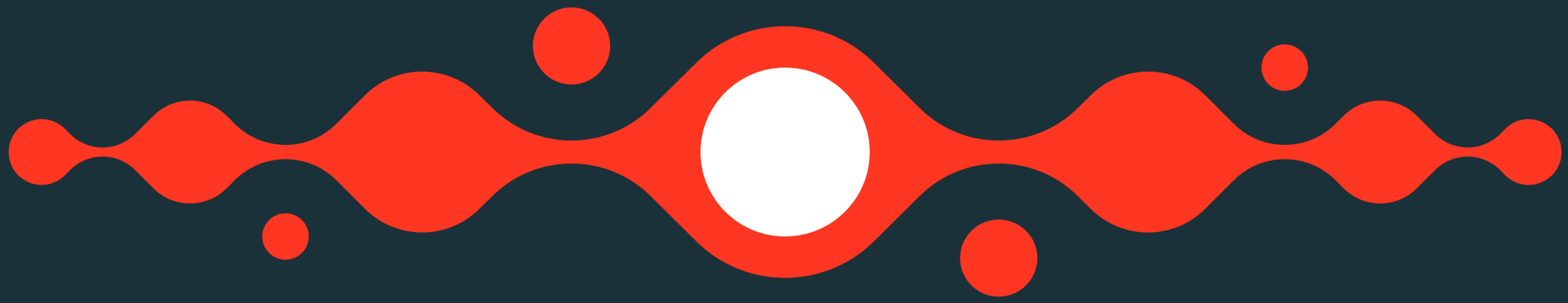
The Big Book of GenAI



5 ways to leverage your data to build production-quality GenAI applications

databricks

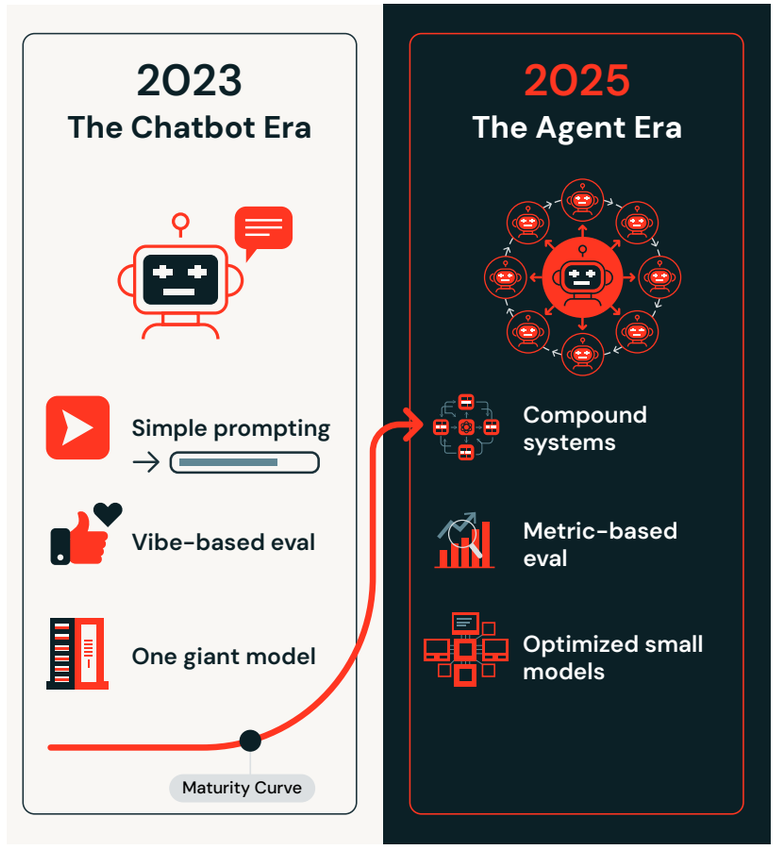
CONTENTS

Introduction 3
Chapter 1: Foundations of GenAI and Agent Quality 7
Chapter 2: Building AI Agents 14
Chapter 3: Working With Data 27
Chapter 4: GenAI Performance and Evaluation 35
Chapter 5: Governance for GenAI and Agents 46
Summary 54

Introduction

From hype to engineering: The 2025 GenAI shift

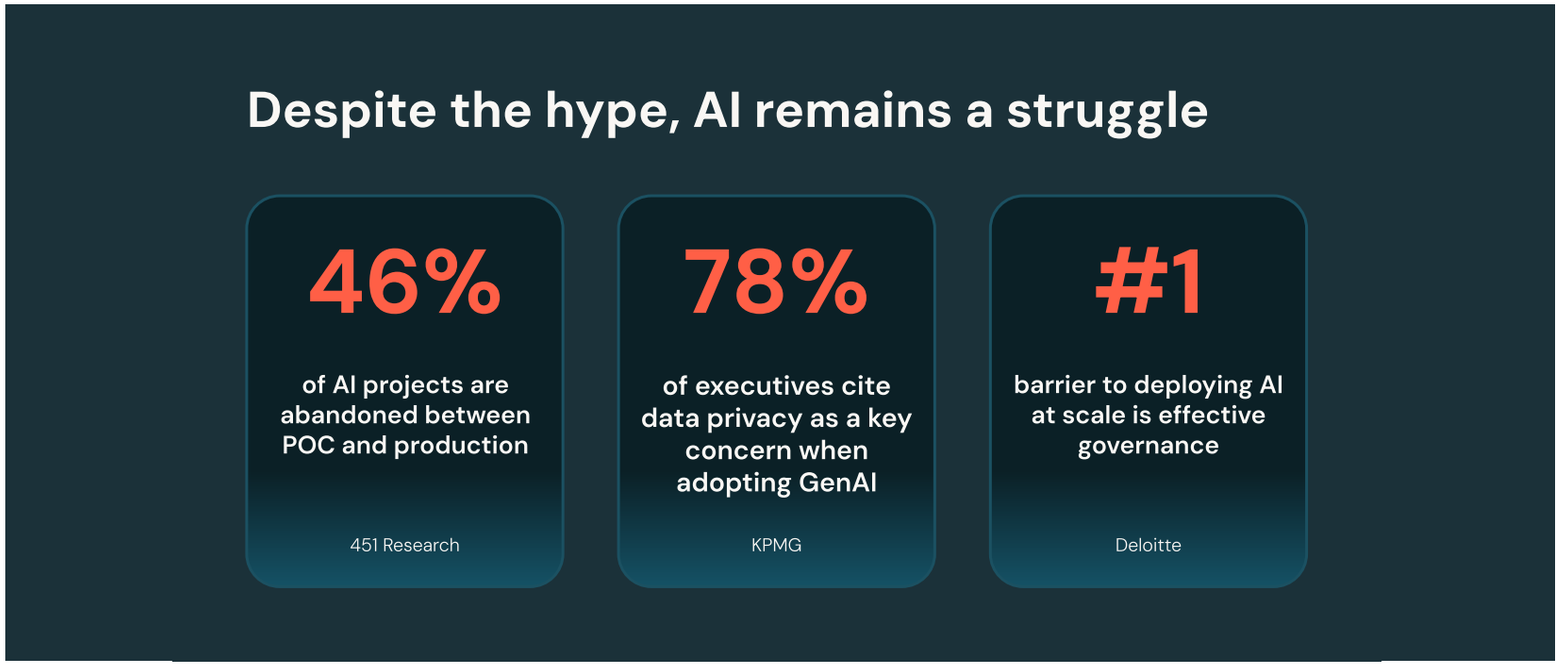
Generative artificial intelligence (GenAI) has fundamentally shifted the horizon of what is possible in software. For the first time, we can build applications that reason, create and interact with the nuance of a human expert. The promise is undeniable: a workforce of intelligent agents capable of analyzing complex financial reports, resolving customer support tickets and optimizing supply chains in real time.



According to a recent report, Unlocking Enterprise AI, 85% of surveyed global enterprises are already using GenAI in some capacity. The pilot phase is over, and the focus has shifted to production. However, moving from a proof of concept (POC) to a production application presents significant engineering challenges.

While building a chatbot that answers correctly most of the time is straightforward, building an agent that is accurate, secure and cost-effective at scale requires a robust architecture. The industry is currently facing a "reliability wall." Research indicates that 95% of AI agent POCs don't reach production deployment due to challenges in quality, governance and cost.

The three blockers to production



Organizations are finding that the tools used to build prototypes are often insufficient for the demands of the enterprise. They face three critical blockers:

- **Quality:** Frontier models are powerful but can be prone to hallucination. Without robust evaluation frameworks and optimization techniques, agents often fail to meet the strict accuracy thresholds required for customer-facing or decision-critical applications.
- **Governance:** Agents need to act, but governance is often cited as the most significant barrier to deploying AI at scale. Giving an agent permission to query databases, access files or email customers requires a security model that many enterprises haven't built yet.
- **Model choice:** AI innovation is happening at a breakneck pace. The best model today might not be the best model tomorrow. Additionally, bigger models might be too slow and too expensive for massive scale. The new mandate is to find the most efficient model, or a combination of many smaller models working together, to achieve high quality at a fraction of the cost.

To meet these challenges, we must stop treating GenAI as magic and start treating it as engineering.

Agent Bricks: Unifying the AI lifecycle

Success in this new era requires a unified approach. You can't address quality in isolation from data, and you can't solve governance in isolation from serving. Databricks Agent Bricks manages this fragmentation by unifying the lifecycle from data collection and preparation to model development, deployment, monitoring and governance.

- **Delivering quality with data:** You can't build a high-quality agent on incomplete data. Agent Bricks unlocks the 80% of enterprise knowledge trapped in unstructured documents like PDFs and images and uses it to ground your agents in reality.
- **Ensuring governance with unity:** Instead of moving your data to a model, we bring the model to your data
- **Optimizing innovation with model choice:** We move beyond the one-model-fits-all approach. Agent Bricks enables you to orchestrate every frontier model (OpenAI, Google, Anthropic, Meta, etc.) so you have the right models for the right tasks.

Chapter 1: Foundations of GenAI and Agent Quality

What is GenAI?

GenAI focuses on creating models capable of generating new content, including text, images, code and synthetic data. Unlike traditional AI models designed to classify (e.g., "Is this email spam?") or predict (e.g., "What will sales be next month?"), GenAI creates outputs that resemble or extend existing content.

These models are typically large language models (LLMs) or foundation models that can be adapted for specific applications. However, the definition of GenAI in the enterprise has shifted. In 2023, "GenAI" meant a chatbot. In 2025, it means agentic systems — software that leverages techniques such as advanced prompt engineering and retrieval augmented generation (RAG) to autonomously reason, plan and execute tasks.

Although many GenAI models include safeguards, they're probabilistic engines, not deterministic databases. They can produce inaccurate or harmful outputs, making careful architecture and governance essential for deployment.

Enterprise AI: Beyond superintelligence

While much of the public discourse around AI focuses on achieving superintelligence, the reality is that 95% of enterprise AI use cases don't require superhuman capabilities. They require mundane but strategically important tasks: extracting information from invoices, answering support questions based on policy documents and automating data entry.

The challenge isn't whether AI can perform these tasks — it's whether it can do so reliably enough for production.

THE RELIABILITY GAP: FINDINGS FROM OFFICEQA

Research from Databricks has quantified this challenge. We introduced OfficeQA, a benchmark designed to capture realistic enterprise work scenarios rather than academic trivia. OfficeQA evaluates how well AI systems can handle common business tasks using the U.S. Treasury Bulletins — a corpus of nearly 89,000 pages spanning over eight decades.

The results reveal a stark reliability gap:

- **The "vibes" deception:** Without access to the specific corpus, frontier models answer only approximately 2% of questions correctly. They sound confident, but they're hallucinating.
- **The RAG ceiling:** Even with access to relevant PDF documents via a standard RAG pipeline, frontier agents plateau at less than 45% accuracy across all questions and **less than 25% accuracy on the hardest subset**

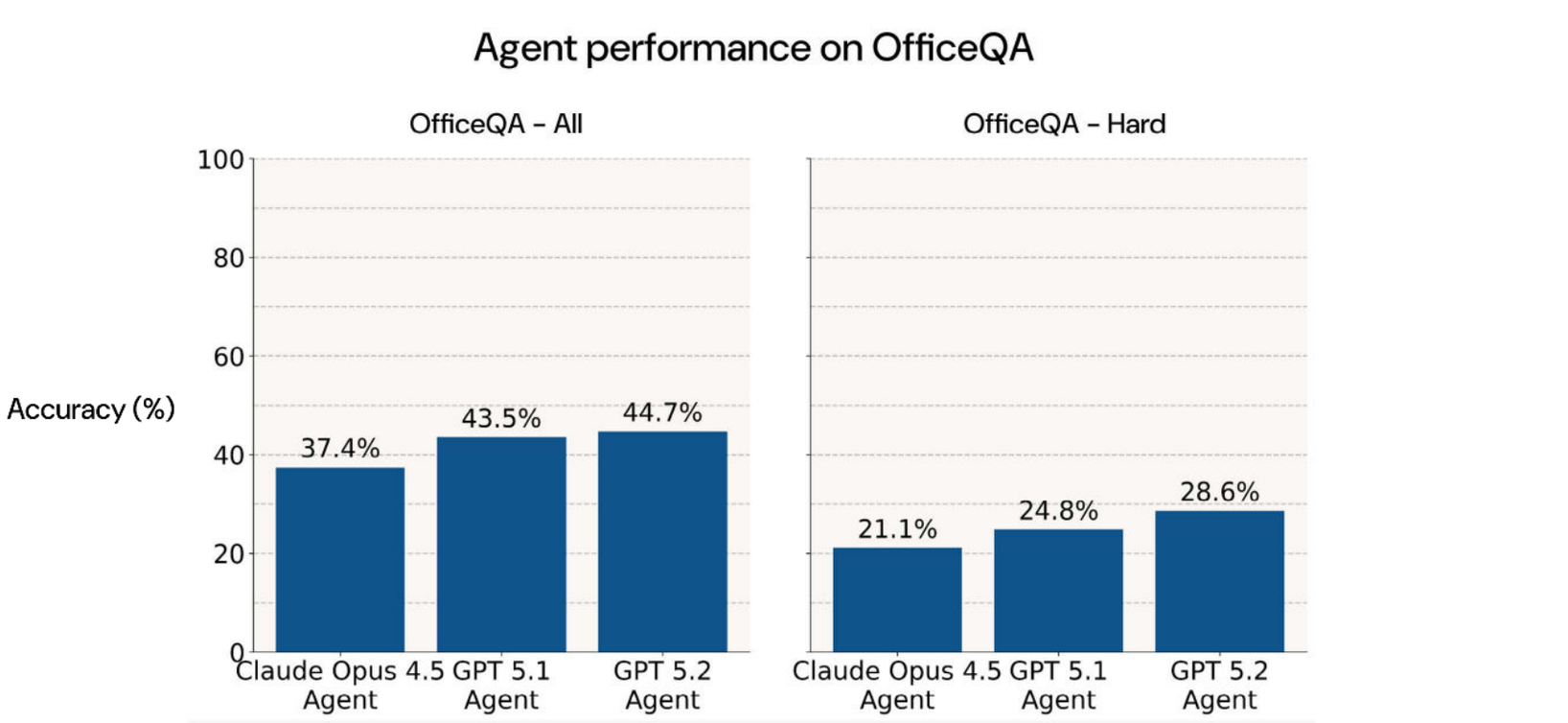


Figure 1: Agent performance on OfficeQA. Note the drop to <25% accuracy on the "Hard" subset (right) compared to the overall average (left).

WHY FRONTIER MODELS FAIL

Current frontier LLMs, despite their impressive capabilities, still struggle with grounded reasoning. The primary point of failure is often not the model's intelligence but its ability to ingest the data correctly.

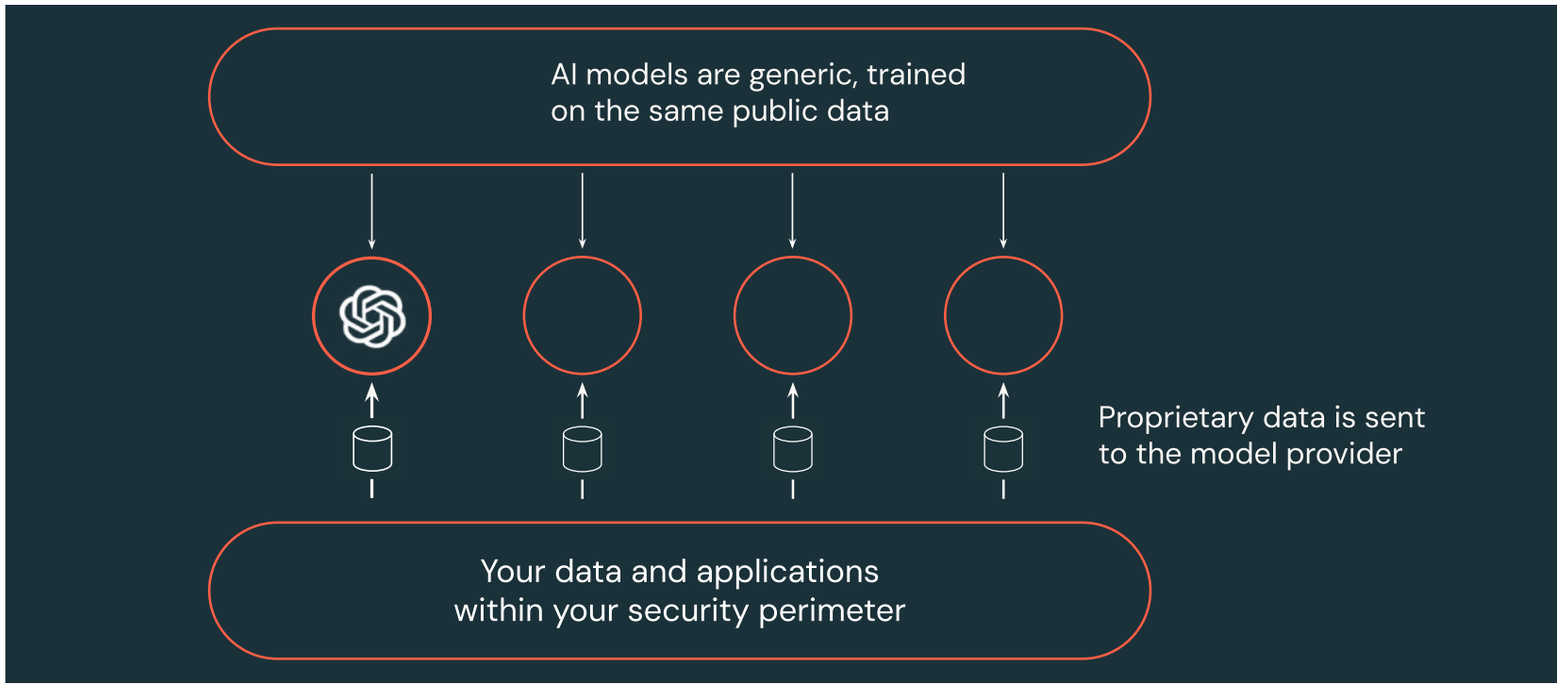


Figure 2: The siloed approach. Frontier models are trained on the public internet — they often fail in the enterprise because you need to ship your proprietary data to the models outside your security perimeter.

