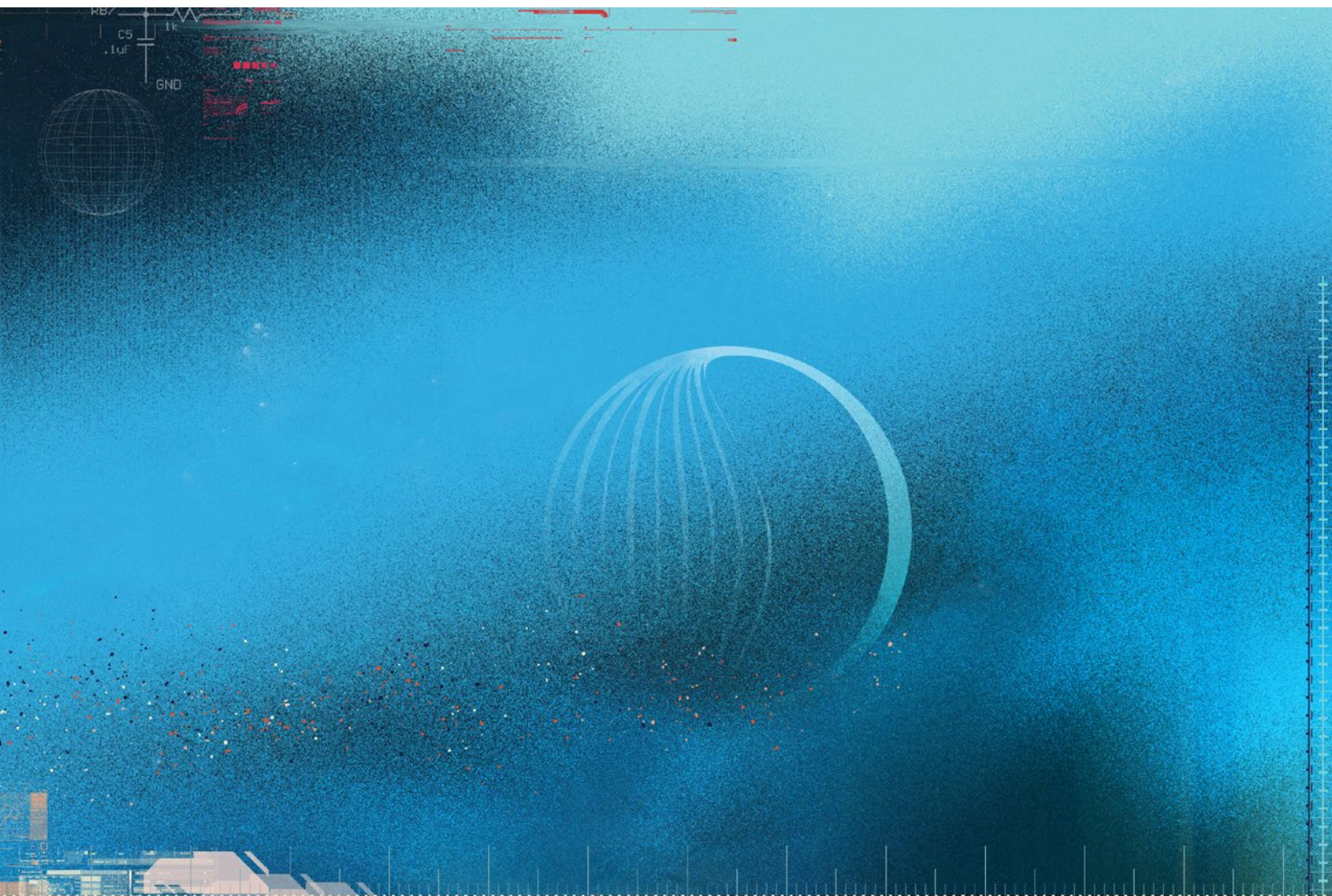




Snowflake vs. Databricks: An AI and Enterprise Readiness Evaluation

AUTHORS

Mitch Lewis

Performance Analyst | Signal65

Russ Fellows

VP, Labs | Signal65

IN PARTNERSHIP WITH



OCTOBER 2026

Executive Summary

Data lakehouses have become the operational backbone of data-driven enterprises, and increasingly, of enterprise AI. That role raises the bar for what a platform must be: open and interoperable, with enterprise-grade security and efficient price-performance at its core, and agentic AI capabilities layered on top, not just scalable storage and compute.

