O'REILLY®
Report

Ensuring Data + AI Reliability Through Observability

Aligning People, Processes, and Technology to Drive Value

Barr Moses &
Michael Segner

MONTE CARLO

End-to-end trust for your entire data + AI system

Reduce risk, improve outcomes, and promote trust across the entire enterprise.

Learn more:
www.montecarlodata.com/why-monte-carlo

Ensuring Data + AI Reliability Through Observability

Aligning People, Processes, and Technology to Drive Value

Barr Moses and Michael Segner

O'REILLY®

Ensuring Data + AI Reliability Through Observability
by Barr Moses and Michael Segner

Published by O'Reilly Media, Inc., 141 Stony Circle, Suite 195, Santa Rosa, CA 95401.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*https://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Aaron Black
Development Editor: Jill Leonard
Production Editor: Gregory Hyman
Copyeditor: Penelope Perkins
Cover Designer: Susan Thompson
Cover Illustrator: Ellie Volckhausen
Interior Designer: David Futato
Interior Illustrator: Kate Dullea

September 2025: First Edition

Revision History for the First Edition
2025-09-15: First Release

979-8-341-64942-2

[LSI]

Table of Contents

CHAPTER 1
AI Is Easy. Data + AI Is Hard.

Let's just say it out loud. Generative AI, or GenAI, will make or break your company, and it will make or break your career. It's thrilling, and if we are being honest with ourselves, terrifying.

Every member of the boardroom and the breakroom has seen how past platform shifts have crowned and dethroned industry titans.

> We are completely convinced the consequences will be extraordinary and possibly as transformational as some of the major technological inventions of the past several hundred years.
>
> —Jamie Dimon, CEO, JPMorgan Chase, 2023 Annual Shareholder Report

Put simply, data is the fuel that will turn this AI spark into a bonfire. Tomorrow's winners will be the organizations that dedicate themselves to delivering reliable data + AI systems. And the only way not to have this flame sputter out is by aligning people, processes, and technologies around reliability and observability best practices today.

In this report, we will provide best practices on how to achieve this based on learnings from the past as well as hundreds of interviews with leading data + AI teams. We will detail the challenges they face and how they are ensuring these complex systems meet enterprise reliability standards.

But first we need to understand exactly what we mean by a *data + AI system*, why it's a game changer, and what makes it so hard to execute reliably at scale.

Clueless Without Context

GenAI has already made an extraordinary impact on enterprise productivity. Marc Benioff has stated that Salesforce will keep its software engineering head count flat due to a 30% increase in productivity thanks to AI. Users leveraging Microsoft Copilot create or edit 10% more documents.

And this impact has been broadly distributed. Powerful AI models are widely available and just a simple API call away (as Meta and OpenAI ads are sure to remind us).

These open foundational models are great at accelerating common, predictable tasks like writing a thank-you email. The challenge is automating or accelerating tasks that require a business-specific context.

Imagine an airline seeking to leverage AI for common customer interactions (Figure 1-1). Unless the model is provided data on company policies and given access to informational systems, ChatGPT can handle this about as well as an intern on their first day.

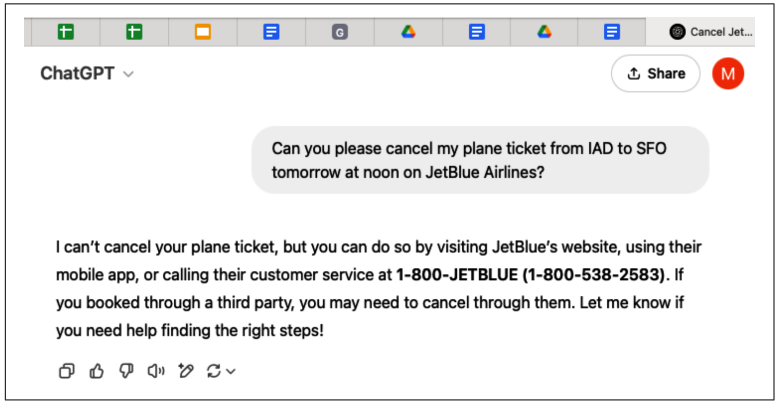

Figure 1-1. ChatGPT can't cancel my flight when only given the above query because it has no data on who I am or my interactions with the airline

And we have yet to mention that when AI doesn't have context from data, it has a tendency to make it up. No one wants their picture next to AI hallucination headlines like these:

- "A Chevy for $1? Car Dealer Chatbots Show Perils of AI for Customer Service"
- "AI 'Hallucinations' in Court Papers Spell Trouble for Lawyers"
- "The New AI-Powered Bing Is Threatening Users. That's No Laughing Matter"

Studies have shown that grounding large language models (LLMs) in certified, reliable data can reduce hallucination rates by nearly 30%.

Powerfully Perceptive

The real disruption lies with *data + AI*: when organizations combine their first-party data with LLMs to inform chatbots or create AI agents (larger AI systems that complete tasks for users).

This mainly takes place using a retrieval-augmented generation (RAG) or other context engineering workflow. You can think of this process almost like passing a note to the AI model to help it cheat on its test.

For example, if a user asked a customer support AI agent if their order could be refunded, the final prompt may look something like this:

```
You are a helpful, courteous customer service agent. Respond to
the user query: "Can I get a refund on my last order?" Reference
the context below:
- Customer: Jane Doe
- Order ID: #45678
- Purchase date: August 3, 2025
- Product: Noise-canceling headphones
- Refund policy: Returns on electronics are accepted within 30
days of purchase if the item is unopened or defective.
- Order status: Delivered August 5, 2025
- Customer note: "The left speaker isn't working."
```

This context doesn't appear by magic. It requires the AI agent to be able to access relevant structured data (order ID, purchase date, etc.) as well as unstructured data (refund policy). It's also the end result of many interdependent, complicated processes that we will describe in further detail in Chapter 3. These inputs need to be monitored to ensure enterprise-grade reliability. Is the data accurate, available, relevant, and successfully retrieved by AI?

And let's not forget the job isn't done once the context has been successfully injected! The AI output also needs to be monitored to ensure enterprise-grade reliability.

The model response may not fully reference the provided context. For example, the model may not make the connection that noise-canceling headphones are electronics. Or it may respond with the wrong tone. So, even with the context from our preceding example, you may still get an output like:

```
"Sorry, we don't offer refunds on headphone purchases."
```

Let's flesh this out further with two real-world use cases.

Credit Karma

Credit Karma is using GenAI to automate some customer interactions and provide personalized offers via a chatbot. This initiative has been successful because AI engagement is informed by key customer context and history.

Monitoring for data quality issues with the inputs and outputs of Credit Karma's credit risk models is a critical step and serves as the foundation of these personalized engagements.

"If the LLM doesn't know my income or my credit score, it's not going to come up with the right answers given my situation. And if I'm looking to buy a car to replace my 10-year-old Honda, once I buy that car, it's done," said Vishnu Ram, VP of engineering at Credit Karma. "We want to make sure the LLM is not thinking about me replacing that car forever."

Monte Carlo

Monte Carlo (our company) offers a data + AI observability platform that detects data + AI reliability issues and accelerates root cause analysis so that issues can be resolved before they impact business operations and stakeholder trust.