- **Document complexity:** Enterprises run on PDFs, not text files. A model can't reason about a table if the extraction tool scrambles the rows and columns.
- **Information aggregation:** Questions like "What was the total revenue for Department X across 2020–2023?" require finding four distinct numbers in four different files and performing math

Crucial insight: The OfficeQA findings show that data preparation and document parsing significantly impact accuracy — often more than model selection. A smaller model with perfect data parsing usually outperforms a larger model with poor parsing.

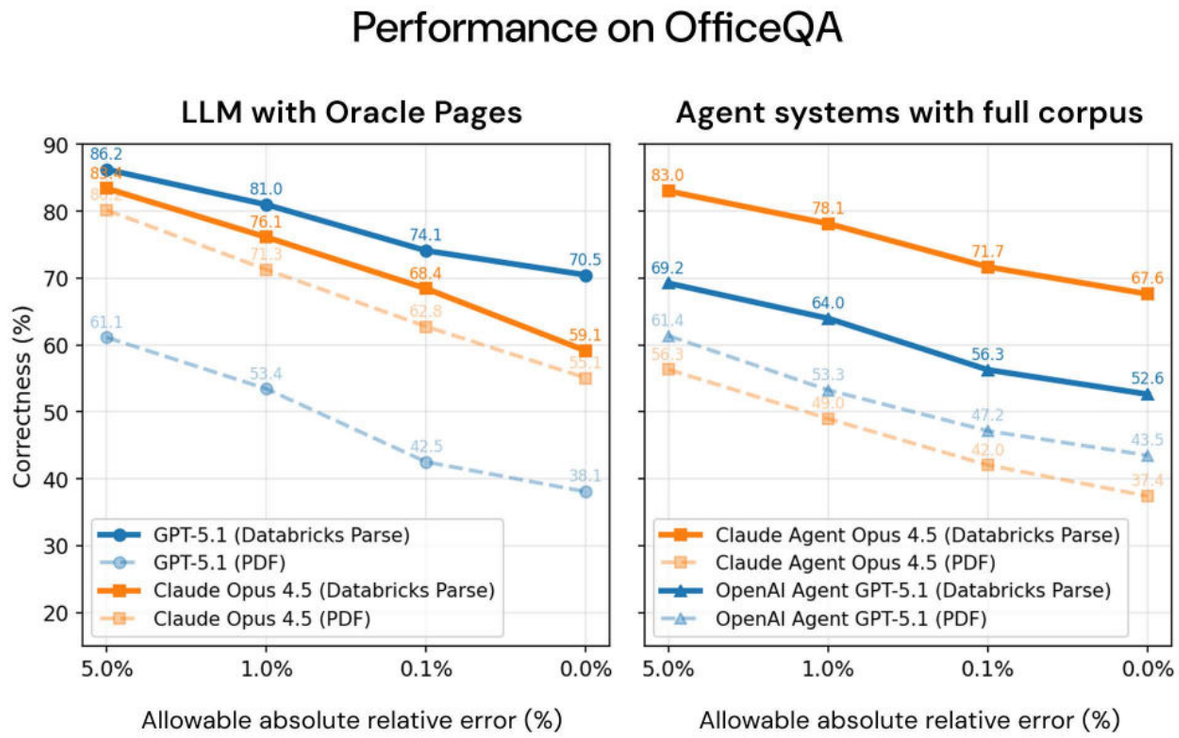


Figure 3: Impact of data parsing. OfficeQA agent performance chart showing performance with and without document parsing. The solid lines (Databricks Parse) significantly outperform the dashed lines (standard PDF) across all error thresholds, proving data quality drives accuracy.

Model selection and benchmarking: The quality-cost frontier

Choosing the right LLM is no longer about picking the smartest one on the leaderboard. It's an engineering decision involving trade-offs between accuracy, latency and cost.

THE QUALITY-COST PARETO FRONTIER

We visualize this trade-off using the Pareto frontier. The Pareto frontier is a set of optimal solutions in multi-objective problems where you can't improve one goal without worsening another, representing the best possible trade-offs between conflicting objectives like cost vs. quality, or speed vs. accuracy. For instance, in Figure 4:

- **The y-axis represents quality:** How accurate is the model on your specific benchmark?
- **The x-axis represents cost:** How much does it cost to process one million tokens?

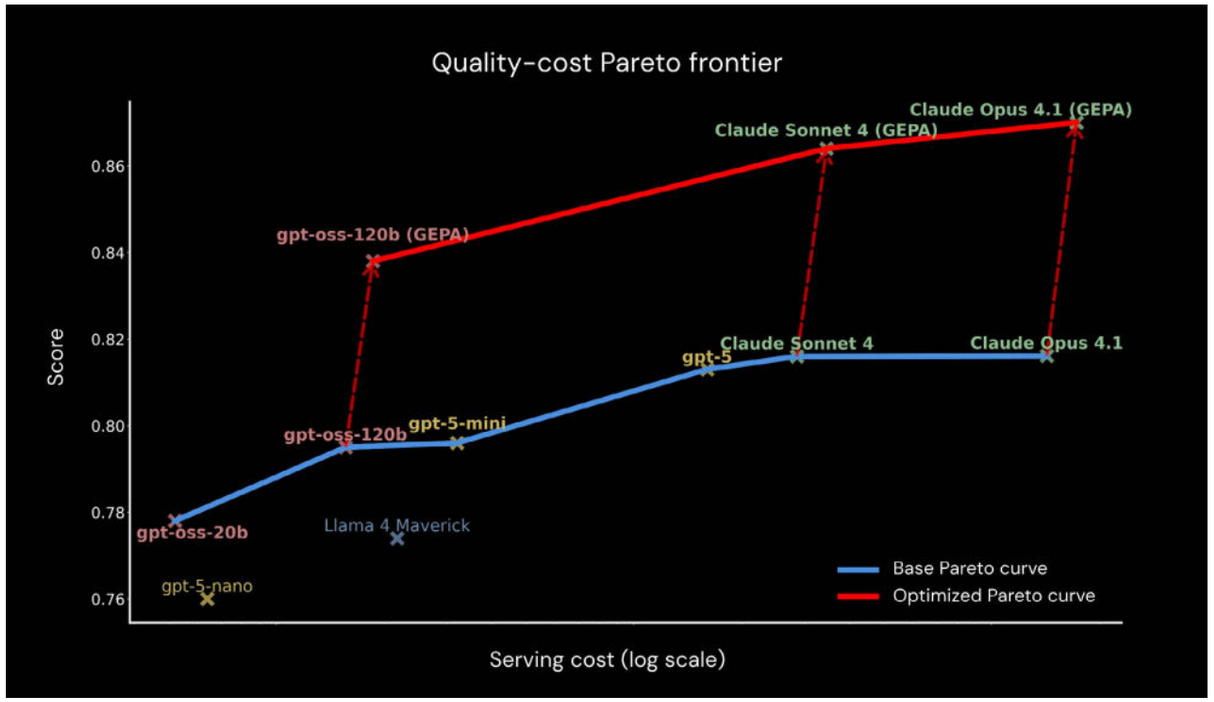


Figure 4: Quality-cost Pareto frontier showing baseline model performance vs. serving cost. Each point represents a different model's trade-off between quality, as measured by the IE bench (text-driven Image Editing Benchmark suite) score and cost per one million tokens.

Points on the frontier represent optimal choices — you can't improve quality without increasing cost, or reduce cost without sacrificing quality. Models below the frontier are suboptimal because you're paying too much for too little performance.

The 2025 shift: Recent research reveals that optimization techniques — like automated prompt optimization and fine-tuning — can shift open source models upward to the frontier. In fact, optimized open source models can now match proprietary performance on specific tasks while being 90x cheaper to serve.

Building and adapting LLMs for specific use cases

To bridge the reliability gap, organizations must move beyond "stock" models. There are three primary levers to adapt an LLM to your enterprise data.

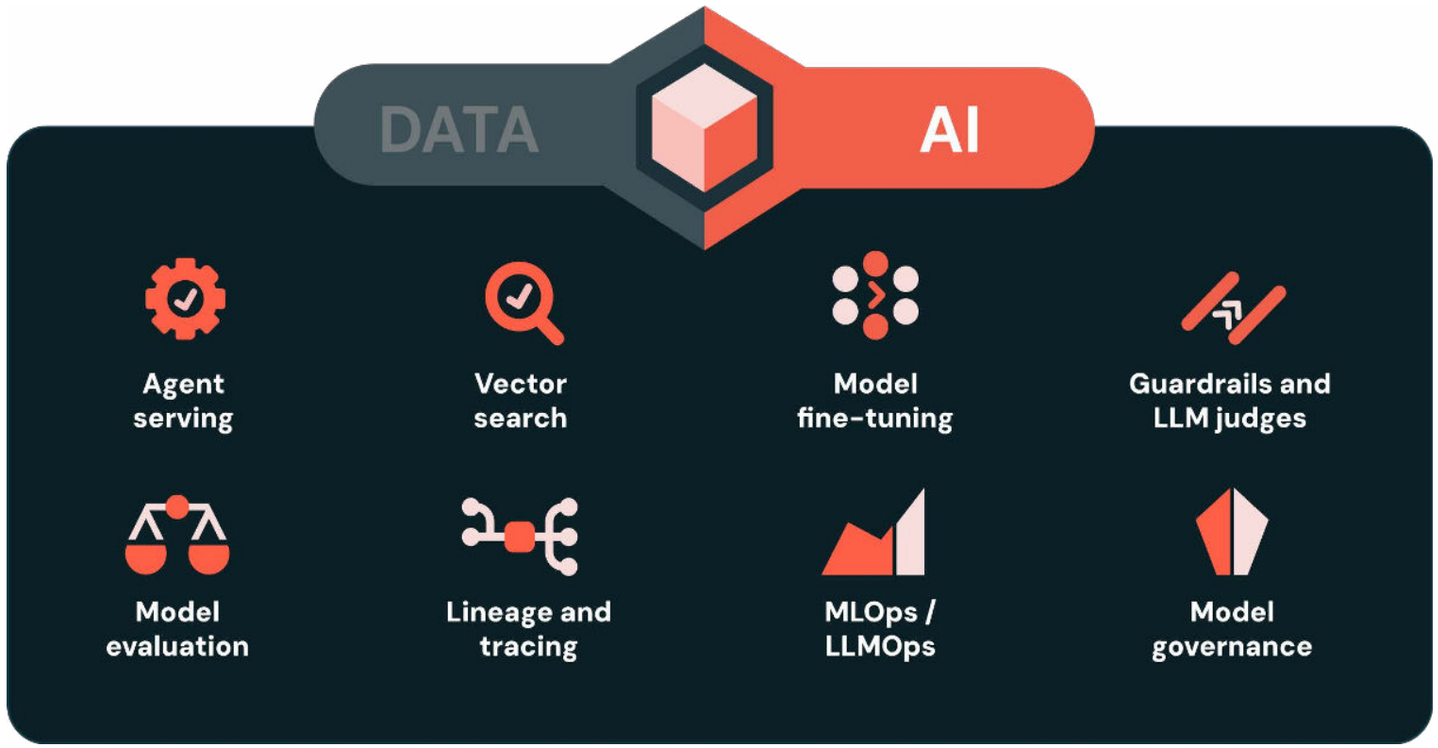


Figure 5: Building AI agents requires many capabilities, including serving agents, deploying vector databases, model fine-tuning, deploying guardrails, LLM judges, lineage, tracing, governance and MLOps/LLMOps.

- **Prompt engineering and optimization:** Rather than modifying the model, you modify the input. This includes either manually crafting instructions or using automated prompt optimization, such as declarative self-improving Python (DSPy) or Genetic-Pareto (GEPA), to programmatically search for the best prompt structure.
- **RAG:** RAG combines the LLM with a retrieval system like vector search. It allows the model to read your proprietary data — PDFs, wikis, databases — before answering. This is the standard architecture for most enterprise knowledge apps.
- **Fine-tuning:** This involves updating the model's weights on a curated dataset. This is effective for teaching a model a specific style (e.g., "Write SQL in our dialect") or behavior (e.g., "Always cite the document ID"). New techniques like test-time adaptive optimization (TAO) allow for fine-tuning on reasoning paths rather than just final answers, drastically improving logic capabilities.

From benchmark to production

Understanding these foundations explains why the failure rate is so high. 95% of POCs fail because teams assume that quality is a property of the model they bought. In reality, quality is a property of the system they build.

AI agents fail because of:

- **Poor quality:** They produce outputs that are of low accuracy or are not relevant to the initial prompt. This is because organizations feed raw, unparsed PDFs into a model and expect it to understand complex tables — the OfficeQA problem.
- **Weak governance:** They don't achieve the rigorous data and AI governance needed in enterprise settings. This is because organizations build a prototype that works for one user but can't handle permissions for 10,000 users.
- **High cost:** They build a dependency on an expensive frontier model that makes ROI negative at scale

The rest of this book is dedicated to solving these three specific engineering problems.

Chapter 2: Building AI Agents

AI agents are advanced applications that combine various AI technologies to perform complex tasks autonomously, enhance decision-making and improve operational efficiency. These agents bring together components such as LLMs, classical machine learning (ML) models and enterprise data to deliver intelligent, context-aware solutions.

AI agents excel in a range of use cases, from automating customer support and generating personalized recommendations to detecting fraud and forecasting trends. By integrating structured data analysis with natural language understanding, they provide precise, data-driven insights while maintaining the flexibility to handle unstructured information. This makes them invaluable across industries, helping businesses streamline operations, improve user engagement and scale AI-driven solutions effectively.

However, similar to broader GenAI POCs, 95% of AI agent POCs don't make it into production.

The main challenges are clear:

- Organizations can't guarantee the AI agent will produce quality and accurate results
- Many AI agents are locked into a single model provider, which stalls innovation, especially as AI is moving so quickly
- Most AI agents require you to replicate or violate your existing security and governance protocols

Databricks Agent Bricks empowers organizations to build high-quality AI agents. It improves quality and accuracy by building custom benchmarks using your own data and tasks, then evaluating every output against those benchmarks. Leveraging the latest research and human feedback, Agent Bricks automatically improves performance, so your agents stay accurate without costly rebuilds. By being model-agnostic, Agent Bricks can use any AI — commercial models, open source models or fine-tuned variants — and swap them in or out as innovations emerge.

Databricks lets you bring AI directly to your data to extend the governance, security and compliance controls you already use. This delivers value by enforcing access policies, applying rate limits to manage costs, preventing harmful content and providing end-to-end lineage — all without ever moving data outside your organization.

What is an AI agent?

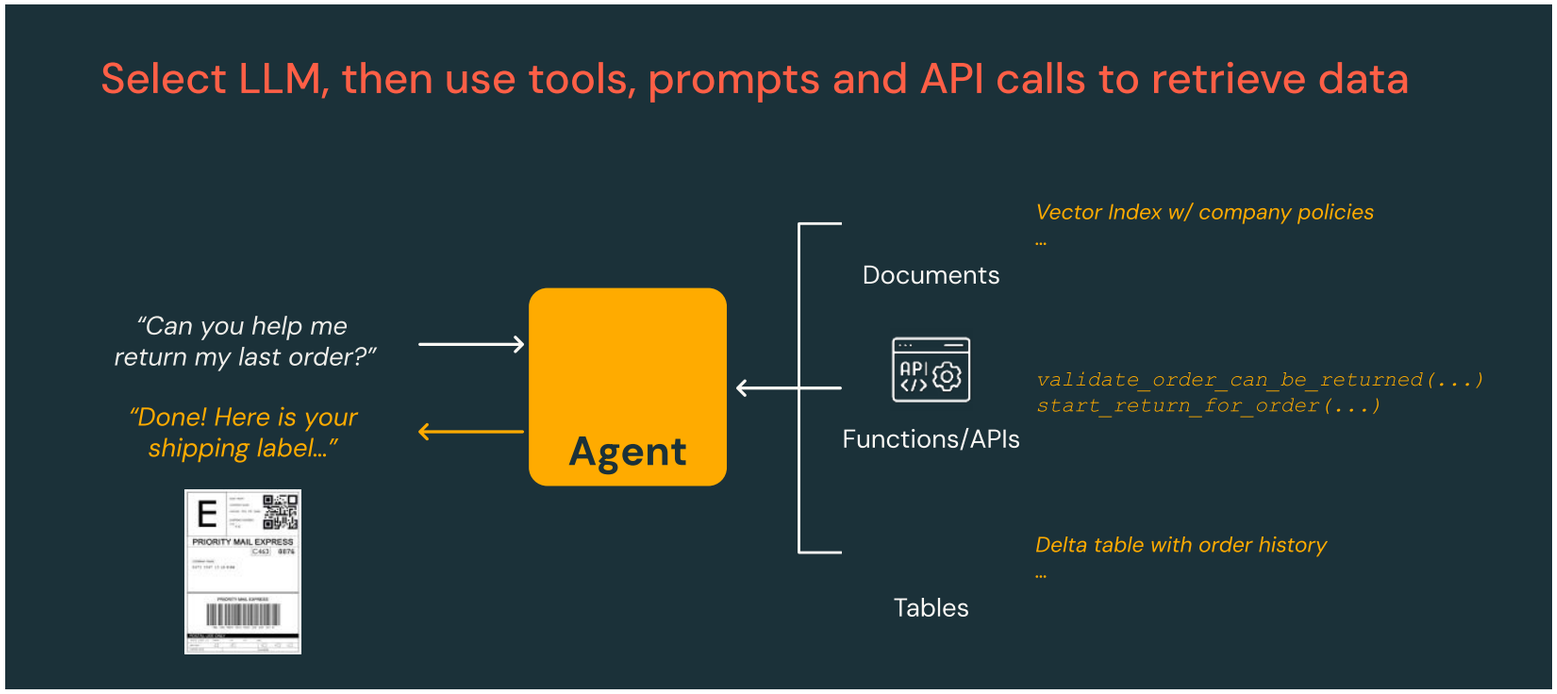


Figure 6: An AI agent has the ability to take autonomous multi-step actions based on the request.