To test that bar directly, Signal65 built the same realistic customer scenario, utilizing the US DOT Flight Delays dataset, on both Snowflake and Databricks: ingesting data, transforming it, applying governance and PII protection, building an application, failing over for disaster recovery, and sharing data externally. Tests were attempted twice on each platform: once manually, and once using each platform's agentic AI tooling.

The same engineers, the same dataset, and the same requirements were used throughout, so any differences in time, effort, or capability largely reflect the platforms themselves rather than the person using them.

Key findings include:

Agentic Coding is the Decisive Differentiator

- Snowflake CoCo completed the full seven-phase workflow in 68 minutes versus 316 minutes for the manual workflow, 4.6x faster.
- CoCo completed every phase with minimal human interaction, including steps Genie Code did not complete in this evaluation.
- Snowflake provides a first party agent that spans in-platform and local development.

AI Activation Extends Beyond the Platform Boundary

- Snowflake's CoCo activates across the full data platform, operating both inside Snowflake and directly on the developer's own machine, with no third-party tooling required — only CoCo's own CLI, a one-time three-minute install.
- In this evaluation, Genie Code did not complete the full external application workflow without manual intervention. Completing a comparable agentic experience required a third-party agent (Claude Code) that took roughly 4x longer.

Cross-cloud Disaster Recovery Remains a Clear Feature Gap

- Snowflake configured AWS-to-Azure failover in 30 minutes; the evaluation did not reproduce an equivalent cross-cloud capability on Databricks.

Governance is Faster because it Lives In-line

- Snowflake completed the same governance and PII-protection tasks 32% faster than Databricks' separate admin console, a gap likely to recur as roles or policies change.

Key Highlights



4.6x faster with agentic development

CoCo rebuilt the same Snowflake workflow nearly autonomously, in 68 minutes versus 316 manually



Cross-cloud disaster recovery

AWS to Azure failover as a configured feature, not a custom build



32% faster governance

Masking, tagging and access policies, hands-on vs. Databricks

Methodology

This evaluation aimed to compare the operational efficiency, user experience, and feature differences between Snowflake and Databricks throughout a realistic, end-to-end workflow. To complete this, Signal65 conducted a hands-on evaluation in both platforms, measuring the time and complexity involved at each step.

The workflow involved the following steps:

- Setup
- Data Ingestion
- ETL
- Governance
- AI-accelerated Application Development
- Disaster Recovery and Failover
- External Sharing

The same workflow was attempted twice on each platform: once manually and once using each platform's agentic AI tools. (See [Experiment 2](#) for scope differences in the Databricks agentic run.)

Dataset

The workflow completed in this evaluation focused on the US DOT Flight Delays dataset, augmented with additional data including historic weather data, airline sentiment data from X (formerly Twitter), and synthetically generated passenger information. The dataset included a mixture of structured and semi-structured data. This data was intended to provide the basis for a realistic workflow, including handling structured and semi-structured data, multiple joins, PII identification and masking, and remediating data quality, rather than pure scale.

Source	Format	Rows	Purpose
US DOT flight delays (2015)	CSV	5,819,079	Structured fact table
Airlines / airports lookups	CSV	14 / 322	Dimensional joins
BTS airport ID crosswalk	CSV	6,259	Data-quality remediation
Passenger bookings (synthetic)	JSONL	275,115	PII governance target
Airport weather (Open-Meteo)	JSONL	3,650	Multi-source join
Airline sentiment (X/Twitter)	JSONL	14,640	Semi-structured text / AI enrichment

Figure 1: Evaluation dataset sources.

Test Tracks and AI Experiments

The key goal of this study was to evaluate the efficiency of using each platform. To do this, each platform was evaluated twice, representative of two distinct approaches to development: a traditional manual approach, and an automated approach utilizing agentic AI.

Evaluating both manual and agentic workflows provides insight into both traditional processes, and the increasingly common trend of AI automation. In both cases, manual and agentic, evaluation of both platform's AI tooling was critical. While AI usage was more extensive in the agentic test scenarios, both test scenarios included AI experimentation.

