answer citing all three figures with varying confidence. The analyst has no way to know which is authoritative. Neither does the AI.

Example: The Stale Instruction

A manufacturing engineer asks about the curing temperature for a polymer compound. The AI retrieves the specification from a product datasheet. The answer is correct, for the formulation that was replaced six months ago. The updated specification exists in a different system. The AI found the most semantically similar fragment, not the most current one.

Example: The Confidential Leak

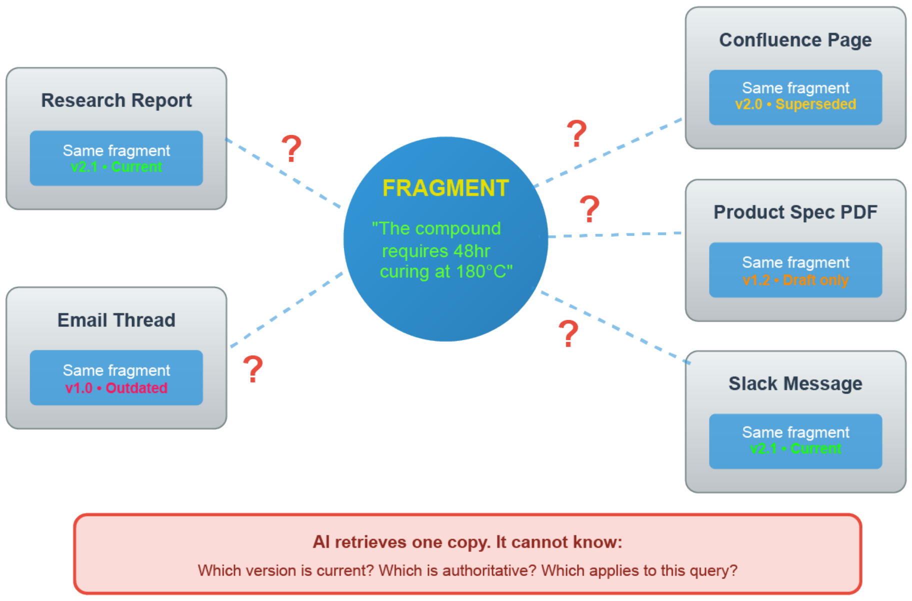
A junior analyst asks about competitive positioning. The AI retrieves fragments from a market analysis, including two sentences from an M&A due diligence memo that was copied into the analysis document before the deal was announced. The analyst is not authorized for M&A information. But the fragment exists in a document they can access. The AI has no way to know those sentences carry different restrictions than the surrounding text.

In each case, the data is present. The data is allowed. But the context required to use it correctly is missing at the moment the AI makes a decision.

Why Fragments Lose Their Context

AI systems do not consume documents. They consume fragments, sentences, paragraphs, table rows, chart data points. A fragment is an orthogonal building block: a discrete unit of content that can be matched anywhere it appears, regardless of what larger structure it's embedded in.

Fragment Sprawl: Same Content, Lost Context



The same fragment might exist in a research report, a Slack message, an API response, a presentation deck, and an email attachment. It's the same content in each case. But each copy carries different implicit context: different authorship, different timestamps, different surrounding content that explains its meaning.

When a fragment is extracted from its source, by a copy-paste, an export, an API call, or a RAG chunking pipeline, that implicit context is severed. The fragment becomes orphaned content. It can still be found. It can still be retrieved. But the information needed to interpret it correctly no longer travels with it.

Example: The 95% Duplication Problem

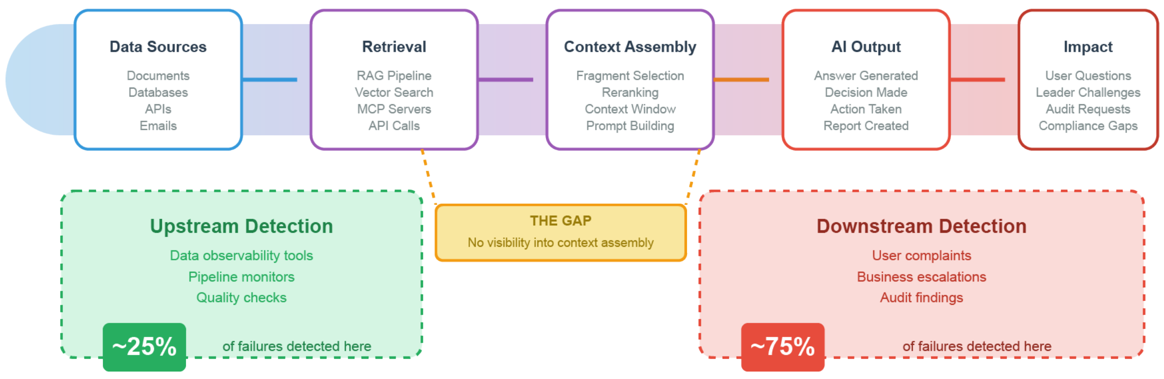
An enterprise content audit reveals that a single paragraph describing a product's regulatory status appears in 847 documents across SharePoint, Confluence, Google Drive, and email attachments. 12 of those documents contain an outdated version from before a label change. 3 contain a draft that was never approved. The remaining 832 are identical, but spread across systems with no linkage between them. When AI retrieves this paragraph, it has no way to know which of the 847 copies it found, or whether that copy is current.

Over 95% of enterprise content fragments exist in multiple copies. The fragment is not the problem. The missing context is the problem.