To build and deploy an effective AI agent, an AI agent system is essential, whether it involves a single agent or multiple interacting agents. AI agents are intelligent applications designed to automate tasks and enhance human productivity. They can analyze information, make decisions and take actions to achieve specific goals, freeing up time and resources for strategic initiatives. For example:

- **Customer service agent:** Interacts with customers to understand their queries and provide relevant responses, improving satisfaction and resolving issues efficiently
- **Campaign generation agent:** Analyzes data, identifies target audiences and generates personalized marketing campaigns to meet specific objectives, such as increasing brand awareness or driving sales
- **Code generation agent:** Assists developers by writing, debugging and optimizing code based on given requirements, streamlining the software development process

What goes into an agent pipeline?

An AI agent pipeline empowers enterprises to design and operationalize intelligent agents capable of executing complex tasks. Unlike stand-alone models, these systems integrate diverse components such as LLMs, classical ML models, enterprise data and external tools to achieve specific objectives efficiently. Additionally, AI agent systems include built-in evaluation techniques and governance mechanisms to ensure quality, accountability and compliance with organizational standards.

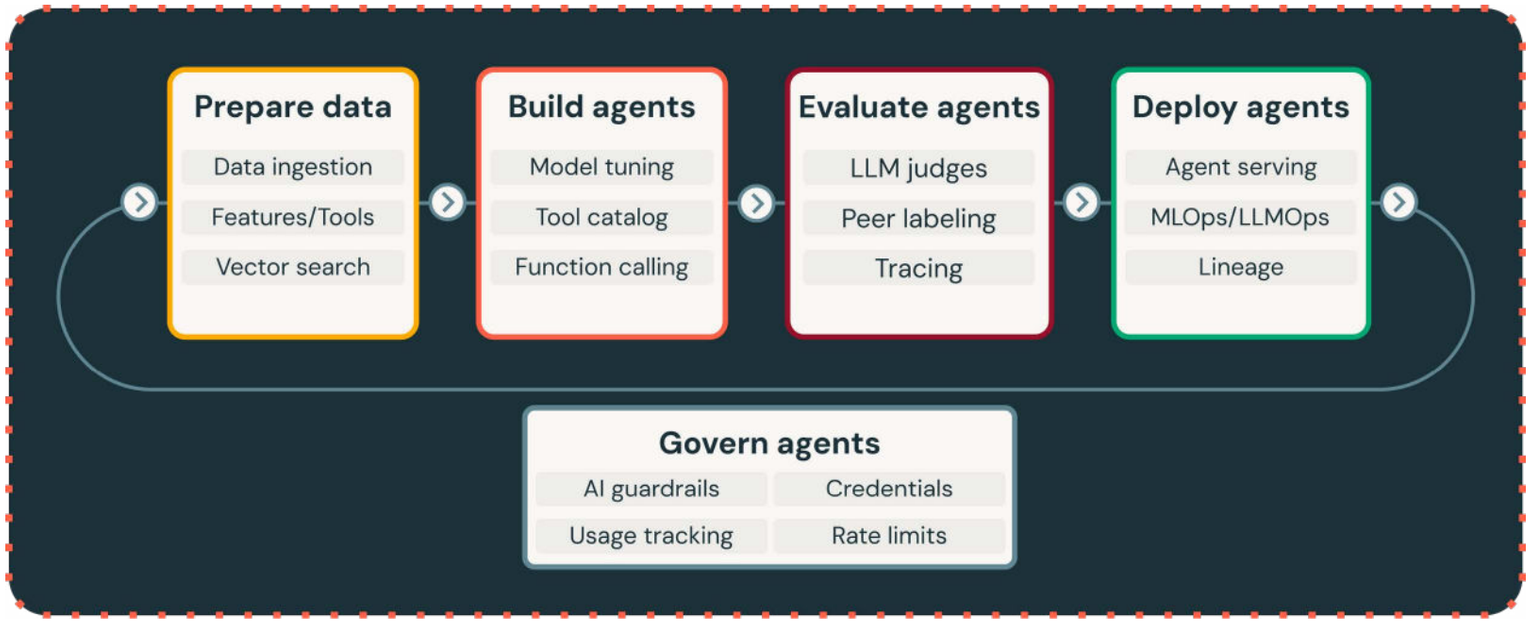


Figure 7: AI agent system development stages include preparing data, building, evaluating, deploying and governing agents.

KEY STAGES OF DEVELOPING AN AI AGENT SYSTEM

- **Prepare data:** Data is organized and preprocessed to ensure it's clean, accessible and relevant for decision-making and agent interactions
- **Build agents:** GenAI models, classical ML models and tools are combined to create agents tailored to specific tasks
- **Deploy agents:** Agents are packaged for performant, secure and efficient use by other users and systems
- **Evaluate performance:** Agent outputs are measured and assessed to ensure they meet objectives. Iterative improvements are made based on feedback.
- **Govern operations:** Security, compliance and ethical standards are maintained while tracking agent activities to ensure transparency and accountability

Core components

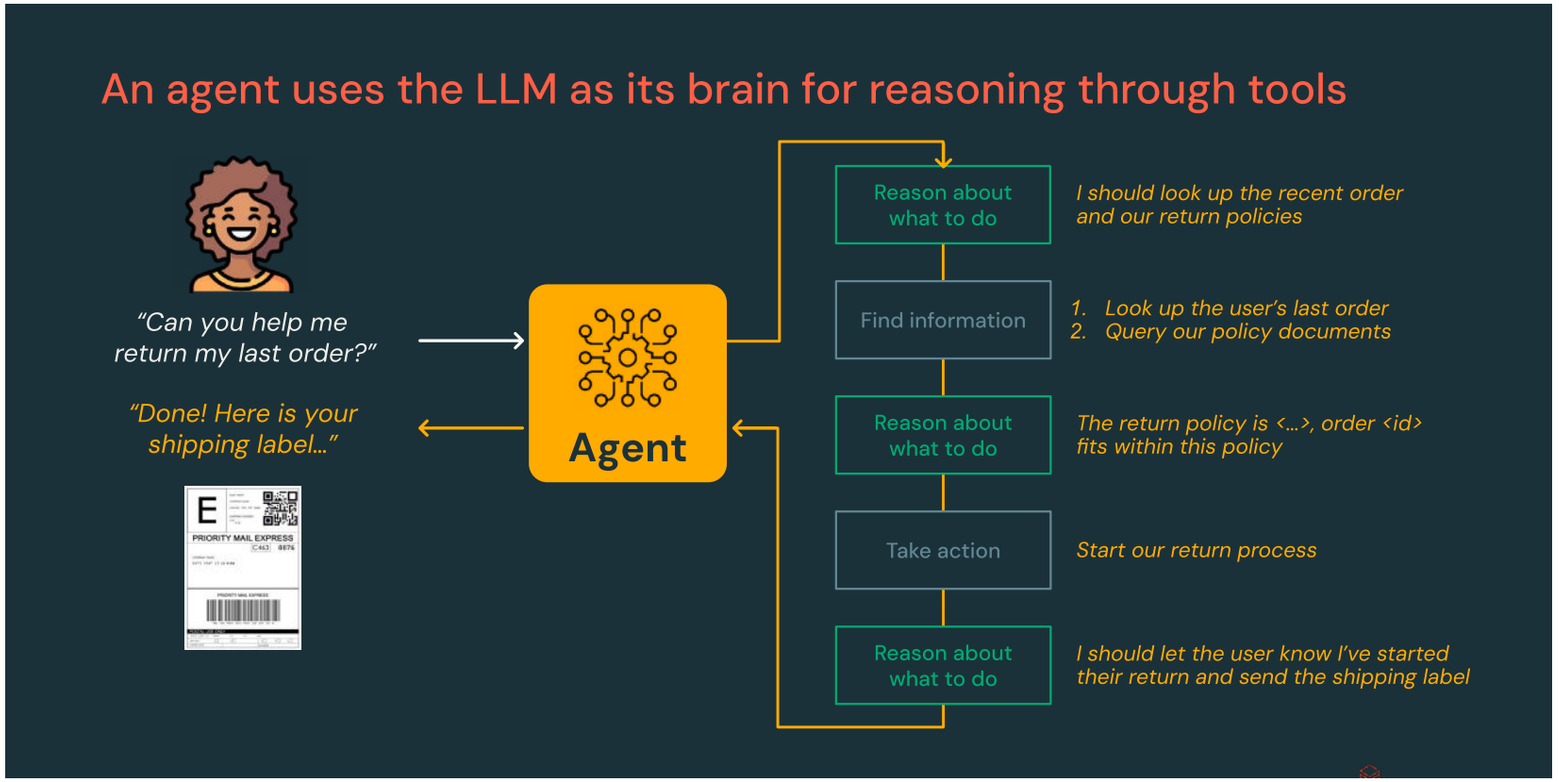


Figure 8: An LLM is used as the central brain to orchestrate and take the next best action.

A robust agent system consists of four primary components working in a loop.

- **The LLM (the central brain):** The reasoning engine. Its job isn't necessarily to know the answer but to decide what to do next. It receives the user's goal and the current state and outputs a "thought" or "action."
  - *Key requirement:* The model must be fine-tuned for function calling — the ability to output structured JSON for API calls rather than conversational poetry
- **The planner:** For complex tasks, the agent can't just guess the next step. The planner is a module that breaks a high-level goal into a sequence of actions.
  - *ReAct:* The most common pattern, where the agent thinks "I need to find the order ID", acts by calling search_orders, observes the result "Found ID 555" and thinks again
  - *Multi-agent orchestration:* For very complex tasks, a manager agent might break the plan into subtasks and delegate them to a coder agent and a reviewer agent
- **The tools:** Tools are the hands of the agent. They're executable functions that the agent can invoke.
  - *Retrieval tools:* Vector search, internet search, wiki lookup
  - *Action tools:* Python REPL (for math and data) and API connectors (Salesforce, Jira and ServiceNow)
  - *Constraint:* Tools must be deterministic and governed. You don't want an agent hallucinating a parameter in an SQL query that drops a table.
- **Memory:** LLMs are stateless — they forget everything after the API call ends. The memory component provides persistence.
  - *Short-term memory:* The context window of the current conversation (e.g., "The user just mentioned their account number")
  - *Long-term memory:* A vector database or SQL log where the agent stores facts about the user or the business for future sessions (e.g., "This user prefers Python over SQL")

Model context protocol (MCP)

The biggest bottleneck for agents in 2024 was fragmentation. Every tool — Salesforce, Slack, Google Drive — required a custom connector for every agent framework.

MCP is the new open standard that solves this. Think of it as USB for AI agents. It provides a universal way to connect agents to data and tools.

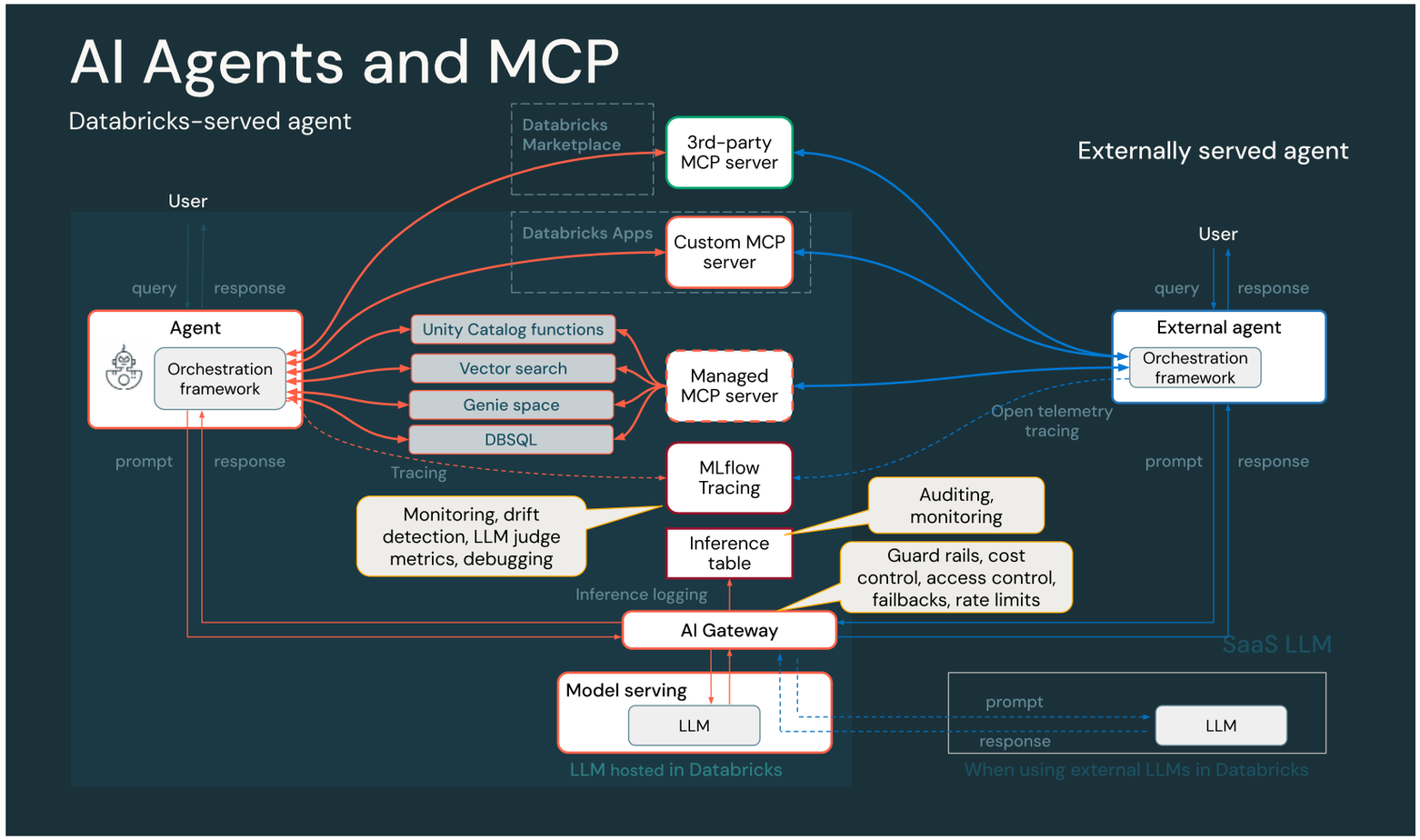


Figure 9: AI agents can now interact with tools through MCP server.

MCP defines three types of connections:

- **Managed MCP servers:** Built-in, fully governed connectors provided by the platform such as Vector Search and Unity Catalog functions. These are "plug and play."
- **External MCP servers:** Connectors to third-party SaaS tools such as Slack, Github and S&P Global. These can be installed from Databricks Marketplace.
- **Custom MCP servers:** Bespoke connectors you build for your proprietary internal APIs

Agent use cases

Financial Services

- Market intelligence for alpha generation
- Personalized offers for cross-sell / up-sell
- Financial crime detection and remediation
- Broker decision support to optimize premiums
- Customer-facing support agent

Healthcare and Life Sciences

- Single-cell models to identify drug targets
- Next best action to optimize provider engagement
- Clinical data abstraction to accelerate research
- Member conversation analytics and insights
- Claims denial appeals to increase reimbursement

Media and Entertainment

- Conversational AI for consumer engagement
- Contextualized ad targeting to improve ROAS
- Auto-generate localized content
- Autonomous network operations
- Anomaly detection to prevent fraud

Retail and Consumer Goods

- Optimized store-level product ordering
- Agentic commerce search
- Visual concept design
- Hyper-personalized offers at scale
- Smart inventory optimization

Manufacturing and Auto

- Imaging defect detection
- Proactive maintenance to increase uptime
- Prescriptive field support
- Process optimization to reduce energy use
- Supply chain simulator to mitigate risks (e.g., tariffs)

Energy and Utilities

- Predictive power outage prevention
- Asset reliability to reduce disruption
- Wildfire risk prevention
- Hyper-personalized energy savings recommendations
- Regulatory compliance copilot

AI agents are versatile solutions that can be applied across a wide range of business functions, offering substantial improvements in efficiency, accuracy and productivity. Below are some practical examples of how you can leverage AI agent systems in real-world scenarios:

- **Customer support automation:** Intelligent chatbots handle routine inquiries, provide accurate information with proper citations and escalate complex issues when necessary
- **Sales and marketing:** Agents automate lead qualification, generate personalized communications and analyze customer data for campaign optimization
- **Data analysis and reporting:** Agents collect, process and analyze data from multiple sources, generating insights and reports without manual intervention
- **Task automation:** Agents streamline scheduling, inventory management, order processing and workflow coordination across operations
- **Information extraction:** AI agents transform large volumes of documents into structured data by extracting pricing from contracts, organizing customer notes or capturing details from reports
- **Knowledge assistants:** RAG systems deliver contextually rich responses grounded in specific, authoritative sources for customer support and internal knowledge management
- **Multi-agent systems:** Several specialized agents collaborate to solve complex tasks. In these systems, agents might include planners, task executors and data retrieval specialists, each designed for a specific function. By orchestrating these agents, businesses can build highly adaptable AI solutions capable of handling intricate workflows in areas like supply chain optimization or multichannel customer engagement.