The AI-based Troubleshooting Agent leverages hundreds of sub-agents working together to review all the telemetry and signals that Monte Carlo collects across the data delivery process to automatically identify the root cause of a data quality incident. An abstracted view of the architecture is shown in Figure 1-2.

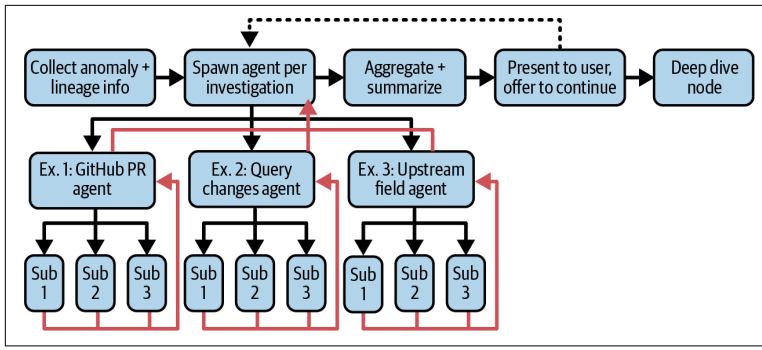

Figure 1-2. Monte Carlo Troubleshooting Agent's architecture leverages hundreds of subagents, each designed to investigate and validate a possible root cause

As we can see, when data is added to the mix, AI becomes exponentially more powerful. Gartner predicts that "by 2028, 33% of enterprise software applications will include agentic AI, up from less than 1% in 2024, enabling 15% of day-to-day work decisions to be made autonomously."

In other words, the difference between AI and data + AI may be the difference between becoming the next Netflix or the next Blockbuster.

Data + AI: An Emerging System of Systems

Data + AI systems are actually three separate stacks coming together: structured data, unstructured data, and AI. The data + AI infrastructure market map by Bessemer Venture Partners has more than three hundred vendors listed across more than two dozen categories. Each of these stacks has many tightly coupled components and fragile dependencies that can introduce reliability issues.

Structured data pipelines, which handle data in a tabular format with a defined schema, ingest data from multiple data sources and refine it through multiple layers of transformations. The reliability of each step depends on the successful execution of each process upstream.

Unstructured data pipelines, which handle data without a predefined schema/organization, also have a multilayered extract, transform, and load (ETL) process. This involves extracting meaning from a file, creating smaller chunks to fit within the model's context

window (short-term memory), and embedding those chunks as vectors so they can be retrieved by AI.

Each of these steps has breakpoints that can create cascading issues as well. When you have issues with the reliability of AI inputs, you have issues with the output. As the well-known data science cliché goes, "garbage in, garbage out."

> The definition of success is not just that [retrieval] was executed without an error, but that also you really successfully extracted all the information that you want to extract…and that's represented in a meaningful way within the vector database so you can retrieve it the way you'd like to.
>
> —VP of data engineering and MLOps, financial software company, March 2025

As we alluded to previously, making sure your structured and unstructured data traverses these pipelines reliably is necessary, but not sufficient, for maintaining trusted data + AI products. AI agents also need to be able to retrieve this context and work together effectively—which, of course, involves more tightly coupled components and fragile dependencies.

Most agentic systems will leverage multiple models and subagents because some models are better-suited or more cost-effective for certain tasks. These agents need to be orchestrated and given instruction on how to perform their tasks. In the software world, this is done through code and APIs. Those components are leveraged in the AI stack as well, but natural language prompts, orchestrators (e.g., LangChain), and model plug-ins (e.g., OpenAI SDK) are more common. Model Context Protocol is also gaining popularity as a standardized means for transferring context between models and tools.

Changes to the prompt, model, or orchestration logic can all have dramatic impacts on the final output.

> Now we're asking software engineers to try and figure out, what does it mean to interact with a mysterious black box? Changing models or even just subtle changes to the prompts…can have strange side effects; it's [difficult] to establish the ground truth.
>
> —Director of engineering, travel technology company, March 2025

As we'll explore in more depth in Chapter 3, these core components across all three systems can be organized into a *data, system, code, model* framework (Table 1-1). These four components refer to the four root causes of reliability issues or main breakpoints for data + AI systems. Visibility across them is vital for enterprise-grade reliability.

Table 1-1. The data, system, code, model data + AI observability framework

	Structured	Unstructured	AI
Data sources	Transactional databases External/internal sources	Files Knowledge repositories	RAG/retrieved context
System	Warehouse/lakehouse Pipeline orchestrator Transformation models Batch/streaming ingestion pipelines	Extractors/parsers Cloud object storage Warehouse/lakehouse Vector databases Pipeline orchestrators	Model APIs LLM orchestrators Guardrails
Code	Query code (SQL)	Query code (SQL) Spark/Python	Prompts Agents
Model	Ingested model outputs	Embedding models	Foundational models

But before we go deep on pipelines, it's critical to understand the key reliability best practices that have emerged over time.

CHAPTER 2

Reliability Lessons from DevOps and DataOps

As we outlined in Chapter 1, delivering reliable data + AI applications is an incredibly challenging proposition. The good news is that there are multiple examples and even playbooks for how teams have solved similar problems in the past. The key to delivering reliable data is having a robust process in place and trusting your people to deliver.

Reliability Issues Are Inevitable

Let's start by saying that achieving 100% reliability is not the goal. Just like any credible cybersecurity professional would never look you in the eye and say, "Our defense is perfect, we will never be breached," no engineer should ever claim to have architected the perfect system.

That doesn't mean organizations shouldn't invest in preventative measures—an ounce of prevention is worth a pound of cure, as the saying goes. However, it's important to build systems, processes, and teams with the understanding that reliability issues are inevitable.

In other words, anticipate breakage that can't be anticipated. Know there will be "unknown unknowns" and, in the wise words of our Site Reliability Engineering forebears, "embrace risk."

> You might expect Google to try to build 100% reliable services—ones that never fail. It turns out that past a certain point, however, increasing reliability is worse for a service (and its users) rather than better!
>
> Extreme reliability comes at a cost: maximizing stability limits how fast new features can be developed and how quickly products can be delivered to users, and dramatically increases their cost, which in turn reduces the numbers of features a team can afford to offer.
>
> —Chapter 3, "Embracing Risk," in *Site Reliability Engineering* (O'Reilly)

Organizational Complexity Is Inevitable

At most large organizations, data + AI teams are just as disjointed and siloed as the complex systems they are charged with building and maintaining. And, just like the reliability issues we referenced at the start of this chapter, much of this complexity is inevitable.

As amazing as it would be to hire professionals that could oversee the data + AI product delivery process from "farm to table," few individuals can excel equally at translating business needs, engineering robust systems, and staying at the cutting edge of AI innovation. There's a reason Henry Ford pioneered the assembly line and microservices have come to replace monolithic applications: specialization enables consistency, speed, and scale.

While organizations will vary across the centralized/decentralized spectrum, most mature data + AI operations converge on a hub-and-spoke model: a centralized function that supports embedded, specialized data + AI teams within each business domain (Figure 2-1).

This is why data + AI reliability is difficult. Systems and teams, both highly interdependent and complicated, must function autonomously while collaborating closely. The next section will dive into exactly how to do just that.