How These Failures Are Detected

Data use failures are detected in two ways, and the ratio between them reveals the depth of the

Where Data Use Failures Are Detected



Tools see data at rest and in pipelines. They don't see how data is used at runtime.

problem.

Upstream detection happens through data observability. Quality and platform teams operate tools that monitor freshness, schema stability, volume, and distribution. These tools detect anomalies in pipelines and transformations. But they were designed for structured data flowing through known paths. They do not inspect AI context assembly. They do not see which fragments were retrieved or how they were combined.

Downstream detection happens through bad outcomes. A user questions an answer. A business leader challenges a decision. A regulator asks how a conclusion was reached. Only then does investigation begin, often with observability tools enlisted to help reconstruct what happened, rather than having detected the issue directly.

> *In enterprise deployments, roughly three-quarters of data use failures are surfaced downstream, by AI behavior or business impact, rather than upstream by observability. The tools see the data. They don't see the use.*

Why Current Approaches Cannot Solve This

Engineering Workarounds

When data use failures surface, responsibility defaults to engineering. This is not because engineering caused the problem, it's because engineering has historically been able to fix data problems through code.

In theory, engineering can influence AI behavior. Rerankers can adjust relevance. Prompts can be tuned. Agents can be instructed to prefer certain sources. But these techniques cannot address the root problem.

> **Example: The Reranker That Can't See Freshness**
>
> An engineering team implements a reranker to prioritize more relevant results. The reranker scores fragments by semantic similarity to the query. But semantic similarity has no relationship to freshness, a two-year-old document may be more semantically similar than a current one. The reranker optimizes for the wrong signal because the right signal (which version is current) isn't available to it.

Engineering teams lack a unified view of content across systems. They cannot connect the same fragment across its many copies. They cannot determine which version is authoritative at runtime. They cannot maintain this logic as content is copied, transformed, and spread across the enterprise.

Data Quality and Observability Tools

Traditional data quality tools were designed for structured data in pipelines. They monitor schemas, freshness, volumes, and distributions. Newer tools extend detection into semi-structured content. But all of them share a fundamental limitation: they observe data at rest or in transit through known paths. They do not control how content is used at runtime.

> **Example: The Freshness Monitor That Misses the Problem**
>
> A data quality tool monitors the 'last_updated' timestamp on a regulatory database. The database is fresh, updated daily. But the AI retrieved a fragment from a PDF export of that database created eight months ago and stored in a shared drive. The quality tool sees a healthy database. It doesn't see that the AI used a stale copy.

These tools observe the problem. They do not control it.

Security and Access Control

Security tools govern who may access which resources. DLP tools detect sensitive patterns and block or redact them. AI guardrails filter prompts and outputs. But none of these operate at the level of fragment context.

Example: The DLP That Creates Hallucinations

A DLP tool detects a Social Security Number pattern in a retrieved fragment and redacts it. The fragment was a customer service note explaining why an account was flagged. With the SSN redacted, the AI loses the context that explained the flag. It infers a reason based on other fragments, and infers incorrectly. The security tool prevented a leak but caused a hallucination.

Security tools today are not designed for data use governance. But this is changing. The EU AI Act and emerging regulations are establishing requirements for controlling how data influences AI decisions, requirements that current security tools cannot meet. Data use governance is evolving from a quality problem to a compliance problem. Security teams will own enforcement tomorrow, but they lack the tools to do so today.

The Gap: No Infrastructure for Fragment-Level Context

The problem is not organizational ambiguity. It is not a lack of tools. It is a specific technical gap: no infrastructure exists to identify, link, and reconcile data fragments across their many copies in a way that AI can use at runtime.

This infrastructure would need to:

- Recognize the same fragment anywhere it appears, regardless of container
- Link copies of that fragment across systems, formats, and time
- Attach context that explains currency, authority, and appropriate use
- Make that context available at the moment AI assembles its response
- Operate continuously as content changes, spreads, and evolves

Building this capability internally would require years of effort, deep specialization, and ongoing operational investment that most enterprises do not possess and should not attempt.

This is not a problem that can be solved with better prompts, smarter rerankers, or additional observability. It requires infrastructure that does not exist in the enterprise today.

The Stakes: AI That Cannot Scale

The *ISG State of Enterprise AI Adoption* report quantifies the impact: in 2025, only 31% of AI use cases reached full production despite widespread experimentation. ISG identifies data readiness as a foundational requirement for achieving value with AI.

As AI use cases move closer to revenue-generating activities, data use failures have direct business impact. Inconsistent answers erode trust. Contradictory outputs create confusion. Unexplainable decisions create liability. Each failure makes stakeholders more reluctant to rely on AI for consequential work.

The result is a ceiling on AI adoption. Enterprises can experiment freely but cannot scale reliably. Projects that work in pilots fail in production. Use cases that should deliver ROI instead deliver risk.

What the Solution Must Do

The executive accountable for AI outcomes, typically the head of AI platform, chief data officer, or enterprise AI governance lead, needs infrastructure that solves the problem engineering cannot.

That infrastructure must:

Identify fragments: Recognize the same content anywhere it appears, across all copies and containers

Track fragments: Follow content as it moves through APIs, agents, and context assembly

Attach context: Link each fragment to the information needed to interpret it correctly

Control use: Ensure AI receives the freshest, most relevant, most appropriate content for each decision

This capability is called AI Data Control. It operates at the level of data fragments rather than datasets or documents. It enables runtime governance of data use, not just data access, not just data quality, but the actual contribution of data to AI-generated outcomes.

The companion paper *AI Data Control: The Solution Architecture* describes how this capability is built and how Caber delivers it.