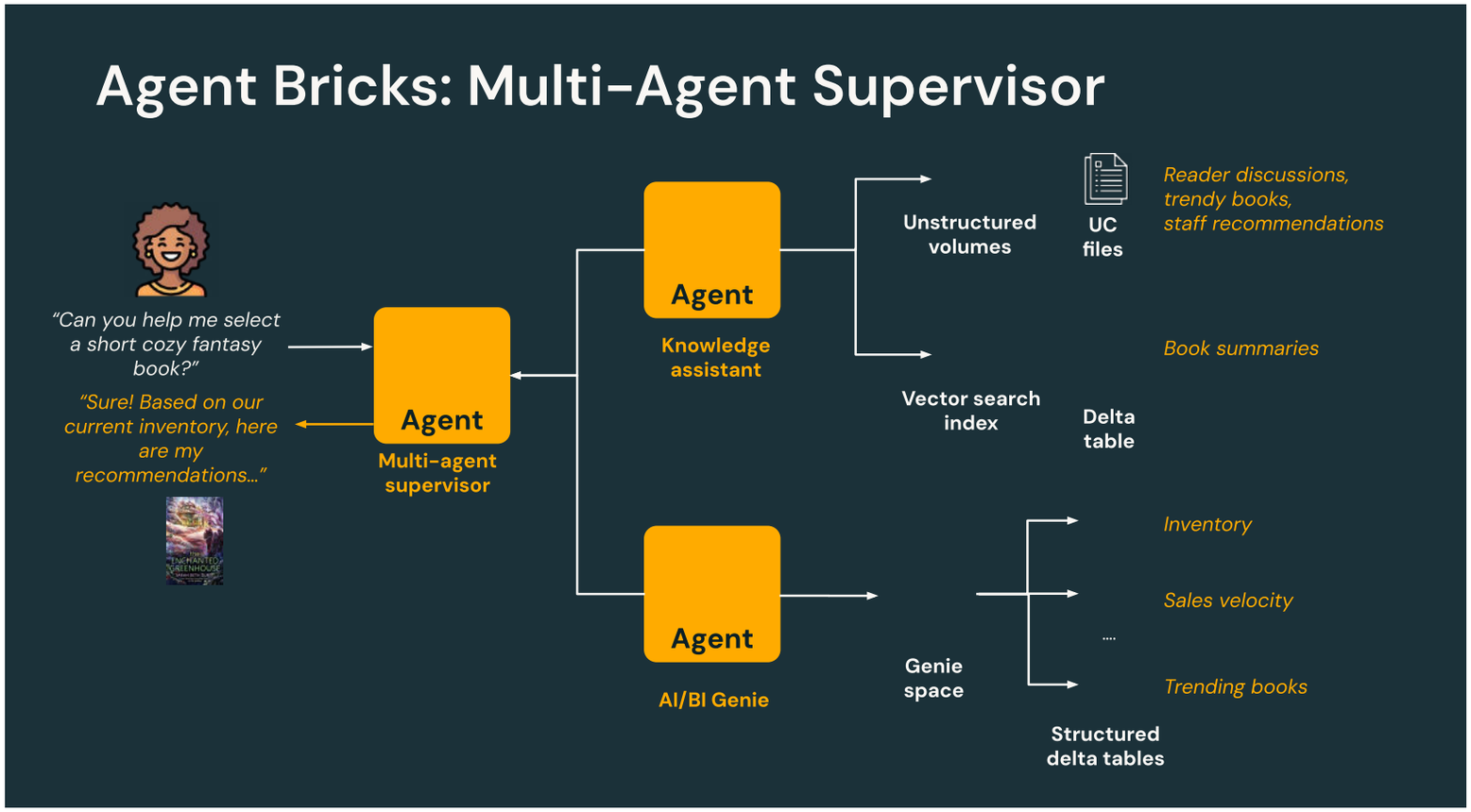
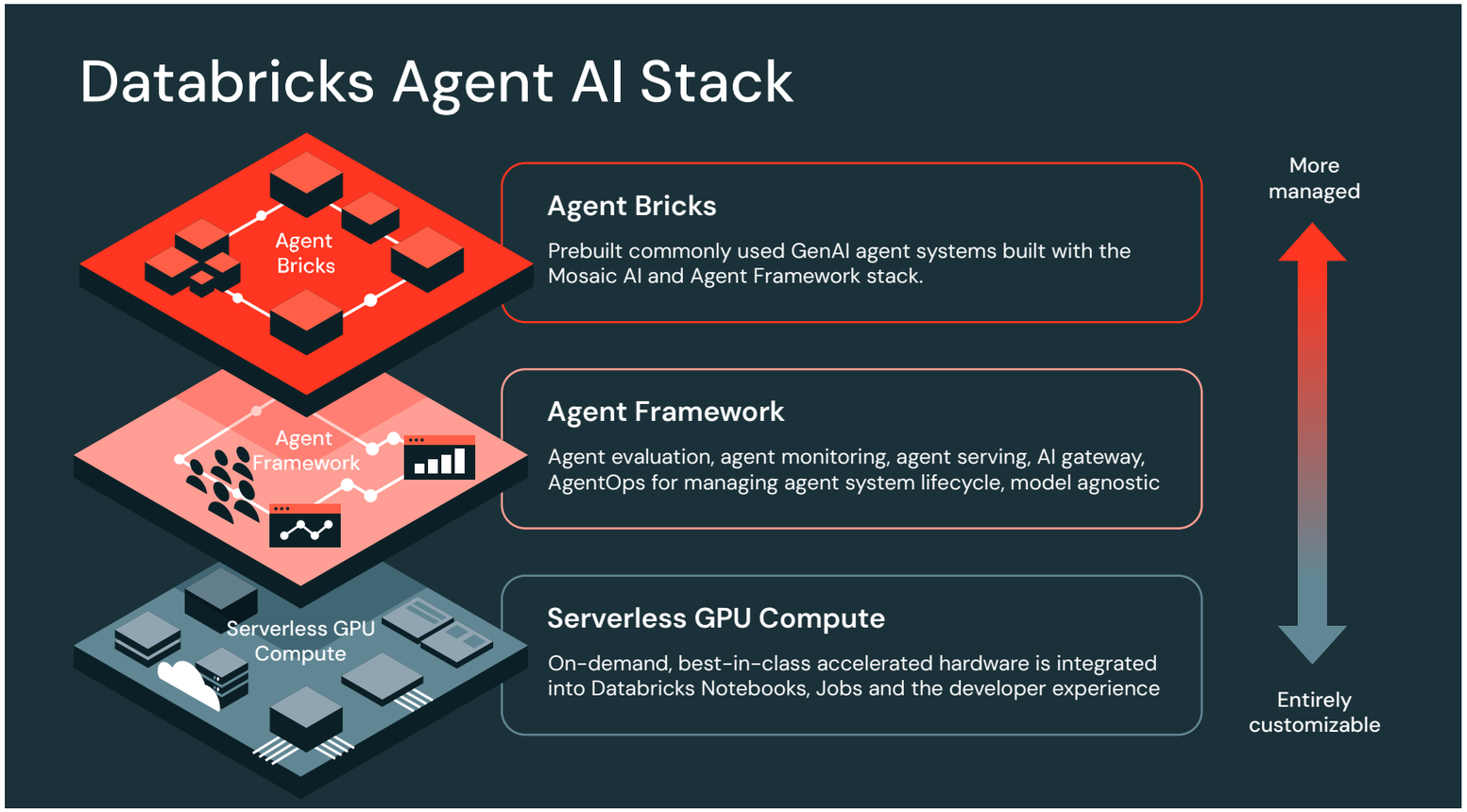


Figure 10: Agent Bricks deploys a multi-agent supervisor to do complex, multi-step processes.

Agent Bricks



The complexity of wiring these four components (LLM, planner, tools and memory) together is significant. To accelerate this, we categorize the stack into layers of abstraction. This stack allows you to choose the right level of control for your use case.

1. Agent Bricks (more managed)
   For common enterprise patterns, you shouldn't have to build from scratch. Agent Bricks provides pre-built "blueprints" for high-value use cases:
   - **Information extraction agent:** Specifically tuned to parse PDFs and structured forms
   - **Knowledge assistant:** A preconfigured RAG system with built-in memory and feedback loops
   - **Data analysis agent:** An agent capable of writing and executing SQL and Python to answer questions like "Why did churn increase last month?"
2. Databricks Agent Framework (flexible control)
   For teams building bespoke agents, the Databricks Agent Framework provides the SDKs for:
   - **Evaluation:** Running the OfficeQA benchmarks
   - **Monitoring:** Tracing execution with MLflow
   - **Governance:** Managing tool permissions via AI Gateway
3. Serverless GPU compute (entirely customizable)
   At the base, you have raw access to the latest accelerated hardware (H100s) to fine-tune open source models or host custom reasoning engines, ensuring you're never locked into a single model provider.

Third-party agents

Databricks recommends the MLflow interface ResponsesAgent to create production-grade agents. ResponsesAgent lets you build agents with any third-party framework, then integrate them with Agent Bricks features for robust logging, tracing, evaluation, deployment and monitoring capabilities.

The ResponsesAgent schema is compatible with the OpenAI Responses schema. To learn more about OpenAI Responses, see Migrate to the Responses API.

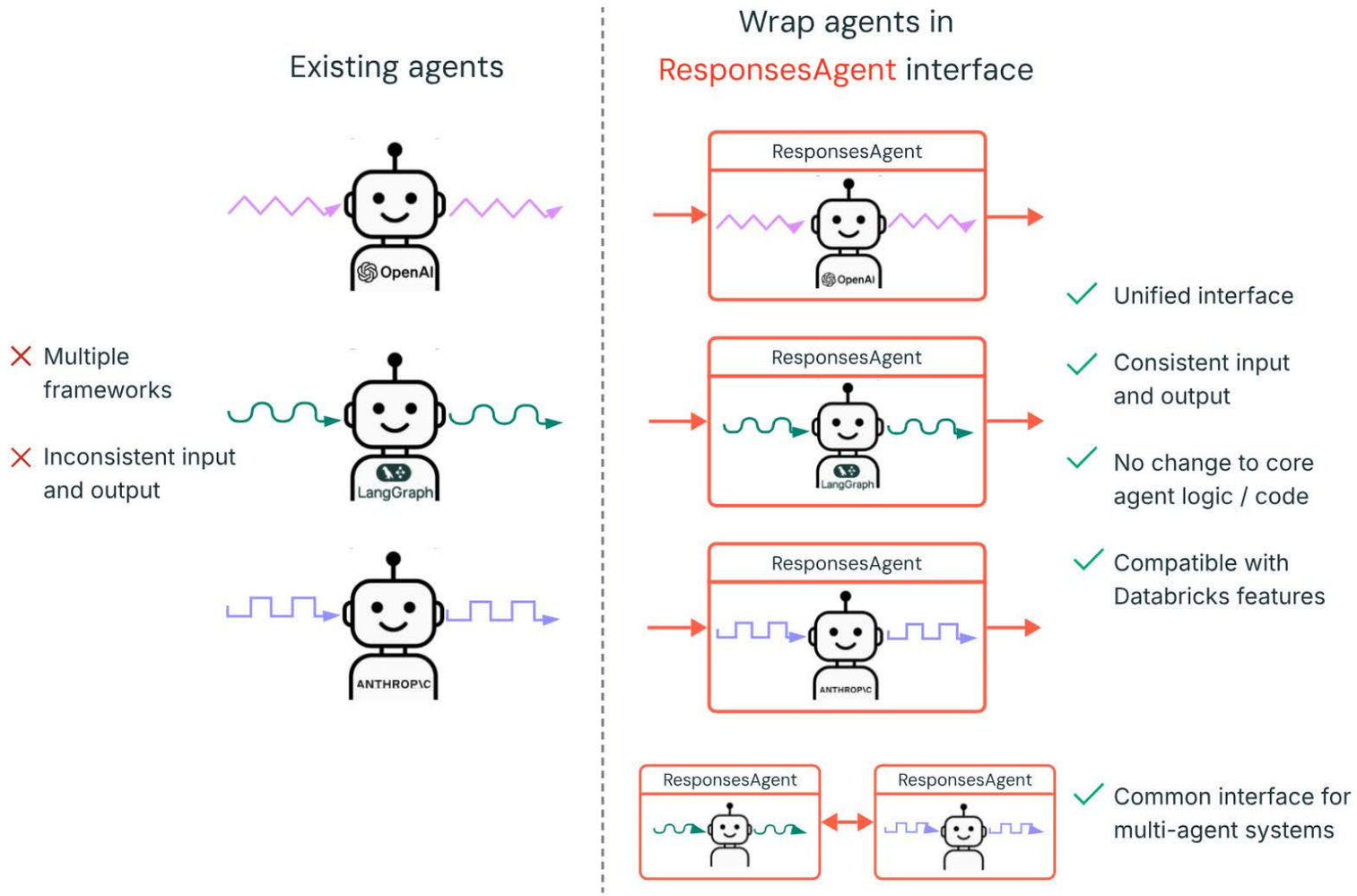


Figure 11: ResponsesAgent lets you build agents with any third-party framework.

`ResponsesAgent` provides the following benefits:

- Advanced agent capabilities
  - **Multi-agent support**
  - **Streaming output:** Stream the output in smaller chunks
  - **Comprehensive tool-calling message history:** Return multiple messages, including intermediate tool-calling messages, for improved quality and conversation management
  - **Tool-calling confirmation support**
  - **Long-running tool support**
- Streamlined development, deployment and monitoring
  - **Author agents using any framework:** Wrap any existing agent using the `ResponsesAgent` interface to get out-of-the-box compatibility with Databricks AI Playground, Agent Evaluation and Agent Monitoring
  - **Typed authoring interfaces:** Write agent code using typed Python classes, benefiting from an integrated development environment (IDE) and notebook autocomplete
  - **Automatic signature inference:** MLflow automatically infers `ResponsesAgent` signatures when logging an agent, simplifying registration and deployment. See Infer model signature during logging.
  - **Automatic tracing:** MLflow automatically traces your `predict` and `predict_stream` functions, aggregating streamed responses for easier evaluation and display
  - **AI Gateway-enhanced inference tables:** AI Gateway inference tables are automatically enabled for deployed agents, providing access to detailed request log metadata

To learn how to create a `ResponsesAgent`, see the examples in the following section and MLflow Documentation — ResponsesAgent for Model Serving.

In this chapter, we focused on understanding the components of building an agent to move away from the black box view of AI and treat it as a modular software stack. In the next chapter, we'll look at the fuel that powers this engine: data

Chapter 3: Working With Data

With GenAI, the model is often a commodity — the differentiator is the proprietary data you feed it.

While it's easy to send a text file to an LLM, the reality of the enterprise is that 80% of high-value knowledge is trapped in unstructured formats: PDF contracts, PowerPoint decks, HTML reports and scanned invoices. These documents contain complex layouts, tables and images that standard extraction tools fail to preserve, rendering the data useless for reasoning.

The data challenge

To build production-quality agents, data strategy must take precedence over model selection. A more powerful LLM with poorly parsed data will be outperformed by almost any other LLM that is fed relevant and perfectly structured context.

The challenge is threefold:

- **The unstructured trap:** Enterprise knowledge is rarely in JSON. It's in PDFs where tables are visual grids, not logical structures. Standard libraries can extract text but destroy the relationship between a header and a cell, causing the LLM to hallucinate associations.
- **Scale and latency:** Running optimal character recognition (OCR) and parsing on millions of documents requires massive compute orchestration. Doing this in single-threaded Python scripts is a nonstarter for production backfills.
- **Grounding and context:** Data isn't static. An agent needs to know who wrote the document, when it was last updated and who is allowed to access it.

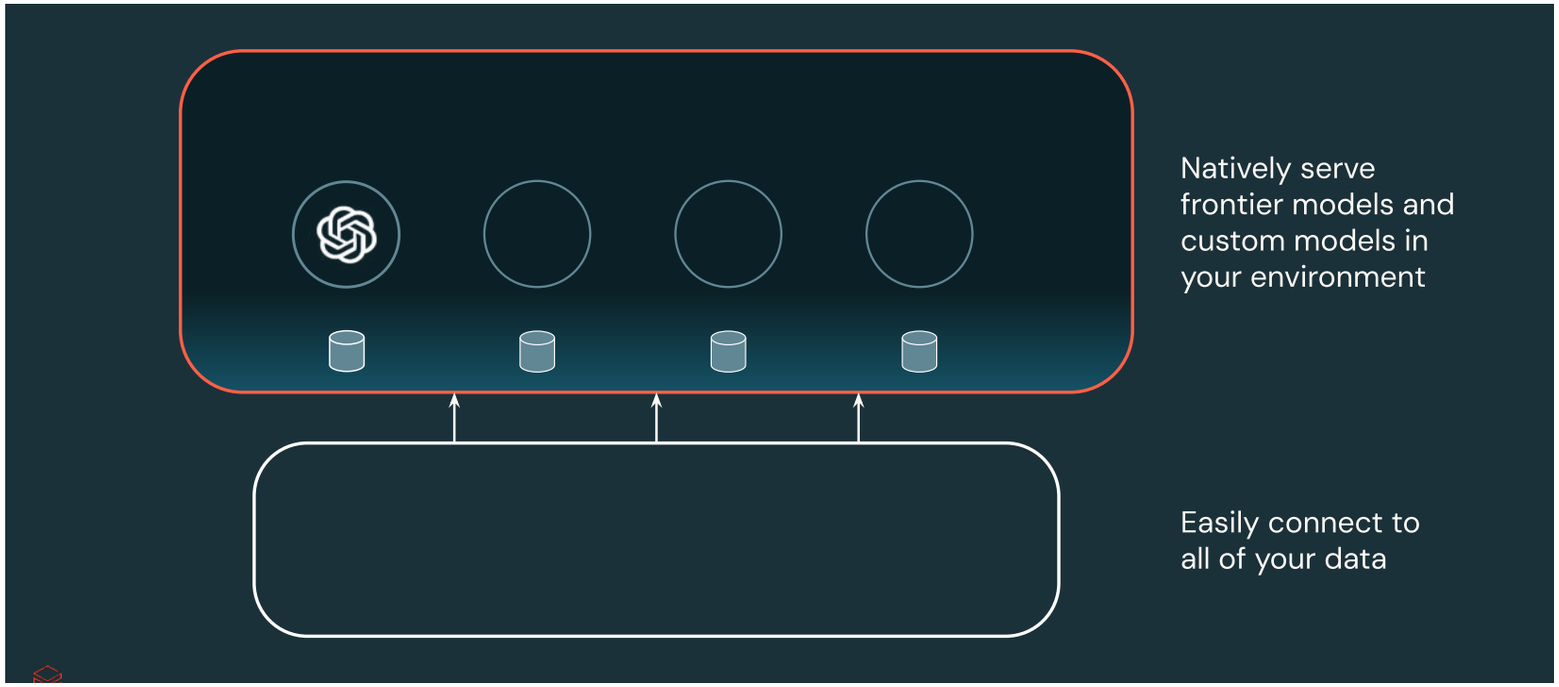


Figure 12: The Databricks Platform brings models to your data. Connectors feed natively served models, ensuring governance and reducing latency.