Track	Description	AI Experiment
Snowflake Manual	Manual workflow, with an AI-developed application using Snowflake CoCo	Single Task: AI developed dashboard
Snowflake Agentic	Same scenario, clean database, single brief to Snowflake CoCo CLI	Entire Workflow
Databricks Manual	Repeat of the manual workflow on Databricks	Single Task: AI developed dashboard
Databricks Agentic	Limited AI workflow using Genie Code	Entire Workflow

Figure 2: Test Track and AI Experiments

Scope and Limitations

Findings in this report reflect the products as configured and available on the dates tested. Both platforms release frequently, particularly their AI tooling, and capabilities, packaging and pricing may have changed since. Results should be read as a point-in-time evaluation of the specific configurations below, not as an absolute statement of platform capability.

	Snowflake	Databricks
Product Version	Snowflake Enterprise Trial, Business Critical Edition used for DR testing	Databricks Premium
Deployment model	Standard account, trial	Express Setup, serverless, Unity Catalog auto-enabled
Cloud	AWS us-east-2	AWS us-west-2
Date Tested	Jul 7–23, 2026	Jul 27 – Aug 4, 2026

Figure 3: Test Environment Overview

Note: Platforms ran in different AWS regions (us-east-2 and us-west-2). Regional effects were not separately measured; all timings represent hands-on wall-clock engineering time.

Timing methodology

All durations are wall-clock time recorded per task by the engineer performing the work, and include time spent on troubleshooting and remediation as encountered.

Inclusion of one-time setup. Headline totals include one-time environment setup and tool installation, since those steps represent real effort for an organization adopting a platform.

	Including setup time	Excluding Setup time
All phases (excluding DR)	Snowflake 286 min / Databricks 425 min — 33% faster	Snowflake 263 min / Databricks 399 min — 34% faster
Hands-on phases only (excl. AI-assisted app build)	Snowflake 266 min / Databricks 342 min — 22% faster	Snowflake 246 min / Databricks 327 min — 25% faster
Manual vs. CoCo-driven build (Snowflake only)	316 min / 68 min — 4.6x	293 min / 68 min — 4.3x

Figure 4: Headline Timings Including and Excluding One-Time Setup

One-time installation accounted for 23 minutes on Snowflake (Snowflake CLI, 20 min; CoCo CLI, 3 min) and 26 minutes on Databricks (Databricks CLI, 15 min; Claude Code and Databricks Agent Skills, 11 min). Excluding it does not change the direction of any finding. CLI tooling was installed during the manual track; the agentic run’s 68 minutes includes no installation time.

Test sequencing: Platforms were evaluated sequentially rather than in parallel, with Snowflake first. The second platform evaluated benefits from architectural decisions already made and data-quality issues already discovered. Comparative timings should be read with this asymmetry in mind.

Practitioner experience and single run: Each task recorded is from a single measurement by a Signal65 engineer. Times reflect that engineer’s experience and are not averaged across runs. The experience of different practitioners may vary.

Agentic runs are non-deterministic. Agent-driven builds depend on model behavior that varies between executions. A repeat run of the same prompt may produce different sequencing, different scope decisions and materially different completion times. Agentic results should be treated as illustrative of an approach rather than as reproducible benchmarks.

Alternative methods were not exhaustively tested. Where a platform offers multiple routes to the same outcome, recorded times reflect the method selected during testing. Different API choices, ingestion paths or tooling may yield different results.

Scope of workload. Testing used a single scenario built on one dataset of approximately 5.8 million records. It is not a production-scale, concurrency or multi-workload evaluation, and performance at scale was not assessed.

Comparative claims are scoped to tested configurations. Where this report states that a capability was unavailable, this reflects what was available and documented as a managed capability in the tested configuration on the dates tested. It does not assert that equivalent outcomes are unachievable through custom engineering, alternative configurations, or subsequent product releases.

Experiment 1: Manual Build Comparison

The first evaluation involved a primarily manual workflow, requiring hands on execution of processes in both platforms, with AI utilized to create a React-based dashboard. Across hands-on steps alone, Snowflake completed the same work 22% faster than Databricks (266 vs. 342 minutes). Including the AI-assisted app build widens the gap to 33% (286 vs. 425 minutes).