> An observability product without an operational process to back it is like having a phone line for 911 without any operators to receive the calls.
>
> —Director of data engineering, JetBlue, January 2024

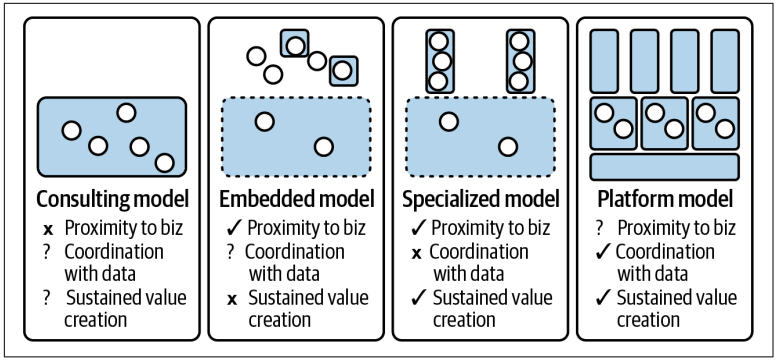

Figure 2-1. Different ways to organize data + AI teams; most teams at scale will employ a hub-and-spoke specialized model or platform model (Source: adapted from an image by Shane Murray)

Workflows and Best Practices

Gartner's *Market Guide for Data Observability Tools* lists three critical workflows: monitor and detect, alert and triage, and investigate. We leverage a similar framework that we refer to as detect, triage, resolve, and measure.

Regardless of what you call them or the technology deployed, these processes are essential to maintaining reliable systems. Let's dive into best practices and considerations for each workflow.

Detect

The first principle is straightforward in concept, but difficult in execution: end-to-end monitoring across the entire system. Reliability issues can creep in at any point of the data + AI lifecycle, and so monitoring must be end-to-end as well.

This means specialized teams with varying levels of technical competency must be able to deploy their own monitors. This is because such teams are closest to the underlying systems, data, and AI and have the deepest understanding of what needs to be monitored and how. It's also the only way to reasonably scale end-to-end.

Of course, "end-to-end" does not mean "boil the ocean." DevOps principles point to "only as much reliability as needed," and the concept of data products also suggests governing datasets at the appropriate level based on the business operation supported.

Triage

If DevOps and DataOps teams have taught us anything, it is this: *alert fatigue is real* (Figure 2-2). When more than five notifications are sent to a channel per day, the average response rate drops to less than 30%.

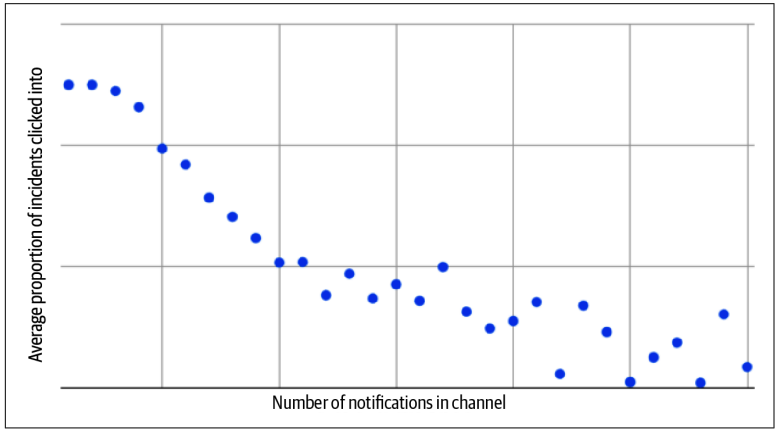

Figure 2-2. Notification interaction versus notification volume in a channel (Source: Monte Carlo telemetry)

There are many reasons why monitoring platforms have a reputation for sending so many alerts:

- It's difficult for humans to set alert conditions that are neither too noisy nor too permissive.
- The scope is excessive, as end-to-end becomes boil the ocean.
- Teams stop responding or adjusting to alerts, which creates… more alerts.

Some of these can be resolved with better technology. For example, many, but not all, alert conditions and thresholds can be automated with AI agents and ML-powered anomaly detection, respectively.

However, the best way to tackle this challenge is to have coherent and cohesive monitoring and notification strategies—in other words, sending the right alert to the right team at the right time. If teams can't immediately see whether an alert is both important and relevant, alert fatigue is just a matter of time. This requires end-to-end data + AI lineage (a map of how data and systems

connect). Otherwise, teams can't understand what's impacted downstream (importance) or the point of origin upstream (relevance).

Point of origin is particularly important, as the owner of the most upstream asset impacted will typically be the owner of the incident; if this information is obscured, then one root cause could trigger two hundred alerts that launch overlapping fire drills across 20 teams.

The final ingredient of this triage cake is accountability. If teams are receiving important and relevant alerts, there needs to be a culture of engagement. Invariably, the teams with the most reliable systems are not only those with solid monitoring and notification strategies, but those with leadership involvement.

> We see asking questions on every alert as an integral part of our team's culture. For every incident, we are going to fix the problem or fix the alert.
>
> —Senior ops engineer, Seasoned, January 2024

Resolve

The major paradigm shift introduced by DevOps and application observability solutions is that instead of *only* monitoring outcomes at each stage in the process, teams need to have visibility into all the major components that influence those outcomes.

To use an analogy, imagine you have a machine that produces widgets. Instead of just picking up and inspecting the widget as it's being formed at each stage of the assembly line, you need visibility into the machines building that widget. Otherwise, you might end up detecting that the widget is the wrong color, but without any idea *why*. And without any idea why, you can't *fix it* quickly and reliably.

We can see this shift in the software engineering world. Instead of only testing each piece of code, software engineers now rely on observability into the logs, traces, and metrics of their systems.

We can see this shift in the data engineering world, too. Instead of only testing each field of data, data engineers are increasingly relying on data + AI observability solutions—or *data observability*, a term coined by coauthor Barr in 2019—to understand how bad data is introduced via data sources, ETL systems, query code changes, or AI model outputs (Figure 2-3).

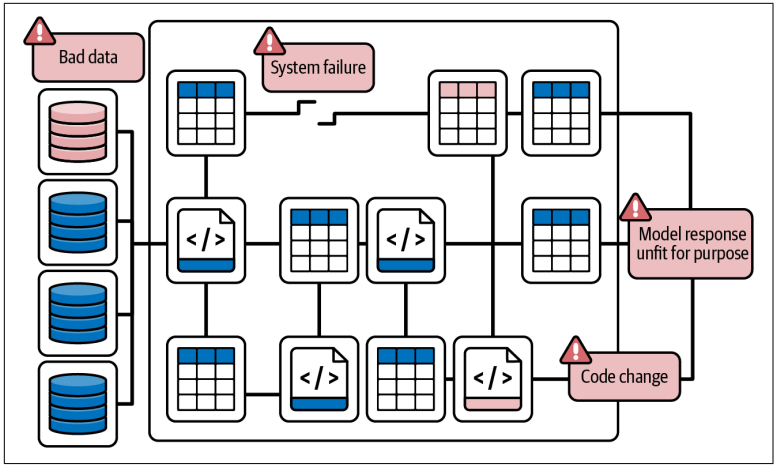

Figure 2-3. The four root causes of data quality issues

Measure

The final workflow that teams need to master to improve data + AI system reliability is the favorite of governance managers everywhere: measuring key reliability metrics.