Document intelligence and parsing

The first step in any high-quality RAG or agent pipeline is document intelligence. This is the process of converting visual documents into a semantic format — typically Markdown — that LLMs can understand.

The problem with just extracting text

Traditional OCR reads left to right and top to bottom. In a multicolumn newsletter or a complex financial table, this approach merges unrelated text streams. When an LLM receives this jumbled context, it can't correctly answer questions like "What was the Q3 revenue for the EMEA region?" because the row and column alignment has been lost.

The solution: `ai_parse_document`

As detailed in the PDFs to production research, Databricks offers a native solution via the `ai_parse_document` function. This leverages vision-language models to "look" at the document, understand the layout and reconstruct it.

- **Layout awareness:** The `ai_parse_document` function detects columns, headers and footers, ensuring reading order is logical, not just linear
- **Table extraction:** This function converts visual tables into Markdown tables, preserving the semantic link between headers and values
- **Image processing:** The `ai_parse_document` function can extract images and generate captions, allowing agents to "read" charts and diagrams

Economics and scale

Running proprietary vision models on millions of pages is prohibitively expensive and slow. `ai_parse_document` is optimized for high-throughput batch processing within Lakeflow Declarative Pipelines or Apache Spark™ pipelines. Benchmarks indicate a 3–5x cost reduction compared to calling frontier model APIs for document understanding, with significantly lower latency.

Serverless batch inference with Agent Bricks AI SQL Functions

Not every GenAI use case requires a complex Python agent. For many analysts and data engineers, the goal is to apply LLM logic directly to rows of data in the warehouse. Agent Bricks **SQL Functions** brings the AI engine to the data, enabling high-scale batch inference without infrastructure management.

These functions allow you to invoke LLMs directly within SQL queries (or Python DataFrames), making GenAI declarative, scalable and governed by existing table access control lists (ACLs).

Production-grade batch processing

Recent updates have transformed Agent Bricks AI Functions into a fully serverless batch inference engine, addressing the need for speed and simplicity in production pipelines:

- **Instant, serverless AI:** Functions like `ai_query` require no endpoint setup or provisioning. You can run inference on millions of rows instantly, with the system automatically managing resource scaling behind the scenes.
- **10x performance:** The backend — Foundation Model API (FMAPI) — has been optimized for batch workloads, delivering over 10x faster performance compared to previous versions, making it practical for massive datasets
- **Structured outputs:** Instead of relying on brittle prompt engineering to get clean data, `ai_query` now supports a `responseFormat` parameter. This allows you to define a specific schema (e.g., `STRUCT<title:STRING, keywords:ARRAY<STRING>>`), ensuring the model extracts structured insights reliably every time.

The core function suite

FUNCTION	DESCRIPTION	USE CASE
`ai_query`	Invokes an LLM endpoint with a prompt. Now supports **structured output** for strictly formatted responses.	"Extract the title and authors from this paper as a JSON struct"
`ai_extract`	Extracts specific entities into a structured schema.	Pulling invoice numbers and dates from email bodies
`ai_classify`	Categorizes text into defined labels.	Tagging support tickets as "Urgent", "Refund" or "Tech"
`ai_summarize`	Generates a concise summary of long text.	Summarizing call transcripts or reviews
`ai_gen`	Generates synthetic data or content.	Creating sample data for testing

Example: Classifying customer reviews at scale

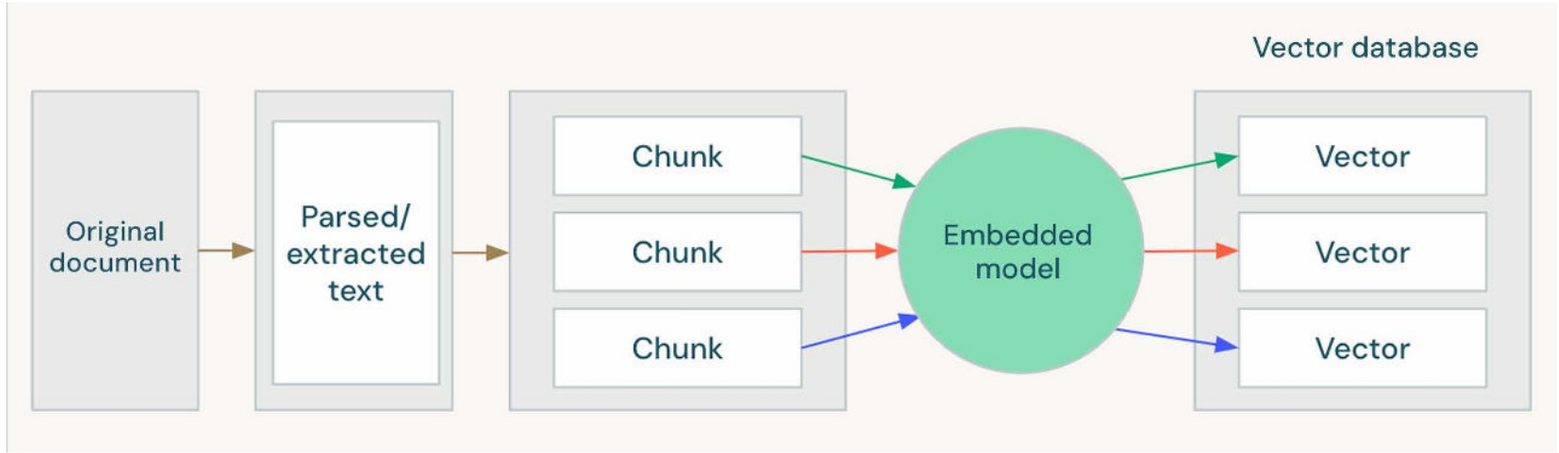
Instead of writing a Python script to iterate through a dataframe, a data engineer can classify millions of reviews in a single query:

SQL

```sql
SELECT
  review_id,
  review_text,
  ai_classify(
    review_text,
    ARRAY('positive', 'negative', 'neutral')
  ) as sentiment
FROM
  customer_reviews
```

Retrieval augmented generation (RAG)

RAG creates a bridge between a model's reasoning capabilities and your organization's memory. However, a naive chunk-and-retrieve pipeline often fails in production due to poor data parsing or a lack of semantic depth. Building a reliable retrieval engine requires mastering three layers of complexity: parsing, hybrid search and advanced retrieval patterns.



THE RAG DATA PIPELINE

This is a production-grade pipeline that consists of four distinct engineering steps:

- **Ingestion and parsing:** Use `ai_parse_document` to convert unstructured files like PDFs and images into clean, layout-aware Markdown
- **Chunking:** Split the Markdown into smaller segments (typically 512 or 1024 tokens)
  - *Best practice:* Use recursive character splitting that respects Markdown headers. This ensures that a section header stays attached to its paragraphs, preserving semantic context.
- **Embedding:** Convert text chunks into vector embeddings using models like `gte-large` or `bge-m3`
- **Indexing:** These vectors are stored in **Databricks Vector Search**, a serverless vector database that syncs automatically with your Delta tables

OPTIMIZING VECTOR SEARCH: HYBRID IMPLEMENTATION

Standard vector search — finding the nearest neighbor in semantic space — is powerful but imperfect. It struggles with exact matches, such as part numbers (e.g., "Part #XJ-900") or specific acronyms, because these often lack semantic depth.

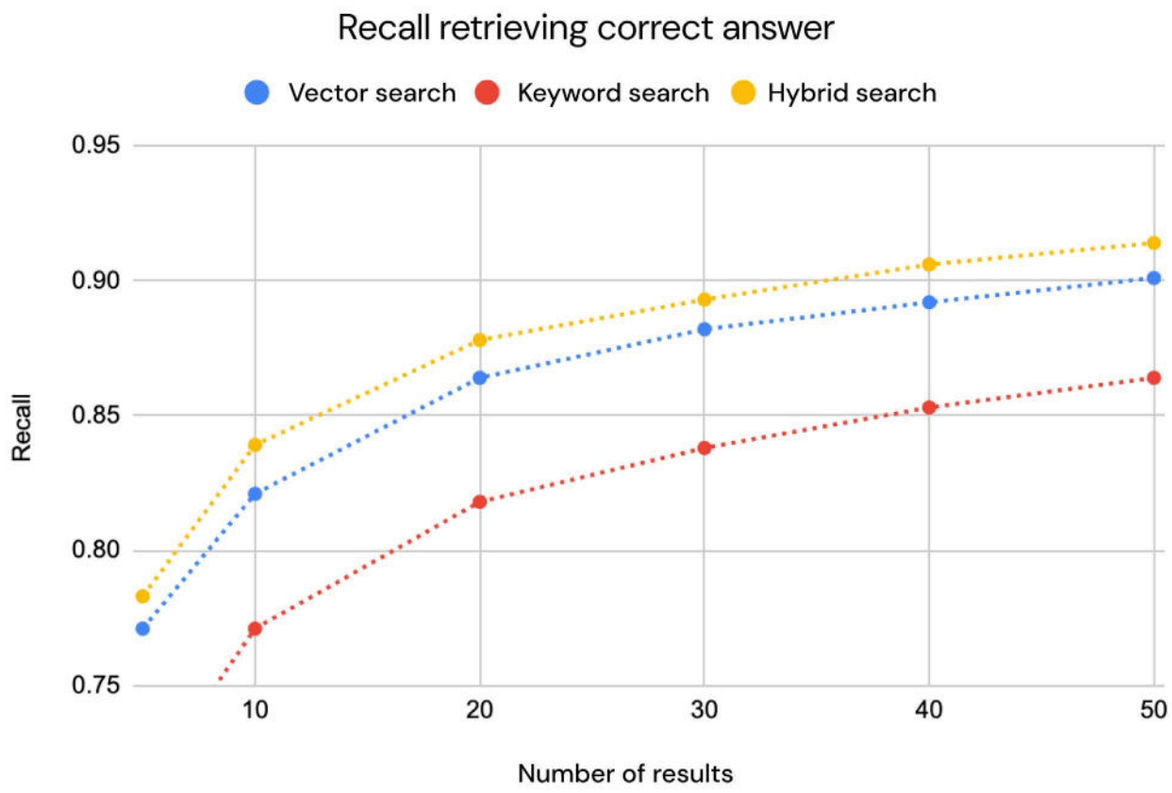


Figure 13: A plot of recall at various values of the number of results retrieved.

Our implementation of hybrid search is based on reciprocal rank fusion (RRF).

To solve this, we use **hybrid search**, which combines semantic search (vector) with keyword search (lexical).

- **How it works:** Our implementation uses **RRF**. It runs both a vector search and a keyword search in parallel, normalizes the scores (0.0 to 1.0) and fuses them into a single ranked list
- **The result:** A query for "Reset procedure for XJ-900" finds both the *concept* of resetting (semantic) and the *exact match* for the part number (keyword)
- **Performance:** Internal benchmarks show a 20% improvement in recall, reducing the number of documents required for high recall from 50 to 40 in typical datasets

ADVANCED RETRIEVAL TECHNIQUES

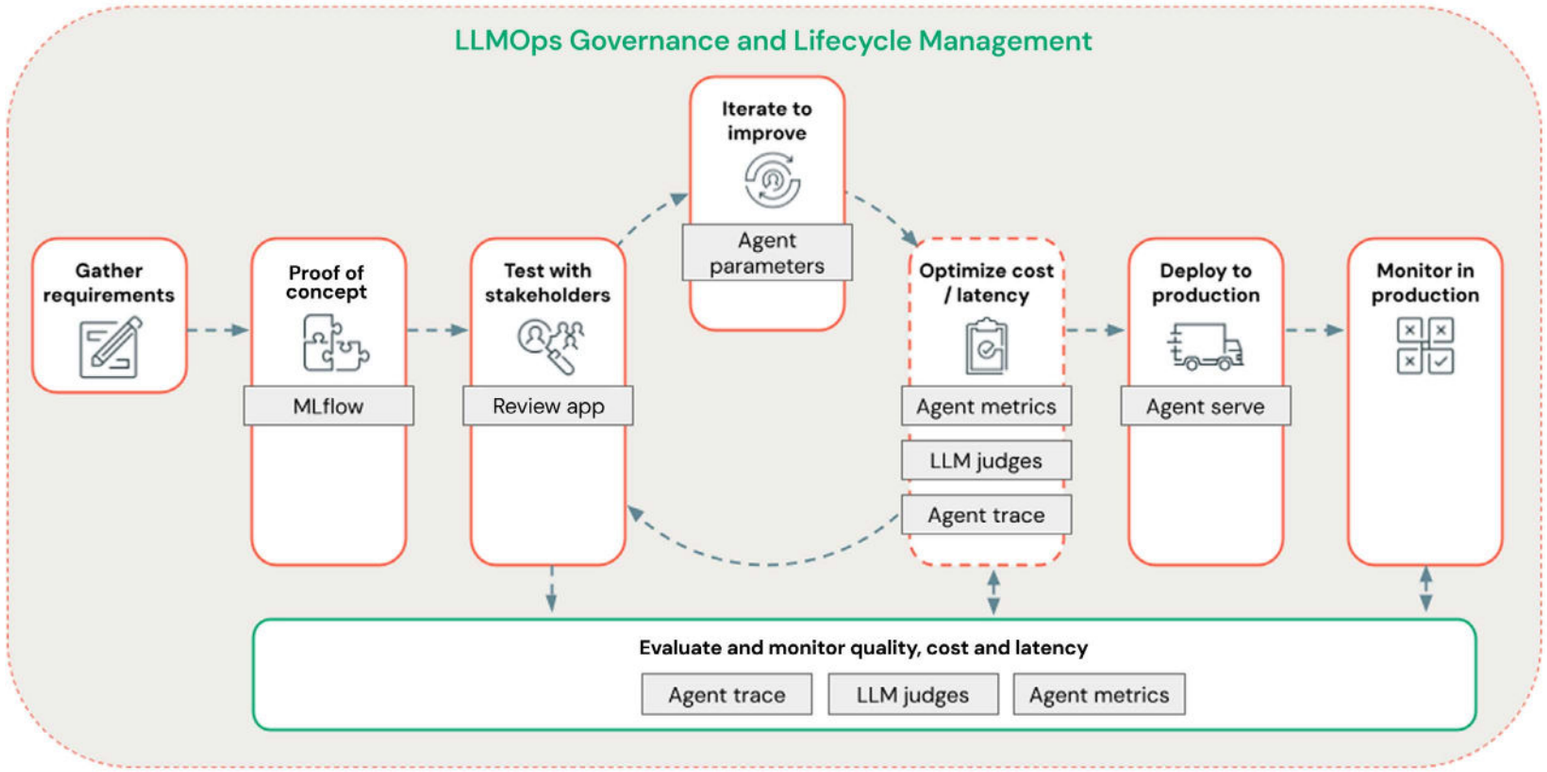
Beyond standard search, production systems often require additional logic to increase precision and governance.

- **Metadata filtering:** Applying pre-filters (e.g., `WHERE department = 'Finance'`) *before* the vector search runs. This ensures the agent only retrieves documents the specific user is authorized to see.
- **Query rewriting:** Using an LLM to refine the user's prompt, such as turning "Show me the last one" into "Show me the sales report for Q4 2024," before executing the search
- **Reranking:** Fetching a larger number of chunks (e.g., 50) and using a specialized cross-encoder model to rescore them, passing only the five most relevant chunks to the LLM
- **Contextual chunking:** Retrieving adjacent chunks (the paragraph immediately preceding and following the match) to give the LLM better situational awareness of the discussion flow

For deeper implementation details, consult the Databricks documentation on similarity search calculation details and the Python SDK for `similarity_search`.

Chapter 4: GenAI Performance and Evaluation

Achieving production-grade GenAI applications requires a complete lifecycle that can be broken down into two primary mechanisms for improving quality: **automated evaluation** using LLM judges and **direct adaptation** using human-in-the-loop feedback.



Automated evaluation and optimization (LLM Judges)

This category focuses on using automated models — **LLM judges** — to measure quality, cost and latency, and to optimize the agent using model-generated signals rather than human labels.

The evaluation framework: MLflow concepts and data model

MLflow is the open platform that unifies tracking, evaluation and observability for GenAI apps. It organizes all GenAI application data within **experiments**.

DATA MODEL CATEGORY	ENTITY	DESCRIPTION
Observability data	**Traces**	Capture the complete execution, including inputs, outputs and every intermediate **span** (LLM calls, tool use) of your application.
	Assessments	Quality measurements attached to a trace, consisting of **feedback** (judgments from users/scorers) and **expectations** (ground truth labels).
Evaluation data	**Evaluation datasets**	Curated, versioned collections of test cases used for systematic quality testing.
	Evaluation runs	The aggregated results of testing an application version against a dataset.
Human labeling data	**Labeling sessions**	Queues of traces are sent to domain experts for human review via the review app.

Scorers vs. LLM judges

MLflow uses both judges and scorers to automate quality assessment:

- **LLM judges:** These are callable SDKs that evaluate text against specific criteria such as correctness and relevance
- **Scorers:** These are functions that act as adapters. They extract the relevant data — request, response, context — from a **trace** and pass it to the judge or custom code for evaluation, returning the result as a feedback **assessment**. The same scorers work in both development and production.