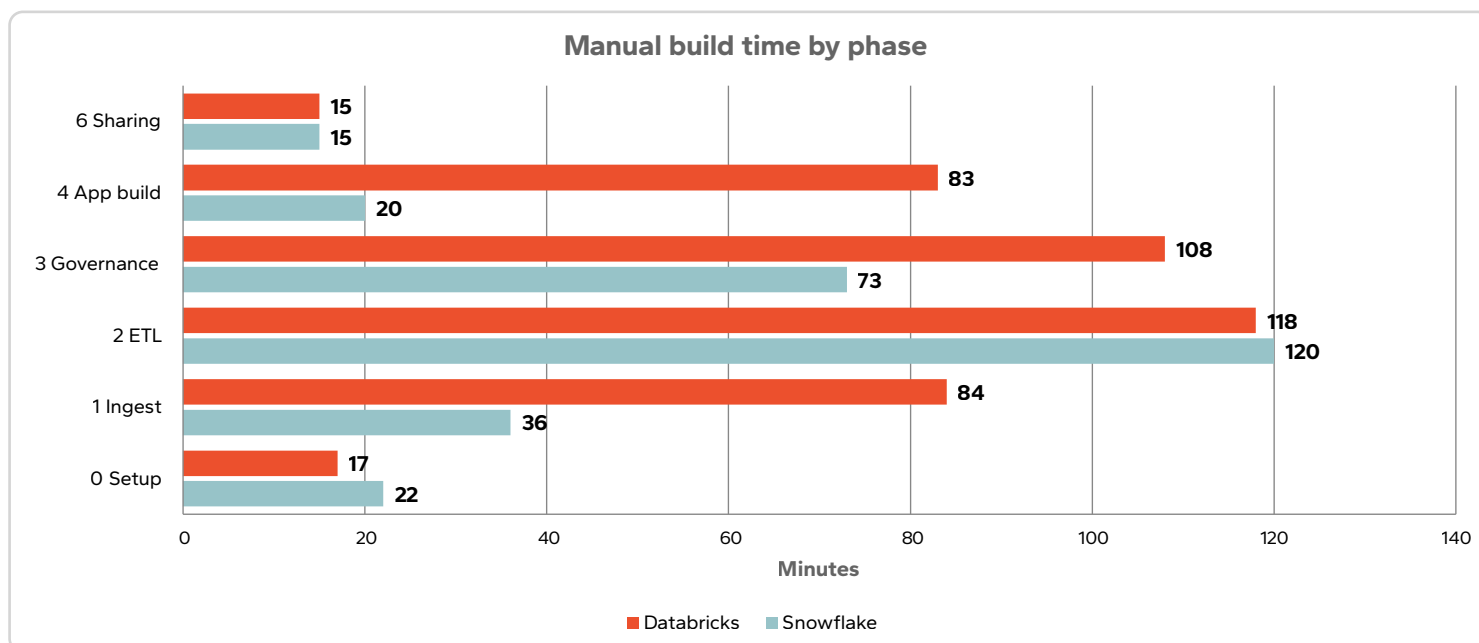


Figure 5: Manual Build Time Comparison (Phase 5 (DR) omitted from totals; see Disaster Recovery section.)

For several steps of the workflow, such as initial setup, ETL, and data sharing, the manual approach was found to be generally similar. These tasks involved relatively similar steps, or the same SQL code, and differences in completion time was largely due to minor variance in engineering efficiency. While the manual build was found to be relatively similar, key differences did arise.

Disaster Recovery: A Feature Gap, Not Just a Time Gap

Notably, the disaster recovery step was omitted from the total times recorded. This step involved setting up disaster recovery and failover between two clouds: AWS and Azure. In Snowflake, this was completed in approximately 30 minutes, however, this functionality was not reproduced in Databricks in this evaluation, and therefore not counted in the total time comparison.

At the time of testing, the evaluation did not identify a Databricks configuration equivalent to Snowflake's tested AWS-to-Azure account failover. Databricks provides cross-region disaster-recovery capabilities, but the tested scenario did not establish equivalent cross-cloud failover.

That distinction matters: a regional outage is survivable with single-cloud DR, but a cloud-wide control-plane, identity, or account-level failure is not, since it takes every region in that cloud down with it, including the DR site.

Snowflake's Business Critical Edition supports AWS-to-Azure failover directly; this evaluation configured it in 30 minutes.

Not every organization needs protection against a cloud-wide failure, for many, single-region or single-cloud DR is a reasonable risk tradeoff. But for organizations whose recovery requirements assume that an entire cloud provider could become unavailable, this evaluation did not identify a native Databricks path to that level of resilience; achieving it would require a custom-built, self-managed failover solution outside the platform, dramatically increasing time, complexity, and cost.



AI-Accelerated Application Development: CoCo vs. Genie Code vs. Third-Party Agents

Of the tasks that both platforms could complete, the largest difference in implementation time was recorded during the AI Accelerated Development step. The goal of this step was to create a React dashboard displaying the airline performance data housed in either platform.