DevOps teams aced this one long ago by focusing on the system availability service level agreement (SLA). The focus is so intense that not only is it measured to three decimal places, but it has made its way into virtually every contract of every major cloud infrastructure provider. You can expect 99.999% uptime.

What makes the system availability SLA such a great North Star metric is that it can be:

- Understood and valued by end users (requests failed/total requests made)
- Standardized across systems
- A leading indicator of future, more consequential, incidents

Data reliability is a little more tricky:

- Inputs are constantly changing.
- The surface area is exponentially larger than application infrastructure and constantly evolving (often subtly).

- The expected result is harder to define than that of a software application, and typically can be defined only by a few experts.
- Data quality resists standardization. It is often measured across six dimensions, the requirements of which vary from use case to use case. If a dataset has a 60% yellow light for completeness, and an 80% green light for uniqueness, what does that communicate regarding its reliability for an end user?

As a result, DataOps teams rely heavily on a combination of how outputs meet defined expectations as well as how closely they fall within historic norms. Measuring downtime (the amount of time that data is wrong or inaccurate) is critical within this discipline as well.

Data + AI systems are even trickier still:

- AI models are not deterministic.
- Standards for unstructured data accuracy are murky.
- Consumption and outputs vary from chatbots to AI agents.

When the accuracy/truth of inputs and the expectation of outputs cannot be strictly defined, creating reliability metrics is extraordinarily difficult. More on this next.

CHAPTER 3

The Pillars of Data + AI Observability

In this chapter, we'll reference common technologies and architectures to provide actionable insights for ensuring reliable data + AI products. These technologies will change over time. However, the larger pillars—specifically, the types of anomalies and their root causes—will be much more enduring. This is because they are tied to the data + AI lifecycle. To quote an O'Reilly classic, *Fundamentals of Data Engineering* by Joe Reis and Matt Housley, "since the dawn of data, we've seen the rise and fall of innumerable specific technologies and vendor products, but the data engineering lifecycle stages have remained essentially unchanged."

Monitoring for anomalies involves understanding which attributes, or metrics, of an output to measure and then identifying outliers from the traditional behavior. Root cause analysis involves reviewing telemetry across the production process to determine what changes or errors occurred that were most likely to have caused that anomaly.

The key is to clearly define the metrics to measure and the telemetry to collect at each stage of the lifecycle across your unstructured data, structured data, and AI systems. Let's take a look.

Unstructured Data

When AI reaches to retrieve data for reference, it needs to be available, discoverable, up-to-date, nonbiased, and useful. This corresponds well with many of the traditional dimensions of data quality, including completeness, freshness, uniqueness, and validity/consistency.

Ironically, one of the primary challenges to accomplishing this level of monitoring for unstructured data is the lack of structure. And, as has been the predominant trend in unstructured data management for the last five years as we've built houses on our lakes, the best practice for solving this quality challenge is to add structure.

Let's back up for a moment. More than 80% of an organization's data lives outside of columnar tables as unstructured text, video, or audio. It's littered across tens of millions of files across tens of thousands of source systems throughout the enterprise. Monitoring at each source is costly, inefficient, and impractical.

Data + AI teams will often consolidate key unstructured data files within cloud object storage (or sometimes directly in the warehouse or lakehouse) before breaking up the text into smaller chunks that are then transformed into vector embeddings that can be retrieved and referenced by AI.

Many of these processes revolve around semistructured JSON files, vector databases, or as chunk and embedding tables in the warehouse/lakehouse. Different teams will use different architectures based on their use case and level of maturity.

Leveraging purpose-built vector databases is often the simplest and most cost-effective route, particularly when supporting a single use case. In our conversations, we've found smaller organizations more likely to use vector databases such as Pinecone, Qdrant, and PostgreSQL, while larger enterprises will more commonly use Azure AI Search and MongoDB.

An increasingly popular trend is for data + AI teams to create multi-use embeddings, or embeddings that can support multiple use cases. An analogy would be the shift from ETL to ELT in structured data engineering. ETL pipelines were built to ingest and transform data for a single use case (often a report or dashboard). Today's more modern ELT (extract, load, transform) pipelines gather data from

a variety of sources and create a standard set of metrics that embedded analysts and others can self-serve as business needs evolve.

As with the ETL to ELT shift, multiuse embedding pipelines are more complex, but ultimately more modular, scalable, and agile. Data + AI teams at this level of maturity will often use a warehouse or lakehouse as their embedding store and central source of truth. This enables and necessitates stronger governance.

As one data scientist at a large enterprise explained to us, "We use a [data warehouse] for internal serving convenience. We are trying to see how we can maximize the use of our embeddings so we don't have teams trying to rebuild and duplicate work, even if it's a bit more expensive to store them in the warehouse. Everyone has access."

In these warehouse- or lakehouse-based architectures, inventory tables often serve as a table of contents of all files in cloud storage and are leveraged for the curation process. Chunk and embedding tables are used to capture associated metadata to aid retrieval and provide visibility into what data is encoded into specific embeddings (since vectors are unintelligible to humans).

For a use case focused on government contract award data, an embedding table may look something like Table 3-1.

Table 3-1. Example embedding table

Field name	Field value
Chunk_id	chunk_123
Chunk_text	NASA awards SpaceX $1.5 billion for...
Chunk_overlap	150
Chunk_created_at	2025-06-1012:32:00
File_id	file_123
File_type	PDF
Object_path	s3://pdfs/contracts/file_123
Object_last_modified	2025-07-0214:56:03
Embedding_model	OpenAI-ada-002
Embedding_vector	[0.12, -.34, 3.53]
Embedding_created_at	2025-07-108:45:03
Language_detected	en
Award_recipient	SpaceX
Award_amount	1,500,000,000

An architecture leveraging structured chunk and embedding tables may look like Figure 3-1.

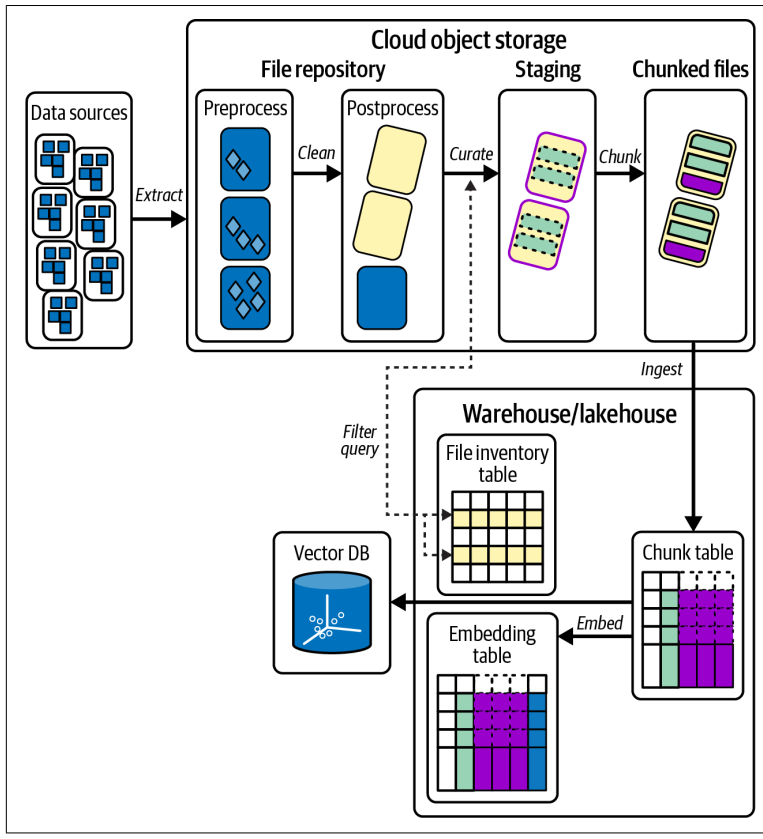

Figure 3-1. Unstructured data pipeline architecture

Monitoring Strategy

In this section, we'll discuss some monitors that can be deployed at the cloud object storage and vector database levels. The key focus, however, will be on data quality monitoring across chunk and embedding tables in the warehouse and lakehouse, as they are the most centralized and structured assets across the pipeline.