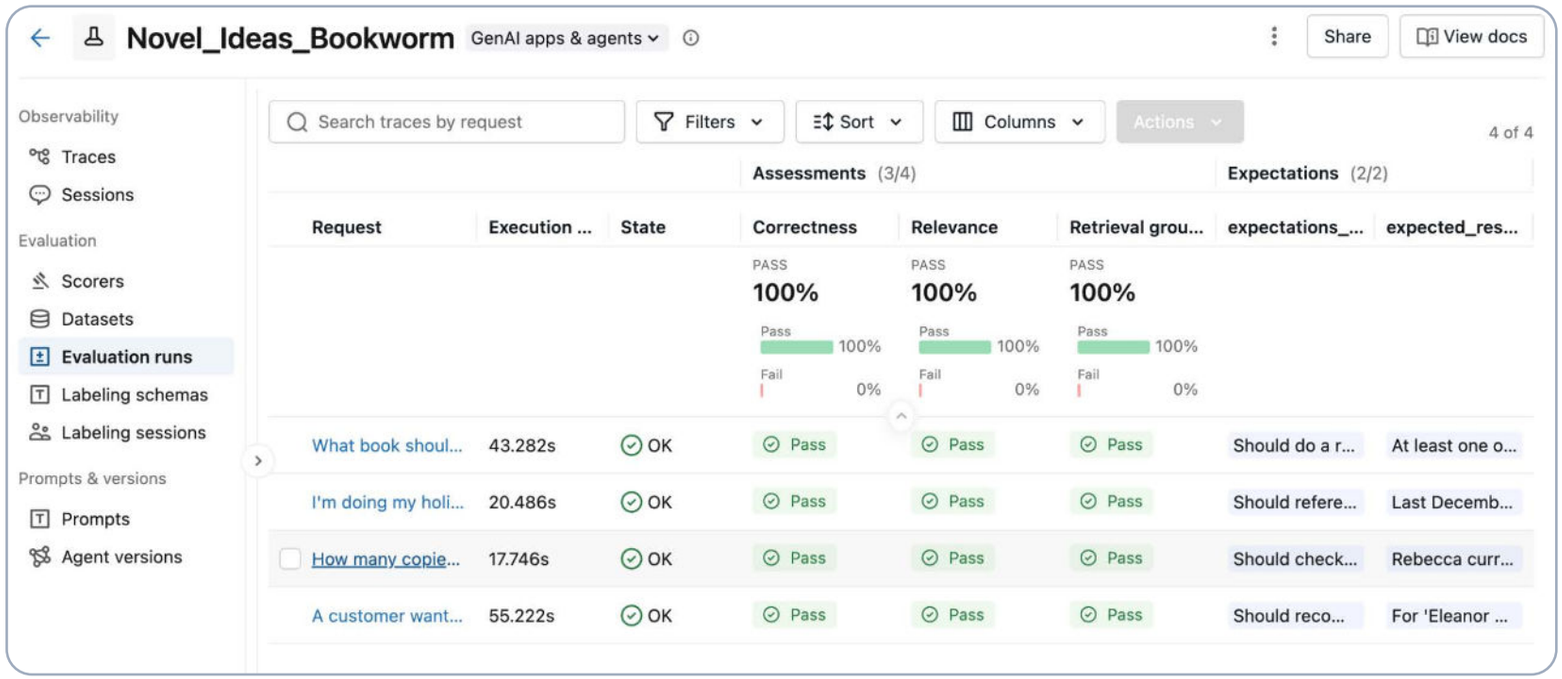


Figure 14: Quality feedback assessment.

Agent evaluation and tracing

MLflow tracing provides the foundational observability required for complex agentic systems:

- **Trace visibility:** Tracing captures detailed execution information, logging **inputs, outputs and metadata** associated with each intermediate step, or **span**, of the request
- **Observability and debugging:** Tracing allows developers to analyze execution **paths**, inspect **span** details and use interactive trace visualizations in the MLflow UI to diagnose issues and identify root causes

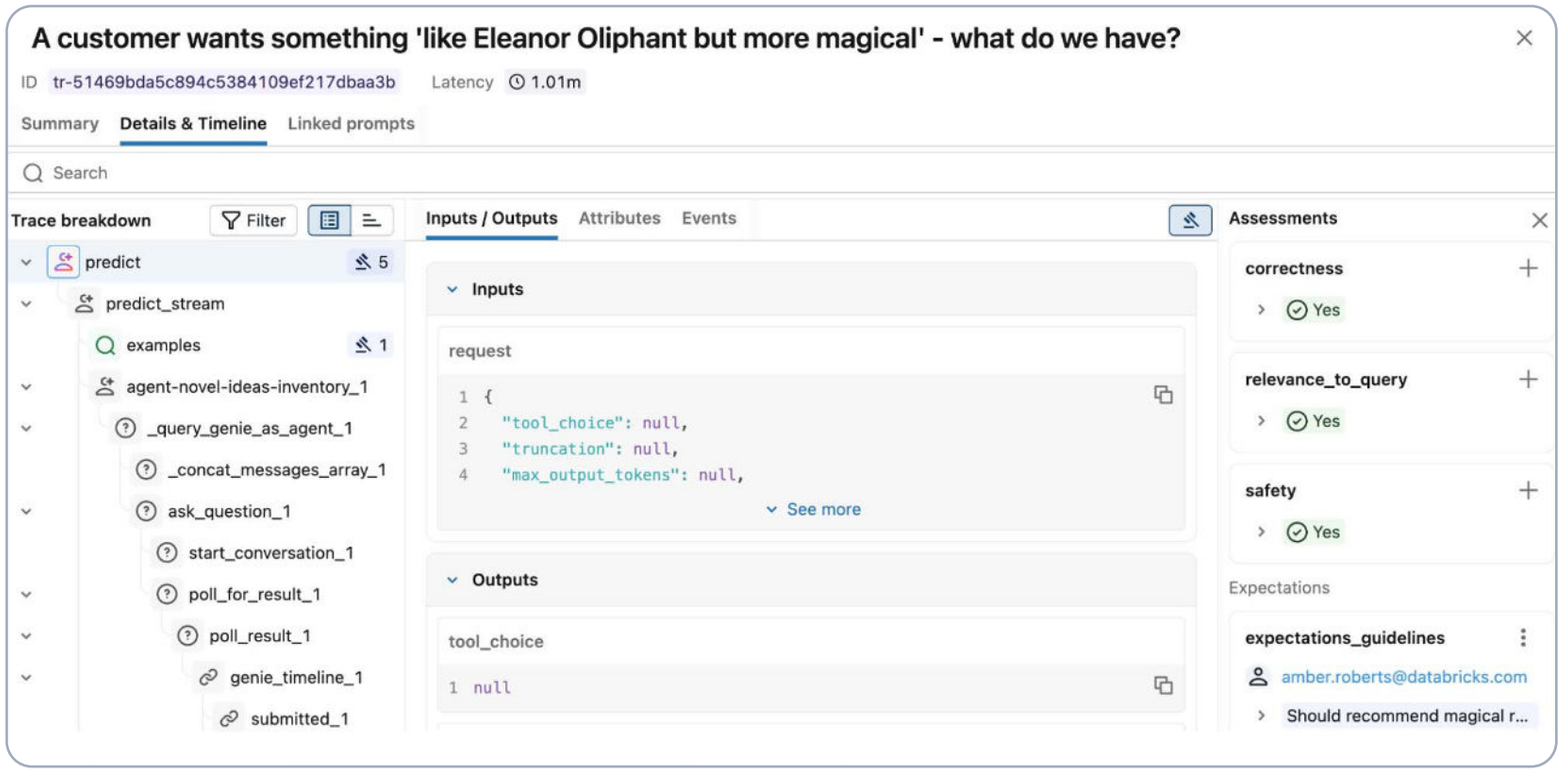


Figure 15: MLflow tracing provides observability.

Optimization path: Test-time adaptive optimization (TAO)

TAO is a model tuning technique that uses LLM judges, such as reward models, to provide the necessary scoring signal for optimization, eliminating the need for extensive human-labeled data.

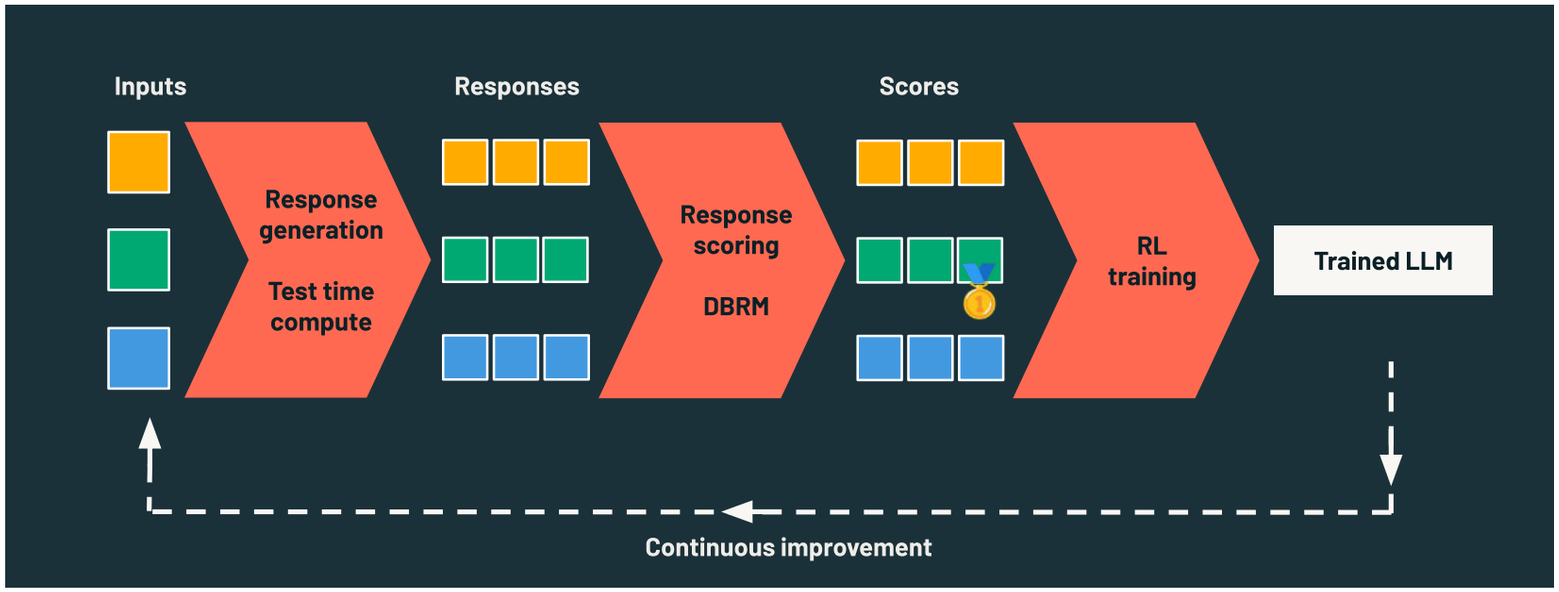


Figure 16: The TAO pipeline.

- **Mechanism:** TAO uses **test-time compute** combined with reinforcement learning (RL) during the tuning phase
- **Process:** Prompts collected from production logs are generated and scored by a reward model — like a database request module (**DBRM**) or custom judges. The base model is updated via RL.
- **Efficiency:** The final optimized model retains the **low latency and low cost** of the original base model while achieving quality that rivals expensive proprietary models

Direct adaptation and refinement (human-in-the-loop feedback)

This category focuses on integrating high-value human expertise for defining quality standards and directly teaching the agent.

The continuous improvement cycle

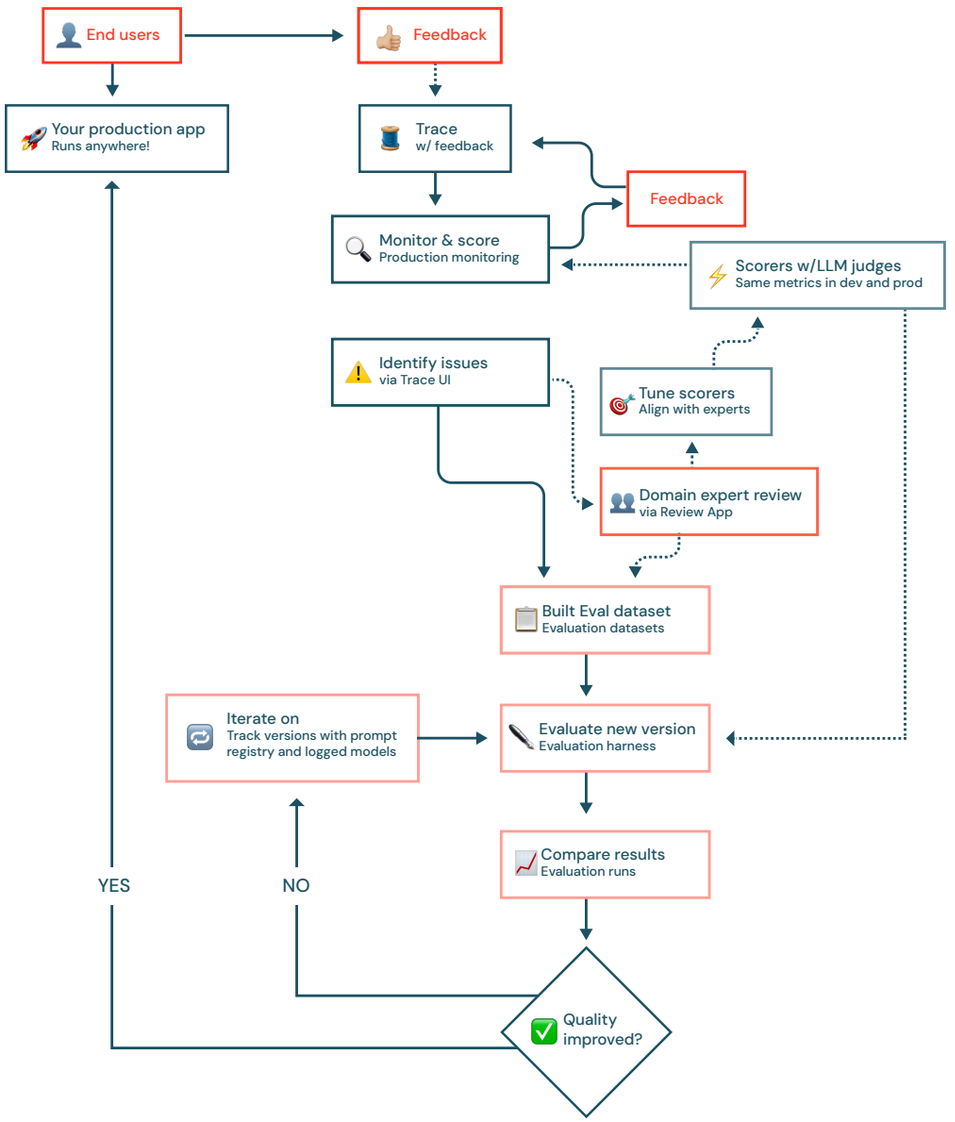


Figure 17: The **continuous** improvement cycle.

MLflow orchestrates a systematic cycle that incorporates both user feedback and domain expert judgment to continuously improve quality.

1. **Production app:** The deployed app serves users and generates **traces**
2. **User feedback:** End users provide feedback — such as a thumbs-up or thumbs-down — that is attached to the trace
3. **Monitor and score: Production monitoring** automatically runs LLM judge–based **scorers** on traces to assess quality
4. **Identify issues:** Teams use the **trace UI** to find patterns in low-scoring traces
5. **Domain expert review:** Optionally, a sample of traces is sent to experts via the **review app** for detailed labeling
6. **Build eval dataset:** Problematic and high-quality traces are curated into **evaluation datasets**
7. **Tune scorers:** Expert feedback aligns scorers and judges with human judgment
8. **Evaluate new versions:** The **evaluation harness** tests improved app versions against the **evaluation datasets**, using the same scorers applied in production monitoring
9. **Deploy or iterate:** Based on **evaluation run** results, the new version is deployed, or the cycle continues

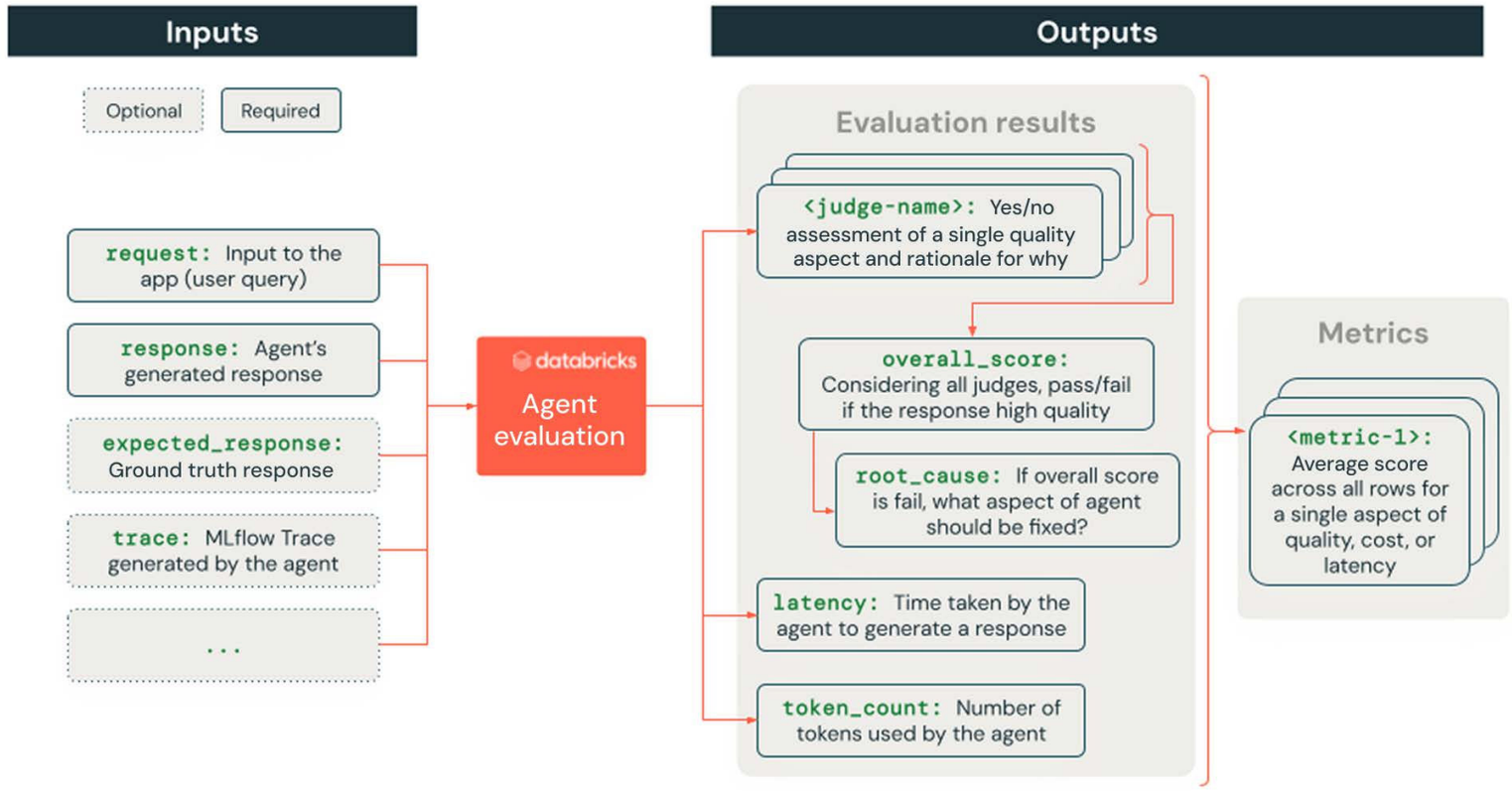


Figure 18: Cycle that incorporates both user feedback and domain expert judgment to continuously improve quality.