The dashboard was required to show the average departure delay by month per airline, a sortable table ranked by total flights and cancellation rate, and dropdown to filter by airline. It additionally included panels correlating delay with weather and showing average customer sentiment by airline. Each agent was required to set up the project structure, install dependencies, and get the application running locally.

Functionally equivalent prompts were used for all tests, with slight variations to account for specific table and platform names.

In Snowflake, this process was completed with the CoCo CLI. In total, CoCo completed the task and achieved a working, locally deployed React dashboard in 20 minutes, three minutes of which was one-time installation of the CoCo CLI, leaving roughly 17 minutes of actual build time. CoCo provided a nearly hands-off, automated experience. The only manual effort required by Signal65 engineers during the entire process was approving CoCo's actions, and editing credentials in a single .env file.

Alternatively, two distinct approaches were tested for Databricks: Genie Code, its native AI tool, and a third party agent connected with Databricks AI skills, both processes took significantly longer than CoCo to achieve similar results.

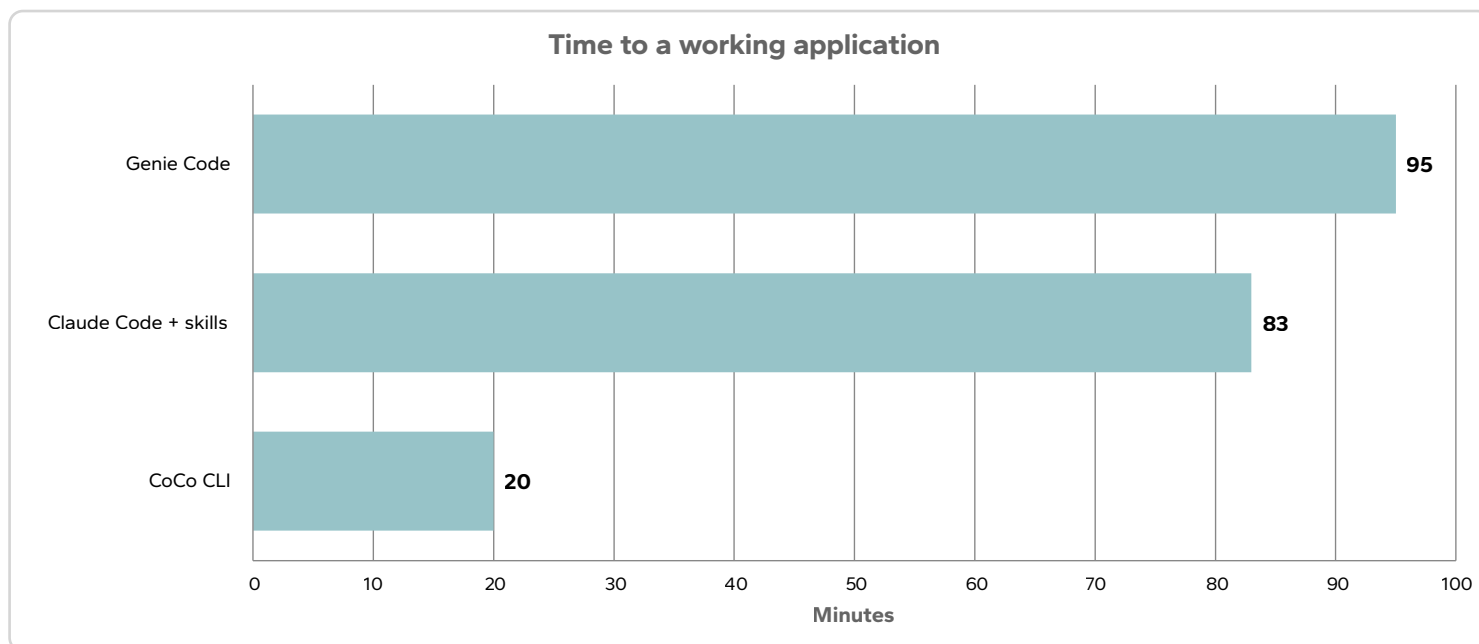


Figure 6: AI-assisted Application Development Time

While Genie Code was provided with an equivalent prompt as CoCo, in this evaluation configuration, Genie Code did not build and deploy the external application autonomously. Instead, Genie Code generated code which required manual copying and editing. While the initial code generation was completed within five minutes, subsequent troubleshooting was extensive, requiring repeated back-and-forth between querying Genie Code and manually editing code. In total, this extended the time to a working application to 95 minutes.