> I think all the classic problems we have with our structured data also apply to our unstructured data.
>
> —Systems architect data and analytics, logistics company, June 2025

Completeness

If files or key components within files are missing, the result will be incomplete chunks or embeddings, which, in turn, will cause AI to be less grounded and more prone to hallucinate.

The challenge, of course, is scale. As one data + AI team told us, "We have [an AI] use case we are launching today with 75 documents [as the reference corpus], so if one is missing we will probably notice, but the next use case could be 20,000 documents."

Missing files can be best detected with anomaly detection on the row count of the inventory table indicating fewer files are being collected, or there is a problem with the inventory process.

Missing key components within a file can be detected by monitoring for anomalies in file size (bytes) within object storage. As a best practice, consider applying metadata tags by dataset and deploy monitors when tagged data is modified or deleted. You can also use protective measures such as resource locks to prevent accidental data loss.

Once the file has been parsed, incomplete data can be detected by monitoring for a spike in NULL values, drop in row count, or string size anomalies within the main text field of the chunk table, particularly if the chunks have semantic meaning (for example, chunking a news article by author, headline, and article copy).

After the chunks have been encoded into vectors, incomplete data can be detected within the embedding table by monitoring for a drop in row count, NULL vector fields, empty arrays (arrays with a length of zero), or vectors with a sum of zero.

Freshness/Timeliness

If data is not updated when it should be, AI will reference outdated materials that may be incorrect. This issue can be detected by monitoring file update timestamps within object-level storage as well as the update timestamps within the chunk or embedding tables to identify any unusual gaps. This helps determine if the underlying issue may be from the source or downstream processes.

There may be unstructured data with irregular but critical updates. For example, a competitive intelligence document may be updated whenever a competitor makes a major announcement, or a mechanic's guide may be updated to support a new car model. In these

cases, teams may deploy monitors to ensure the last modified date of the file in object storage is before the chunk and/or vector creation date in the respective tables within the warehouse or lakehouse. However, this check is predicated on a smart unstructured data pipeline that extracts and tracks this metadata.

Uniqueness

Duplicate data can imbalance a model response and create bias, not to mention generate excessive costs throughout the pipeline.

A data + AI team at an international news company ensures it is not duplicating data within its chunks or embeddings by setting alerts for duplicate values on the uniform resource identifier field that identifies and points to each article in its respective table. This way, each article is chunked and embedded only once.

Another emerging approach involves leveraging new LLM SQL functions offered by data warehouses and lakehouses to compute vector similarity for embeddings. Duplicate or near duplicate embeddings—perhaps one chunk is "Michael Segner" and the other "Michael R. Segner"—can also be detected by alerting to cosine similarities of, say, .95 or higher.

Cosine similarity monitors need to be deployed with caution, as they can be costly if leveraged across large datasets. A smart approach is to check for changes in patterns over time by grouping vectors into daily (or periodic) batches and then comparing the overall "shape" and average of each batch to the previous batches (Figure 3-2).

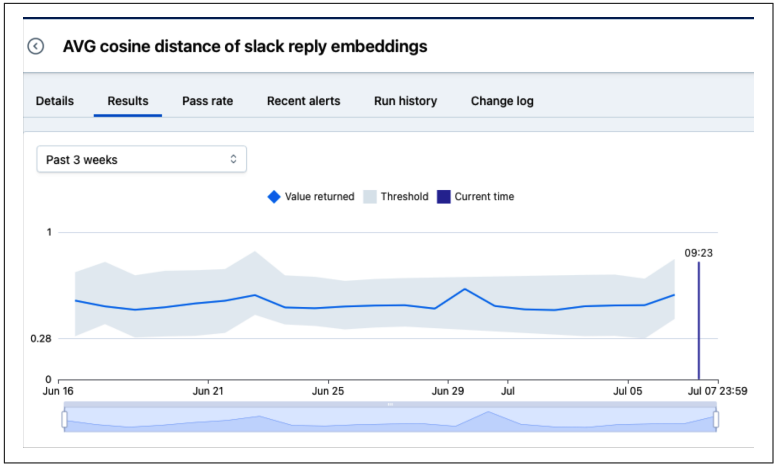

Figure 3-2. A monitor detecting semantic similarity between a group of embeddings

Validity/Integrity

Just like structured data, if unstructured data doesn't conform to expected formats and specifications, it can not only impact an AI model's output, but also create failures in downstream processes such as chunk sizing, vector dimensions, and encoding errors.

A chunk that contains too few characters may indicate an anomaly in the chunking process, and that chunk runs the risk of not carrying any semantic meaning. Conversely, a chunk that is too large will exceed the model's context window and fail to encode as a vector. Monitoring for anomalies on string length for the text field within the chunk table can be an effective validity monitor.

As one data + AI leader put it, "[Monitoring for chunk length would be good] because right now…our chunkers are pretty good, but we also run into hard situations where the length of our [documents] can be highly variable…so making sure the average chunk length isn't five one day and the next day a thousand [would be helpful]."

Many vector databases require consistent vector dimensions to function properly, and changes in the length of vectors can indicate a change in the embedding model, which may impact AI performance downstream. This validity check can be executed by monitoring for the expected array length within the embedding table.

For example, an embedding of `[.8, .65, .43]` would have three dimensions and be an array length of three.

One data + AI leader relayed painful learnings in this area, telling us, "We had a [team] parse, chunk, and embed and when they went to push all of this into the vector database, the dimensions didn't match and the retrieval was horrible. They only realized this during retrieval after all the indexing and costs had been accrued."

Extracting text from PDFs and images, often done by optical character recognition (OCR), can be fraught with errors. Helpful monitor types can include those alerting to abnormal whitespace, non UTF-8 characters, or high rates of non-dictionary words. One data + AI leader in the health care space told us, "We deal with a lot of clinical PDFs attached to [a popular electronic health record system], and it's the worst OCR you have ever seen—straight from 1989."

Additional validity checks upstream of the warehouse or lakehouse frequently include detecting what language is being used and whether the data contains personally identifiable information (PII). This is because many models are trained on, and are most performant with, the English language, and privacy breaches are compliance nightmares. These are often best filtered further upstream in the initial cleaning and curation process within cloud object storage prior to chunking.

Consistency

Consistency, and particularly the dreaded *embedding drift* (changes in how the same data is numerically represented as a vector), is one of the foremost concerns for data + AI teams.