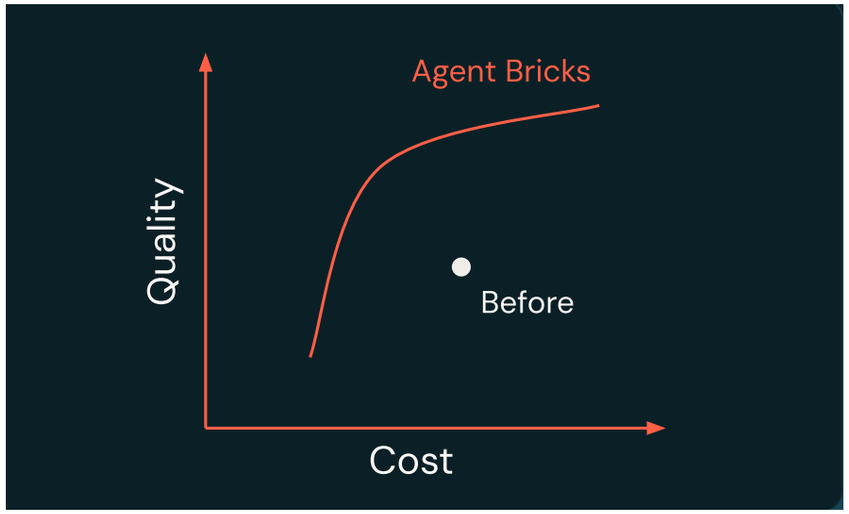
Codifying expertise: Custom judge calibration

- **Judge portfolio design:** Quality requirements are decomposed into precise, measurable dimensions (**top-down** regulatory rules and **bottom-up** observed failure modes)
- **SME workflow:** SMEs label a small, targeted set of edge cases. The **judge builder** uses this human-labeled data to train the LLM judge prompt to reliably align with expert consensus.
- **Judge builder:** A tool that streamlines the process of aligning custom LLM judges with human feedback, ensuring technical accuracy and consistent governance

Accelerating evaluation with Agent Bricks

Databricks Agent Bricks streamlines the evaluation process by automating the creation of judges and datasets, accelerating the continuous improvement cycle described in Figure 18 above.

- **Synthetic data generation:** Agent Bricks includes an SDK that generates high-quality synthetic questions and answers. These datasets provide a robust foundation for testing diverse scenarios and can be reviewed by subject-matter experts to ensure relevance.
- **Automated and custom judges:** When using Agent Bricks, LLM judges are often created automatically to measure core quality metrics. For specific business needs, it offers robust customization options — including custom judge creation, few-shot examples and selective application — empowering organizations to perform precise assessments.
- **Optimization with TAO:** Agent Bricks automatically leverages **TAO** in the background to optimize agent quality, cost and speed, integrating these evaluation tools directly into the optimization loop



Continuous adaptation: Agent learning from human feedback (ALHF)

ALHF is a paradigm that allows the agent to learn and adapt based on minimal, high-signal **natural language feedback** from experts. This overcomes the limitations of static labels and simple numeric rewards.

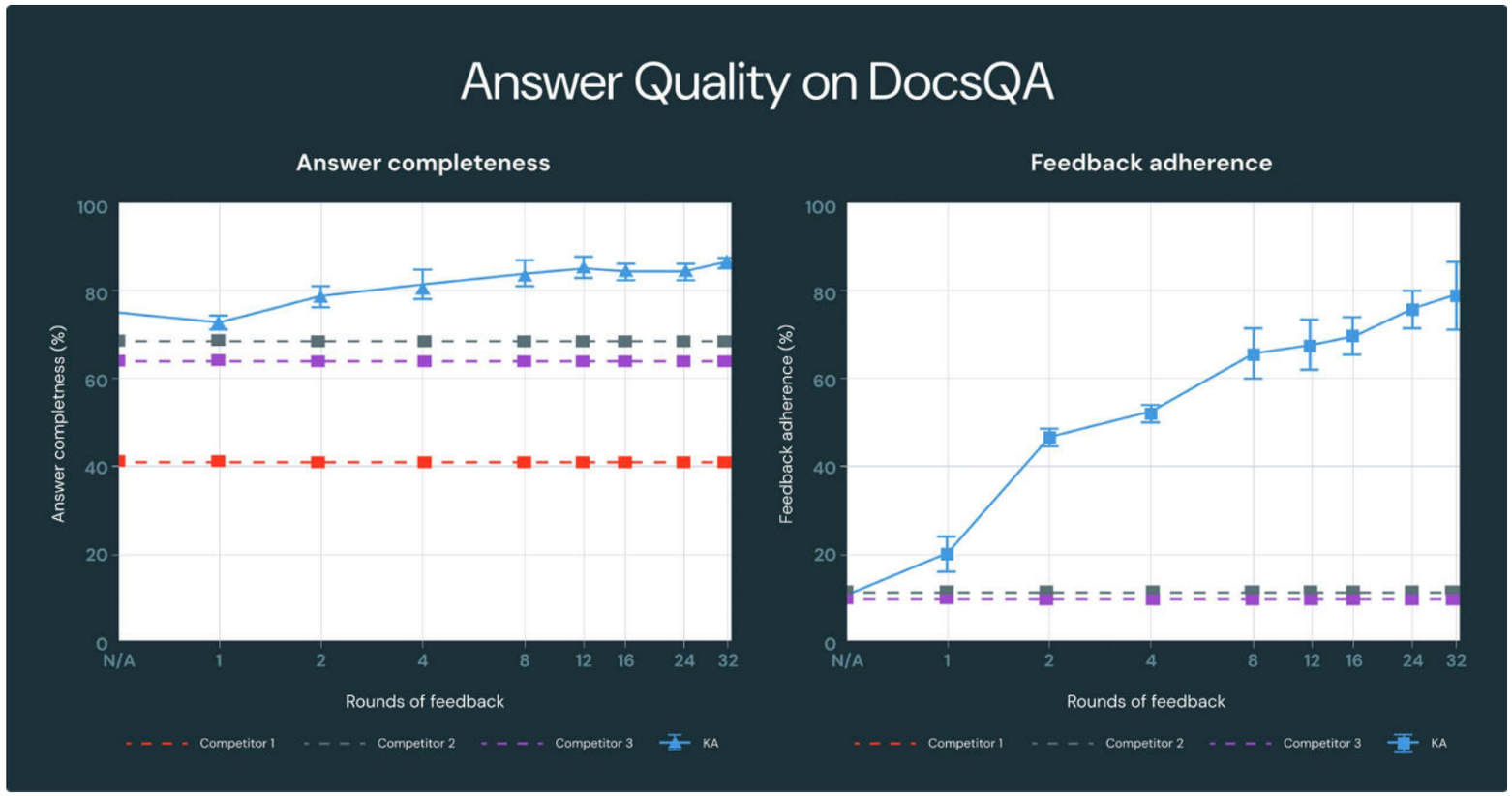


Figure 19: In contrast to static baselines, the Agent Bricks Knowledge Assistant (ABKA) improves its response with increasing amounts of feedback in terms of both answer completeness and feedback adherence. Results are reported on unseen questions from the DocsQA dataset.

ALHF in practice

- **Mechanism:** An expert provides specific, natural language feedback (e.g., "The answer must use PostgreSQL syntax") rather than a simple numeric score
- **Impact:** As shown in the **Agent Bricks: Knowledge Assistant (KA)** case study, ALHF dramatically boosts quality. With as few as **32 feedback records**, the **feedback** adherence score jumped from 11.7% to nearly 80%, showcasing high sample efficiency.
- **Technical solution:** The agent uses an internal memory mechanism to **scope** the feedback (determining which future questions it applies to) and routes it to the correct **components** for adaptation

Chapter 5: Governance for GenAI and Agents

As enterprises rapidly adopt GenAI solutions, ensuring robust security, compliance and operational efficiency becomes paramount. While GenAI offers transformative potential, it also introduces significant risks — from data privacy concerns to model integrity challenges and escalating operational costs.

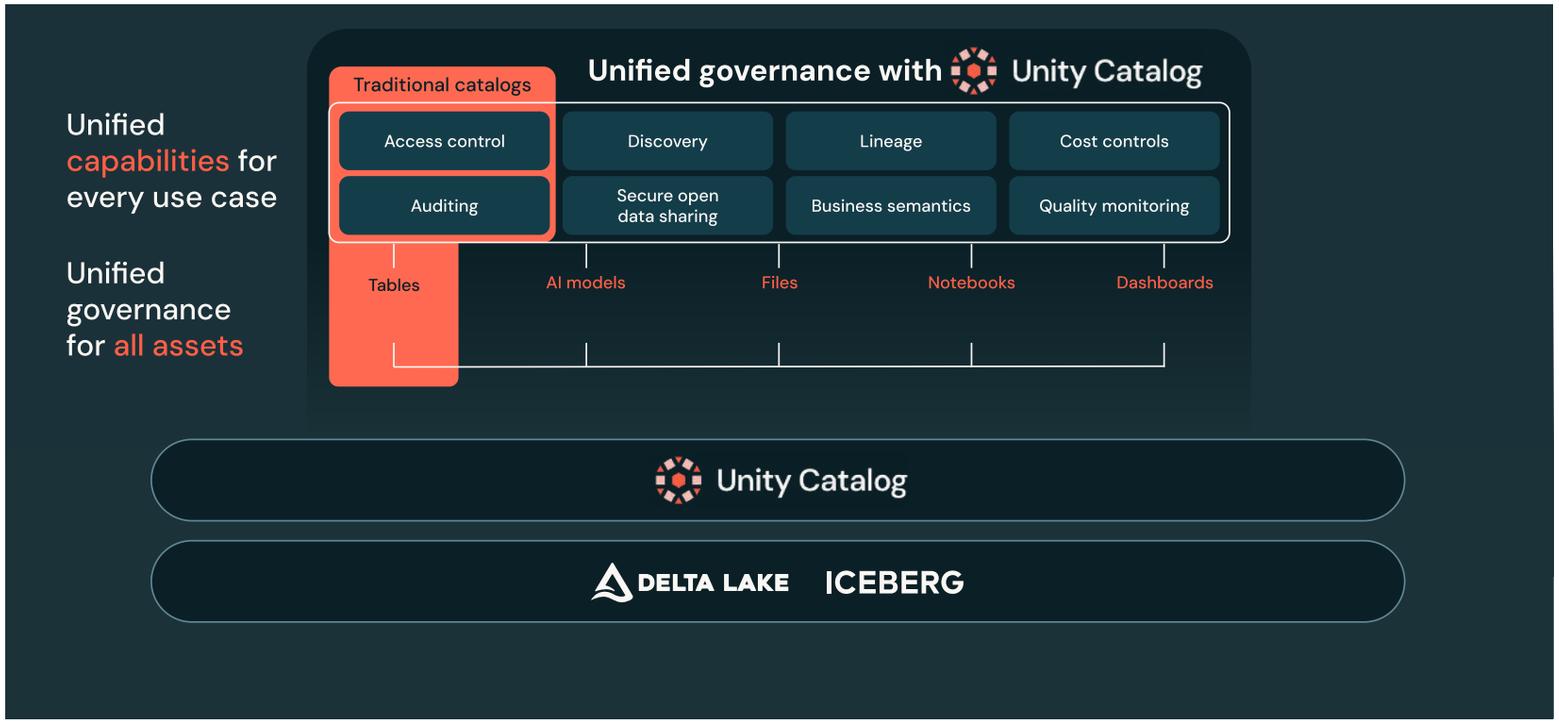


Figure 20: Unified governance over your data and AI assets with Databricks Unity Catalog.

Effective governance is essential for deploying LLMs at scale. **Databricks Agent Bricks**, reinforced by **Unity Catalog**, provides centralized control over data, models and AI endpoints. This ensures high security standards without compromising performance, providing a unified platform with enterprise-grade security, compliance controls and advanced monitoring.

Unity Catalog for GenAI: The unified governance layer

In the past, organizations managed data permissions in one system and API keys in another. Unity Catalog unifies this by treating AI assets — models, functions and tools — as first-class citizens alongside tables and files.

Governing the MCP ecosystem

As discussed in Chapter 2, the MCP standardizes how agents connect to tools. However, connecting to tools at scale creates a sprawl problem. Databricks integrates different types of MCP servers directly into the Unity Catalog governance model:

- **Discovery:** The **MCP catalog** acts as an internal app store, allowing developers to discover approved tools, such as a company knowledge retriever, and instantly connect them — but only if they have the correct permissions
- **Trust:** For external tools, such as Salesforce or S&P Global, administrators can control which third-party vendors are allowed to run within the environment via Databricks Marketplace, preventing shadow IT

Governing agent identity: On-behalf-of (OBO) authentication

When an agent performs an action, *who* is performing it? In many systems, agents run as a generic service account with broad privileges, creating a security risk.

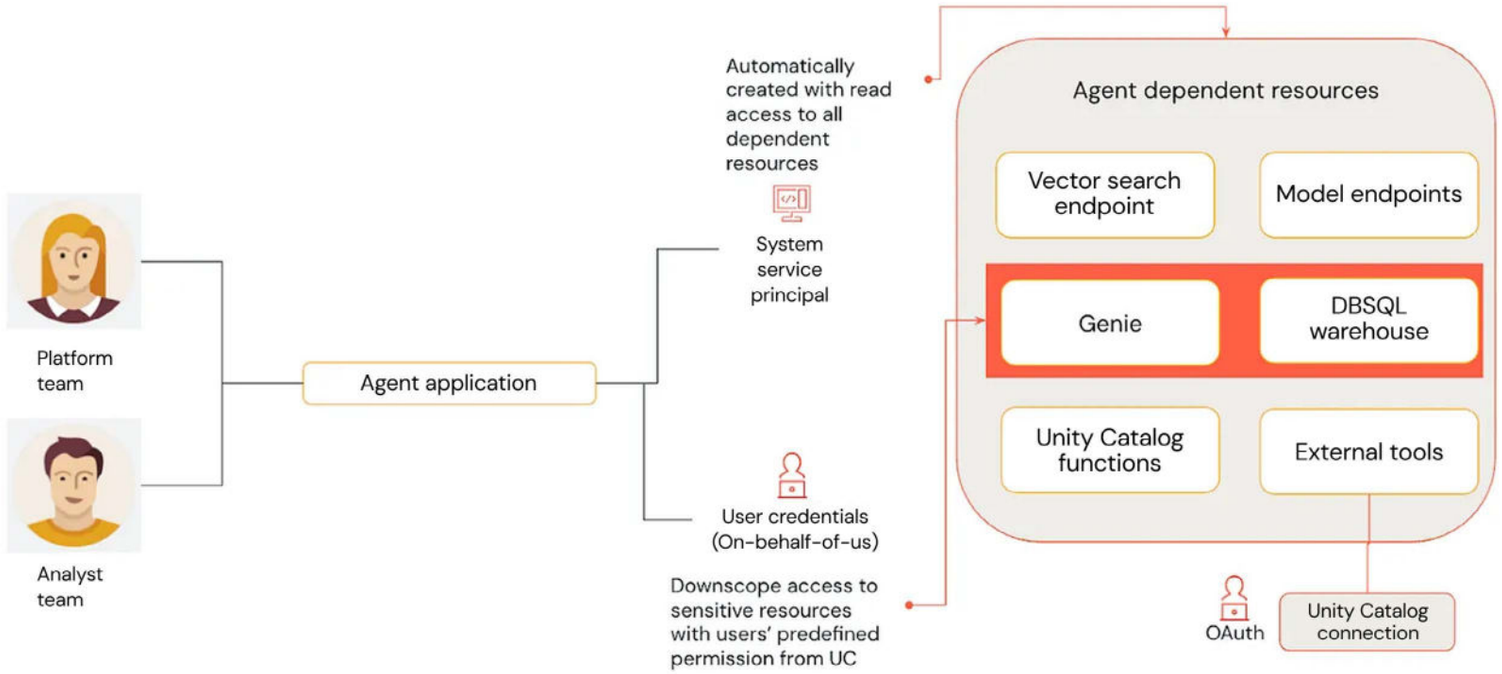


Figure 21: Agents run tasks on behalf of different users, requiring different permissions.

Databricks solves this with **OBO authentication**. When a user interacts with an agent, the agent executes tools using the user's specific identity and permissions. This enables **attribute-based access control (ABAC)**, where security policies dynamically filter data based on context — ensuring a sales agent only retrieves revenue data matching the user's region.

Model Serving: The unified interface

Before you can govern traffic, you must standardize it. In many organizations, AI is fragmented: one pipeline for OpenAI, another for open source Llama and a third for custom agents. This leads to bespoke connections and heavy code rewrites.

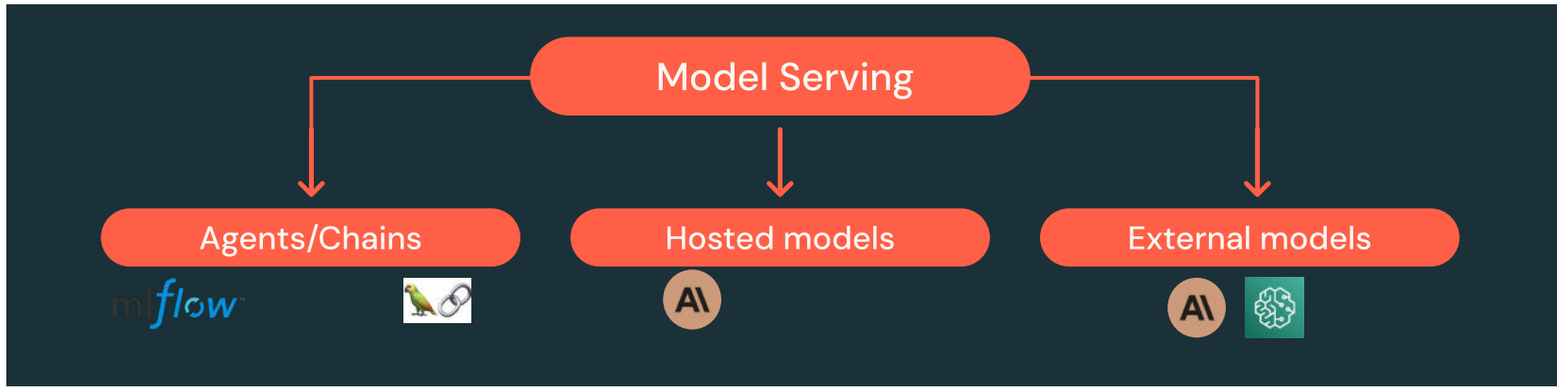


Figure 22: Model Serving supports all AI frontier AI models, as well as agents, chains and external models.

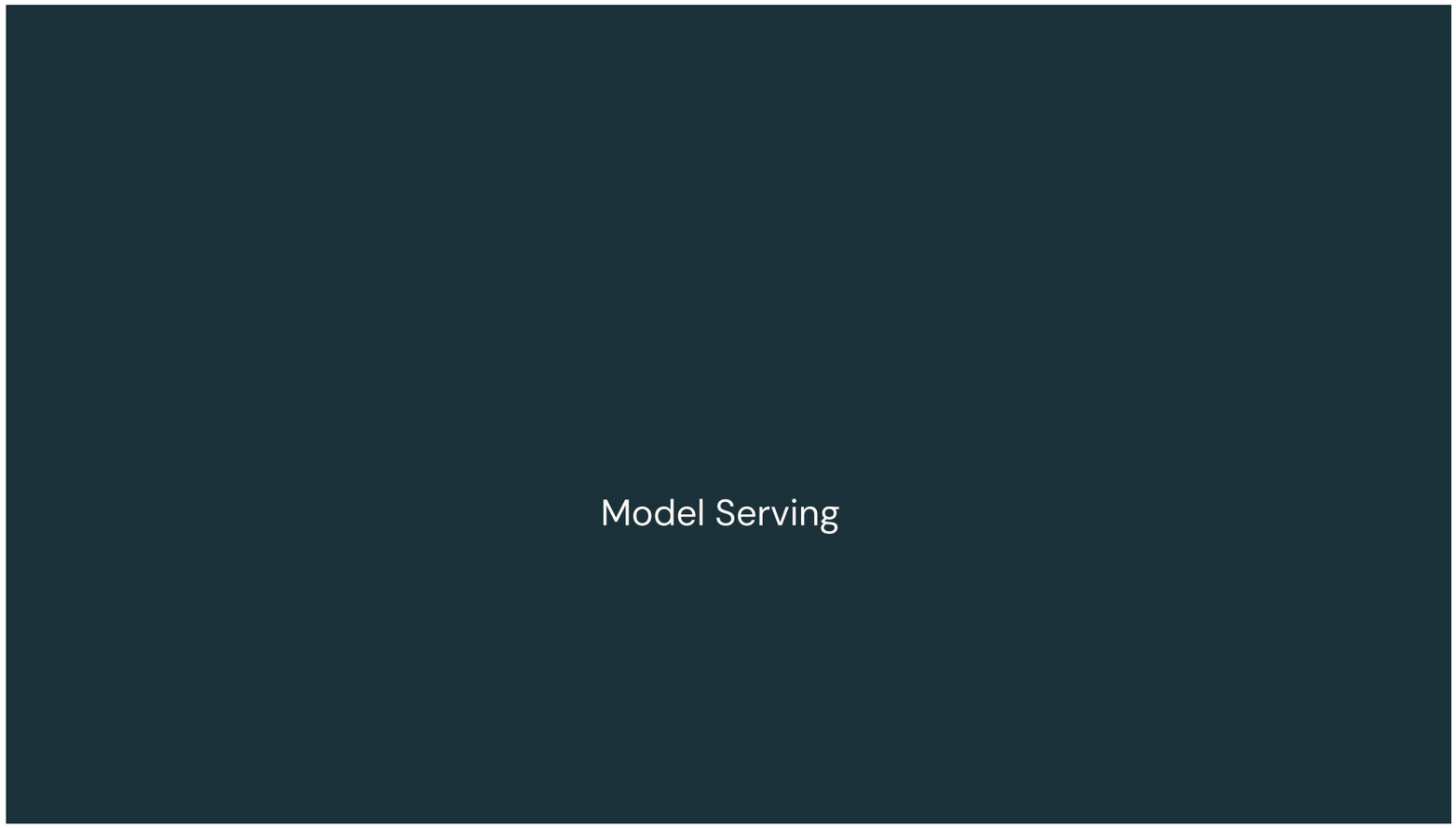
Model Serving solves this with **unified access**. It provides a single, standard query interface (API and SDK) for managing all types of AI models:

- **Agents and chains:** Custom logic — such as LangChain and PyFunc — running on serverless compute
- **Hosted models:** Open source foundation models — such as Llama and DBRX — fully managed by Databricks
- **External models**: Secure proxies to third-party providers — such as OpenAI, Anthropic and Gemini

This unified layer means you can swap a proprietary model for an open source one without changing your application code, preventing vendor lock-in and simplifying operations.

AI Gateway: Centralized governance

Sitting directly on top of Model Serving is the **AI Gateway**. If Model Serving is the pipe, AI Gateway is the filter. It acts as a secure, intelligent proxy that enforces governance policies on every request and response, replacing diverse security protocols with one centralized control plane.



> "Databricks AI Gateway is providing us with a secure way to consume AI models and connect them to our proprietary data. This enables us to build secure, compliant and context-aware AI systems."
>
> — Kapil Ashar, Vice President, Accolade

Traffic policies and reliability

The gateway manages the flow of traffic to ensure uptime and cost control.

- **Rate limiting:** Granular controls to prevent runaway costs by capping usage (e.g., tokens per minute) per user or endpoint
- **Fallbacks and retries:** Automatically reroutes requests if a provider fails. If Azure OpenAI returns a 429 error, the gateway can seamlessly failover to a backup model without the user noticing.

Comprehensive guardrails for safety

Safety is nonnegotiable. The gateway enforces real-time policies to filter inappropriate content.

- **Personally identifiable information (PII) detection:** Automatically scans inputs and outputs and masks sensitive data, such as payment card information (PCI) and protected health information (PHI), before it leaves the environment
- **Safety filtering:** Blocks harmful or toxic content in real time

> "Guardrails help us prevent unsafe content from reaching our end users. With payload logging, we can also trace guardrails to track the performance of the application."
>
> — Ryan Jockers, Assistant Director, North Dakota University System

Simplified access and experimentation

Because governance is centralized, teams can experiment faster without compromising security.

- **Traffic splitting:** Safely roll out changes by routing a small percentage of traffic (e.g., 10%) to a new model version for A/B testing before a full cutover

> "With Databricks AI Gateway, we were able to confidently experiment with a variety of open and proprietary AI models . . . This allowed us to integrate multiple GenAI apps, reducing information search time."
>
> — Harisyam Manda, Senior Data Scientist, OMV

Full observability and monitoring with MLflow

Governance requires visibility. You can't secure what you can't see. Databricks replaces the black box problem of disparate systems with **MLflow**, an open platform that unifies tracking, evaluation and observability for GenAI apps and agents.

MLflow Tracing: End-to-end visibility

GenAI agents are complex systems involving retrievers, tool calls and multi-step logic. MLflow Tracing provides complete execution visibility by automatically logging inputs, intermediate steps and outputs.

- **Deep debugging:** Instead of just seeing the final answer, tracing captures prompts, retrievals, tool calls and metadata for every step
- **Cost and latency monitoring:** Tracing automatically captures detailed performance metrics, allowing you to track latency and costs down to the individual component level
- **Production parity:** You can use the same instrumentation in development and production, ensuring consistent evaluation across environments

Automated evaluation with LLM judges

Measuring quality is no longer a manual process. MLflow 3 integrates LLM judges and scorers directly into the monitoring loop to automate quality assurance.

- **Built-in scorers:** Automatically evaluate production traffic for key metrics like correctness, relevance and safety
- **Custom judges:** Define specific business logic to monitor adherence to internal policies (e.g., "Did the agent cite a document?")
- **Production monitoring:** Continuously monitor a sample of production traffic using these judges to detect quality drift over time

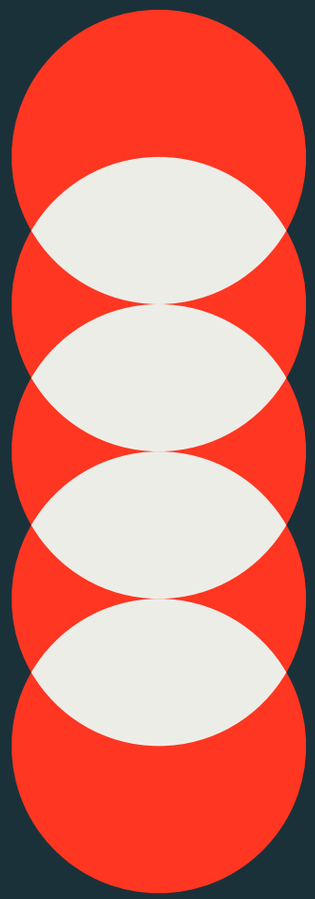
Feedback loops and continuous improvement

To close the loop, MLflow 3 includes review apps that collect feedback from domain experts and end users. This feedback is crucial for:

- **Aligning judges:** Using expert feedback to tune automated judges and scorers
- **Dataset curation:** Selecting representative traces to build evaluation datasets for future testing
- **Version tracking:** Comparing app and prompt versions to track improvements over iterations

By combining the **unified access** of Model Serving with the **centralized governance** of AI Gateway and **observability** of MLflow, organizations can confidentially deploy agents that are governed, observable and production-ready AI architecture.

Summary



Whether you're looking to disrupt traditional industries, enhance creative endeavors or solve complex problems in novel ways, GenAI offers a wide range of possibilities. The potential applications are limited only by your imagination and willingness to experiment. Remember, every significant advancement in this field began with a simple idea and the courage to explore it further.

For those seeking more knowledge or who are simply curious about the latest developments in the realm of GenAI, we've provided some resources on training, demos and product information.

Resources

Databricks Agent Bricks unifies the AI lifecycle from data collection and preparation to model development and LLMOps to serving and monitoring. The following features are specifically optimized to facilitate the development of GenAI applications:

- Unity Catalog — For governance, discovery, versioning and access control for data, features, models and functions
- MLflow — For model development tracking
- Model Serving — For deploying LLMs. You can configure a model serving endpoint specifically for accessing GenAI models:
  - State-of-the-art open LLMs using Foundation Model APIs
  - Third-party models hosted outside of Databricks. See External models in Model Serving.
- Vector Search — Provides a queryable vector database that stores embedding vectors and can be configured to automatically sync to your knowledge base
- Lakehouse Monitoring — For data monitoring and tracking model prediction quality and drift using automatic payload logging with inference tables
- AI Playground — For testing GenAI models from your Databricks workspace. You can prompt, compare and adjust settings such as system prompt and inference parameters.
- Foundation Model Fine-tuning — now part of Databricks Model Training — For customizing a foundation model using your own data to optimize its performance for your specific application

- Databricks Agent Framework — For building and deploying production-quality agents like RAG applications
- MLflow — For evaluating the quality, cost and latency of GenAI applications, including RAG applications and chains

Courses

- Take the Get Started With Generative AI self-paced tutorial and earn a Databricks certificate
- Generative AI Fundamentals (Databricks Academy)
- Generative AI Engineering With Databricks (instructor-led training via Databricks Academy)
- Look to Databricks Training and Databricks Academy for new courses
- Governing AI Agents with DeepLearning.ai (Databricks Free Edition)

Reading

- A Compact Guide to AI Agents
- Blog posts
  - Building State-of-the-Art Enterprise Agents 90x Cheaper With Automated Prompt Optimization (Sep 2025)
  - Judging With Confidence: Meet PGRM, the Promptable Reward Model (Aug 2025)
  - Agent Learning From Human Feedback (ALHF): A Databricks Knowledge Assistant Case Study (Aug 2025)
  - TAO: Using test-time compute to train efficient LLMs without labeled data (March 2025)
  - Introducing Serverless Batch Inference (March 2025)
  - Accelerate AI Development With Databricks: Discover, Govern and Build With MCP and Agent Bricks (Nov 2025)
  - Governing AI Agents With Unity Catalog (July 2025)
  - Introducing OfficeQA: A Benchmark for End-to-End Grounded Reasoning (Dec 2025)
- The Big Book of MLOps: Second Edition for a deep dive on MLOps with Databricks, including LLMOps
- Databricks AI page for a product overview, details on features and links to many resources
- Databricks documentation for GenAI for AWS, Azure and GCP

Build Production-Quality GenAI Applications — See How

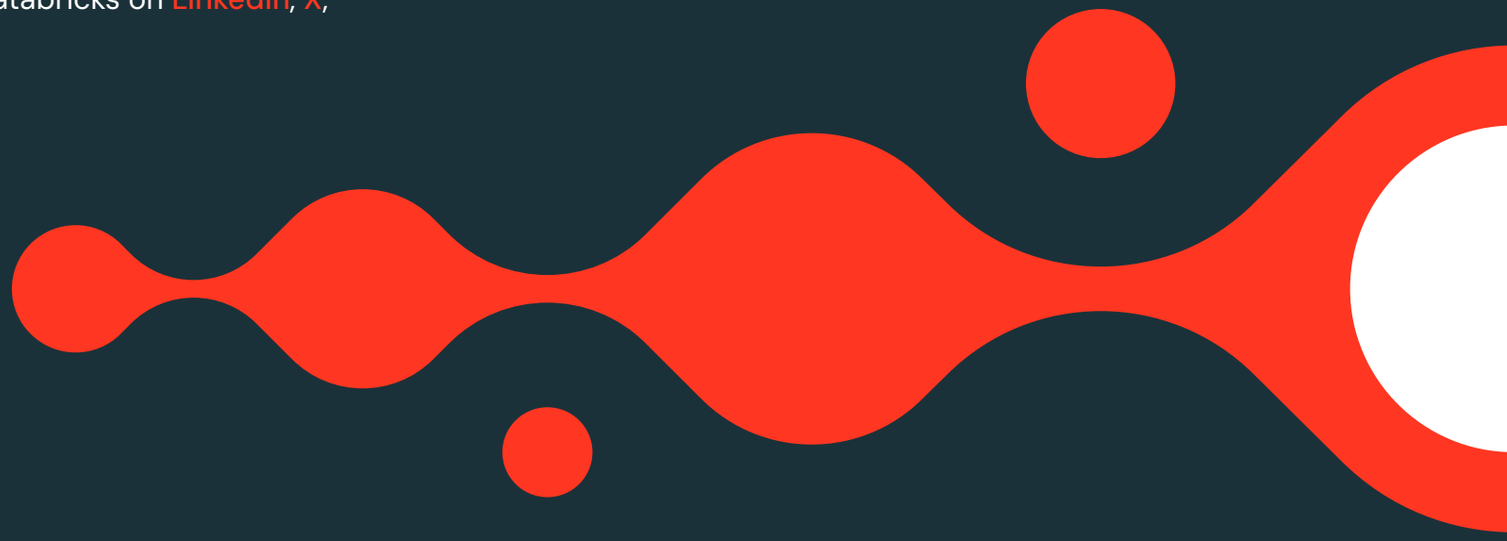
Create high-quality generative AI applications and ensure your output is accurate, governed and safe. See why over 10,000 organizations worldwide rely on Databricks for all their workloads from BI to AI — test-drive the full Databricks Platform free for 14 days.

Try Databricks free **Take Generative AI Fundamentals On-Demand Training**

About Databricks

Databricks is the Data and AI company. More than 20,000 organizations worldwide — including adidas, AT&T, Bayer, Block, Mastercard, Rivian, Unilever and over 60% of the Fortune 500 — rely on Databricks to build and scale data and AI apps, analytics and agents. Headquartered in San Francisco with 30+ offices around the globe, Databricks offers a unified Data Intelligence Platform that includes Agent Bricks, Lakeflow, Lakehouse, Lakebase and Unity Catalog. To learn more, follow Databricks on LinkedIn, X, YouTube and Instagram.

Contact us for a personalized demo



databricks

© Databricks 2026. All rights reserved. Apache, Apache Spark, Spark and the Spark logo are trademarks of the Apache Software Foundation. Privacy Policy | Terms of Use