Vector cosine similarity, which we discussed earlier, is one of the most effective monitors to detect embedding drift. Another is monitoring the L2 norm, which measures vector magnitude (how far the vector is from the point of origin).

Some embedding models will normalize the magnitude of a vector so that it always equals 1, while others will use the magnitude to convey specific meaning, such as confidence level.

Anomaly detection on the L2 norm value of embeddings can help catch outliers that may indicate model change or other issues that may require a second look.

As one data + AI leader told us, "Having a way to confirm normalization took place would be really cool.… A few years ago I used to keep a record of defined model weights where I needed to apply L2 normalization. I'd also record the normalization values used for the training data so I could make sure the math all worked out how it was supposed to."

There are two other "drifts" that data + AI teams should be just as concerned about:

Schema drift
: Schema drift happens when the metadata or structure around your files changes—fields get renamed, removed, or added—but your pipelines don't adjust. Suddenly, the document is valid but the index is blind to key filtering or ranking signals. Schema change monitors on chunk and embedding tables can detect this drift.

Semantic drift
: Semantic drift happens when the meanings of terms or concepts shift over time and is the hardest to detect. For example, what happens when "risk" means two different things to two different domains? Or when teams can't align on one definition of "revenue," yet rely on embeddings to resolve meaning with no human in the loop? This may be best detected by evaluating model response (more on that later in this chapter).

Unstructured Monitoring Examples

Mapping some of these monitoring concepts across the embedding table we saw earlier (Table 3-1) would look something like Table 3-2.

Table 3-2. Example monitors on an embedding table

Field name	Field value	Monitor(s)
`Chunk_id`	`chunk_123`	Uniqueness
`Chunk_text`	`NASA awards SpaceX $1.5 billion for...`	NULL, string length, sentence boundary, encoding errors
`Chunk_overlap`	`150`	Distribution based on historic values (max, mean, min)
`Chunk_created_at`	`2025-06-1012:32:00`	Freshness
`File_id`	`file_123`	Percent unique
`File_type`	`PDF`	Allowed values

Field name	Field value	Monitor(s)
Object_path	s3://pdfs/contracts/file_123	
Object_last_modified	2025-07-0214:56:03	If value is after Chunk _created_at
Embedding_model	OpenAI-ada-002	Allowed values
Embedding_vector	[0.12, -.34, 3.53]	Cosine similarity, zero value, NULL, L2 norm, array length
Embedding_created_at	2025-07-108:45:03	If value is after Chunk _created_at
Language_detected	en	
Award_recipient	SpaceX	NULL
Award_amount	1,500,000,000	NULL

Root Cause Correlation

To evaluate what level of observability is required for accelerated root cause analysis on unstructured data quality incidents, it is helpful to consider the main processes and systems involved in these pipelines. In other words, from which gears in our unstructured data delivery assembly line do we need to gather telemetry? This is where the data, system, code, model framework comes into play. These are the four avenues through which bad data can be introduced.

For example, you can get bad data straight from the data source, from a system or job failure, from a change to the logic or code governing the transformation processes, and—even if this all goes according to plan—nondeterministic models can still introduce errors.

What is particularly insidious about these failure points is that they are not always under the control of even the most fastidious data + AI team. For example, one team we work with experienced drift because the embedding model provider—a major, modern data platform—automatically deployed a new version of the model. That might be OK for a low-stakes internal application with 10 users, but it's diabolical for a production system serving thousands of customers per minute.

Another data scientist we spoke with bemoaned his overactive engineering team saying, "We had an example where one of my data engineers changed from my preferred model Claude, where I do most of my training, and they swapped it out for some Llama model and the output diminished so fast."

As we laid out in the previous section, there are a few major stages to an unstructured data pipeline, including generation, cleaning, curation, chunking, and embedding. Let's take a look at some possible root causes using this categorization across the unstructured data lifecycle:

Data

- Source stops generating data
- Source changes metadata or schema
- Duplicate or incomplete data generated

System

- Permissions issues across tools
- Cleaning, chunking, or embedding jobs (directed acyclic graphs) fail or don't run
- Model API slow and unresponsive

Code

- Cleaning, curation, or chunking logic changes
- Metadata enrichment and extraction changes
- Embedding model or dimensionality changes

Model

- Response unfit for purpose (tone too aggressive, hallucinations, inconsistencies, etc.)

This is by no means an exhaustive list, but it does provide an idea of what components and processes require visibility from a data + AI observability solution.

> The ultimate raw unstructured data source is what it is, but we're looking at the fidelity of the information as it passes through.… The definition of success is not just that it executed without an error, but that also you really successfully extracted all the information that you want to extract.
>
> —VP data engineering, financial services company, March 2025

Structured Data

Structured data pipelines are much more mature than—yet just as unreliable as—unstructured data pipelines. This is because they have just as many fragile interdependencies across disjointed systems and teams.

Instead of cleaning, curation, chunking, and embedding, structured data pipelines generally employ a *medallion* architecture with data moving from raw inputs in a bronze layer, to standardized metrics in a silver layer, to use-case-specific aggregation within a gold layer.

As with unstructured data pipelines, each of these stages—from initial ingestion to each layer of transformation—is an opportunity for bad data to infiltrate. When AI reaches to retrieve data to inform its response or next actions, this bad data is often accessed directly from these gold (sometimes silver) tables with the agent issuing a simple SQL query.

This is why, according to Gartner, for data to be AI-ready, it needs to be consolidated in a source of truth for AI to access it, governed for AI to understand it, and observed to generate trusted outcomes (Figure 3-3).

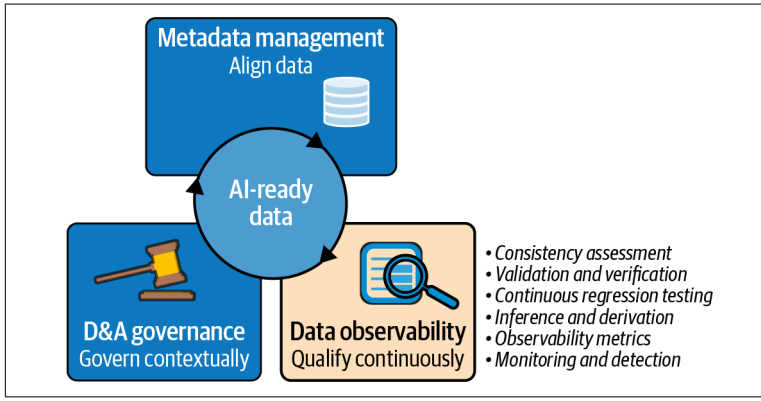

Figure 3-3. How Gartner defines AI-ready data

Monitoring Strategy

Barr coauthored *Data Quality Fundamentals* (O'Reilly) with Lior Gavish and Molly Vorwerck, which has more than three hundred pages detailing a modern approach to data quality across structured data pipelines, so we won't go into quite as much detail here.

When Barr coined the term "data observability" in 2019, she introduced the initial pillars, which include four core anomaly types:

Freshness
: Did the data update when expected?

Volume
: Did the table receive as many rows as expected?

Schema
: Did the column name, structure, or data type change?

Quality
: Is there an abnormal distribution in values, percent unique, NULLs, etc.?

At a high level, a structured data monitoring strategy should monitor data movement metrics (freshness, volume, schema), end-to-end across the pipeline while diving deeper into data quality metrics (consistency, uniqueness, distribution, etc.) at the initial ingestion and gold layers, as shown in Figure 3-4. This should be modified based on the specific use case, of course, but serves as a good initial template.

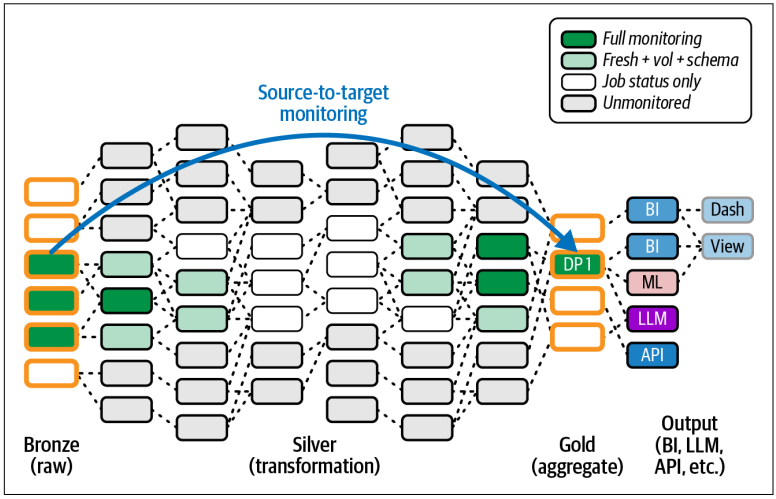

Figure 3-4. A structured data monitoring strategy across the lifecycle of a data product

Root Cause Correlation

One of the most frequent questions we get asked is, "What is the difference between data quality and data + AI observability?" While there are several nuanced differences, the simplest way to think about it is that while data quality and observability both start with detecting issues, only observability takes the next step in pointing to *why* there is an issue (aka the root cause).

To figure out the *why*, we again need to employ our data, system, code, model framework. We've seen the failure points already (refer to Figure 2-3), so this time let's look at the framework's application and common key root cause analysis features:

Data

- *Breached or anomalous rows*: When bad data is detected, particularly upstream in the pipeline, reviewing the breached rows to quickly determine whether garbage data was ingested from the source can be a helpful spot check.
- *Metric investigation*: The next step in sophistication is to then understand if there are patterns in these breached or anomalous rows. For example, does a spike of NULLs in the `contact_id` field correlate highly with the value "LinkedIn" in the `ad_source` field?
- *Cross-system data lineage*: Tracing an issue via data lineage across systems can help identify the point of origin, a crucial step in the root cause analysis process.

System

- *Job failure correlations*: When a data issue arises, the metadata of upstream jobs can be analyzed to identify any relevant job failures that may be the root cause.
- *Job runtime metrics*: Similarly, understanding whether the time needed to run a job increased or decreased significantly can indicate issues like incomplete loads or duplicate data.
- *Permission insights*: Pipelines have multiple systems, each of which requires permissions and credentials for interoperability. These change and fail from time to time.

Code

- *Query and pull request (PR) change detection*: If there was a change in the query code or a new PR was issued immediately prior to an anomaly, it is highly likely that it may be the root cause.

Model

- *Agent monitors*: AI agents and the underlying LLM models are nondeterministic and can hallucinate or produce unfit responses that can introduce quality issues.

By consolidating these signals and understanding the relevancy to a specific issue via lineage, AI agents can be leveraged to dramatically accelerate the root cause analysis process.

AI

Even if the data is perfect and "AI-ready," there may be errors in the retrieval, inference, and response process.

While evaluation criteria are important to consider during model development, they become imperative to monitor once in production and scaling up to thousands of users. At that point, manual evaluation will no longer be sufficient.

As we saw earlier, the most effective strategy for monitoring unstructured data is to structure it. In a similar fashion, one of the most effective ways to monitor AI models in production is to structure and consolidate the outputs and key telemetry (traces and spans) within an inference table in the warehouse or lakehouse. There are several benefits to this approach, including an auditable history, ease of governance, and the ability to compare across other datasets that may serve as the "ground truth." An example architecture may look like Figure 3-5.

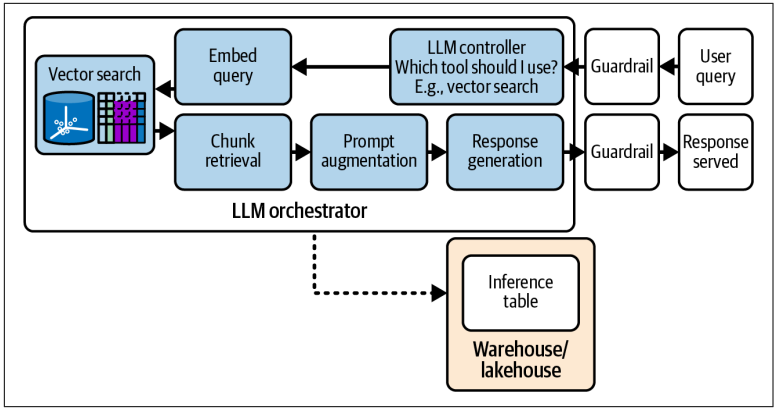

Figure 3-5. AI or agent observability architecture

Monitoring Strategy

AI models should be monitored for several types of issues or anomalies:

Failure
: A technical failure or timeout occurs, and a response is never returned.

Refraining
: The model excessively indicates that it doesn't know, defers to other sources, or refuses to answer.

Wrong response
: The model returns a hallucination or a response not grounded in the retrieved context.

Unhelpful response
: The response is irrelevant to the prompt, not actionable to the user, or uses the wrong tone.

Damaging response
: The response is toxic or results in privacy or security issues.

Whenever feasible, teams should leverage heuristic (rule-based) monitors to spot these types of issues. This is frequently the most cost-efficient and explainable approach at enterprise production scale.

Some heuristic AI monitoring approaches for each type of issue include:

Failure
: Monitor to detect errors in the log data or spans with high latency for technical failures, detect NULLs in the response output, and detect excessive retry loops within the telemetry.

Refraining
: Employ regex monitors to detect spikes in phrases such as "I'm not sure, but" or "I don't have enough information to answer that." Monitor to detect that no embedding retrieval occurred and the subsequent response was short in length.

Wrong response
: The distance between the query embedding and retrieved embedding or the cosine similarity could be evaluated to indicate a gap between the semantic meaning of the question and the retrieved chunk.

Unhelpful response
: Monitor for overly long or short responses.

Damaging response
: Monitor to detect logs from guardrail systems indicating moderated inputs or `completion_blocked`.

However, this is only half the puzzle. For one thing, the team must know the likely outputs and alert conditions. For another, heuristic approaches struggle with more subjective evaluations such as tone, relevance, and even correctness. This is why the "unhelpful response" category is pretty sparse.

Data + AI teams frequently deploy a complementary monitoring or evaluation approach using an LLM as a judge. While monitoring AI with AI may seem counterintuitive, it is often effective since each model is given a different objective. For example, one model might be given the objective "produce a helpful response," while the LLM judge is given the more skeptical objective "evaluate whether this response fits this specific criteria."

> I would say we're early to mid maturity. We're not just operating on vibes anymore, we have a framework in place by which we're capturing inputs and outputs from the LLM. We have a system where we ask the LLM-as-judge, how did the first LLM do the task?
>
> —Data scientist, media company, July 9, 2025

These monitors can detect issues with response reliability, clarity, helpfulness, and faithfulness (prompt adhesion). Some LLM-as-judge monitoring approaches to detect these issue types include:

Failure
: Evaluation models can be prompted to evaluate whether the AI agent leveraged tools at its disposal when it should have or whether it accessed the right tools.

Refraining
: An AI model that responds "I don't know" is much better than one that hallucinates. However, it's important to evaluate whether the AI had the appropriate context to respond to the question accurately.

Wrong response
: Models can evaluate the factual correctness of the response and whether the response was grounded in the context retrieved.

Unhelpful response
: Models can evaluate the tone or sentiment of the response, whether the response was relevant to the user query, how closely the response followed the prompt instructions, and whether the response was actionable and the task was completed.

As you may have noticed, it's particularly important to evaluate relevance across the RAG chain: Was the retrieved context relevant to the query? Was the response grounded in the retrieved context? Was the response relevant to the query?

LLM-as-judge monitors need to be deployed diligently, as they can be costly. Some of the teams we work with have reported their evaluation costs were ten times higher than the inference cost. Additionally, the nondeterministic nature of AI means the evaluations can vary even when the inputs do not.

Again, leveraging heuristics and non-LLM evaluations whenever possible can help. Other methods to mitigate these downsides include sampling, using evaluation tables and test cases, anomaly detection, and supplementing with human labeling.

> We have six metrics, four of them are based on LLM-as-a-judge [evaluations. Two are based on accuracy.] We also ask a model if the sources are used correctly and also check if the answer is actually grounded in the documents.… And the [other] metric is a simple cosine similarity metric [between golden answers and LLM answers].
>
> —System architect, data and analytics, June 3, 2025

Root Cause Correlation

After teams have detected the issue, the next question is "why did this happen?" Answering this question is the difference between monitoring and observability. The data, system, code, model framework is once again extremely relevant and helpful for this type of root cause analysis.

For a quick (but not exhaustive) illustration, let's map some possible root causes for each type of AI reliability issue mentioned in the previous section (Table 3-3).

Table 3-3. Possible root causes for types of AI reliability issues

	Data	System	Code	Model
Failure	Context window exceeded	Latency exceeds limit set by orchestrator	Excessive retry loops	Overpowered model leads to excessive latency
Refraining	Conflicting data lowers certainty	Cost controls limit token usage	Overly cautious or restrictive prompt instructions	Training dataset (e.g., help desk tickets referring to user guides)
Wrong response	Data not available or outdated	Retrieval failure	Prompt points model at wrong tool set	Model version drift
Unhelpful response	Embedding drift	Overly restrictive guardrails	Poor chunking logic (splits mid-sentence)	Model not fit for use (struggles with domain)

It's particularly important to point out that many of these root causes will be found upstream in the unstructured and structured data pipelines, which is why the only practical solution to AI reliability is not AI observability but data + AI observability.

Unlike the distributed nature of structured and unstructured data pipelines, much of the telemetry required for AI observability is consolidated and can be extracted from LLM orchestrators like LangChain and LangSmith. Standards like OpenTelemetry and OpenLLMetry are also emerging to provide a common framework for capturing this critical data, which includes user query, span latency, retrieved chunks, prompt version, model version, and more.

> They [LLM orchestrator and model registry] spit out the inference tables which give you everything you need if you really want to go through and look at it, but it's a lot of information.… You can't go through hundreds of examples very easily. So we're doing some manual review just to check accuracy. [We need to automate and scale this further].
>
> —Senior manager, pharmaceutical company, April 10, 2025

Final Thoughts

We started this report with the headline "AI is easy. Data + AI is hard." Hopefully, it is now clear just how true that statement is. These are complex, constantly evolving systems of systems with multiple fragile dependencies.

We've now arrived at a point where we can truly look at the big picture of what data + AI observability means, as shown in Figure 3-6.

As we think back to the hundreds of data + AI teams we have talked to, and the pain and best practices they shared that form the core of this chapter, one insight stands out from all the others shared in this report. It's an insight most eloquently captured by our colleague, and we can't think of a better conclusion:

> Just monitoring your data or your AI is like only wearing sunscreen on half your body at the beach.

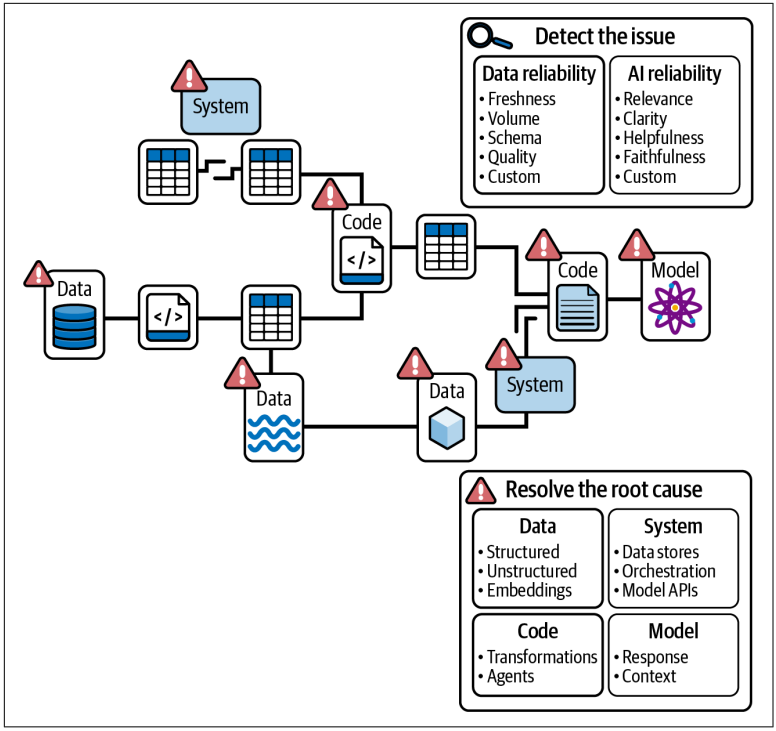

Figure 3-6. A data + AI observability solution monitoring for reliability issues and providing visibility into root causes

The challenges are great, but so are the rewards. The teams that deliver reliable data + AI products will ultimately become the winners of the AI era.

Thank you for reading this report. We hope it helps you on your data + AI reliability journey.

About the Authors

Barr Moses is the CEO and cofounder of Monte Carlo, a data + AI observability company. In her decade-long career in data, Barr has served as commander of a data intelligence unit in the Israeli Air Force, a consultant at Bain & Company, and VP of operations at Gainsight, where she built and led their data and analytics team. The instructor of O'Reilly's first course on data observability, an emerging discipline in data engineering, Barr has worked with hundreds of data teams struggling with these problems.

Michael Segner has spoken with and advised dozens of data teams on their quality- and reliability-related initiatives in his role leading product marketing at Monte Carlo. Prior to Monte Carlo, Michael led content marketing at AvePoint, a software platform for managing and governing unstructured data within the Microsoft ecosystem.