The result technically worked, but the process bore little resemblance to autonomous development: it required sustained manual intervention for prompting and code editing, and more complex applications would likely stress this workflow further.

Since Databricks Genie Code was limited in its ability to produce an automated AI generated application, an alternative solution utilizing a third party agent was also evaluated. This evaluation utilized Claude Code with access to the Databricks Agent Skills toolkit. This approach more closely resembled the agentic experience found with Snowflake CoCo with little manual interaction; however, it required over 4x the total time at 83 minutes to achieve a working application. Eleven of those minutes were one-time setup, installing Claude Code and the Databricks Agent Skills toolkit, before any development work began.

The third party approach highlights a key difference in agentic AI strategy between Snowflake and Databricks. Snowflake provides a native AI coding agent in CoCo, while also supporting 3rd party agents. Databricks provides Genie Code for in-platform development and also supports governed integration with external coding agents. In this test, the external application workflow was completed using Claude Code and Databricks Agent Skills rather than Genie Code alone. Utilizing third party AI tools introduces additional complexity, licensing costs, and security and compliance considerations.



Where Databricks Added Friction: Data Types and Governance

In addition to the significant differences in disaster recovery and AI capabilities, two additional steps were found to require additional time and effort in Databricks compared to Snowflake.

During the Data Ingestion process in Databricks, an error was experienced when copying CSV data into a table due to a type mismatch. In Snowflake, the types were automatically cast and the error was avoided. Although this error led to a difference in implementation time, this specific issue is representative of a one-time error rather than a true, ongoing operational difference between the two platforms.

Debugging this error, however, did uncover a more pressing pain-point. To identify and correct the error, Signal65 engineers attempted to utilize Databricks' Genie Code AI assistant, however, it was unable to correctly identify the error or fix the code. Significant time was spent asking Genie Code to fix the problem before engineers gave up on that approach and resolved it through manual investigation. In total, the data ingestion step took 2.3x longer in Databricks.

A second source of operational friction found in Databricks was during the governance steps. Creating account-level users and groups, and assigning them to the workspace, required the Databricks account console rather than SQL; Unity Catalog object grants were applied in SQL. On Snowflake, the entire sequence — users, roles, and grants — was completed in-line in SQL, simplifying and streamlining the governance process.

Automated PII classification, which underlies both platforms' governance workflow, showed both platforms correctly flagging the most sensitive identifiers, email, national ID, name, and date of birth, with high confidence. Where they diverged was in how each handled more ambiguous fields.

Snowflake's classifier was conservative: it flagged phone and passport numbers only when confidence was high, leaving some coverage gaps, and left billing city untagged rather than guess. Databricks' classifier cast a wider net, catching billing city automatically, but it also produced false positives elsewhere: it flagged the destination airport code as location data, and separately flagged a number of fields within the free-text sentiment data, some plausibly correct and others likely false positives.

In effect, this is a precision-versus-recall tradeoff: Snowflake required manual review to close coverage gaps, while Databricks required manual review to remove incorrect tags.

Field	Snowflake	Databricks
Phone number	Flagged, medium confidence	Flagged as phone_number
Passport number	Flagged, low confidence	Flagged as passport
Billing city	Not flagged; required manual tagging	Flagged automatically, as location
Destination airport code	Correctly ignored (not PII)	Flagged as location - false positive
Sentiment / tweet text	Not flagged	Flagged multiple fields; mix of plausible hits and likely false positives

Figure 7: PII Classification Comparison

Experiment 2: AI Automated Workflow

The second experiment repeated the entire workflow, but changed the implementation from a manual approach to a fully automated process completed by each platform's AI agent solution: Snowflake CoCo or Databricks Genie Code.

The intention of this experiment was to evaluate if an AI agent could complete the entire workflow from start to finish. To evaluate this, both Snowflake CoCo and Databricks Genie Code were prompted with the same instructions, including all steps from the previous manual workflow.

Snowflake CoCo completed the entire process in 68 minutes, 4.6x faster than the manual workflow — which itself included a CoCo-assisted dashboard build, so the gap versus fully manual work would be larger. CoCo was initially prompted with instructions to complete the workflow and told to generate an initial plan. After reviewing the initial plan, minor adjustments were provided to ensure the process closely matched manual steps. After this round of instruction, CoCo completed all tasks with minimal human intervention. Human intervention was limited to basic administrative tasks such as approving actions, and editing credentials.

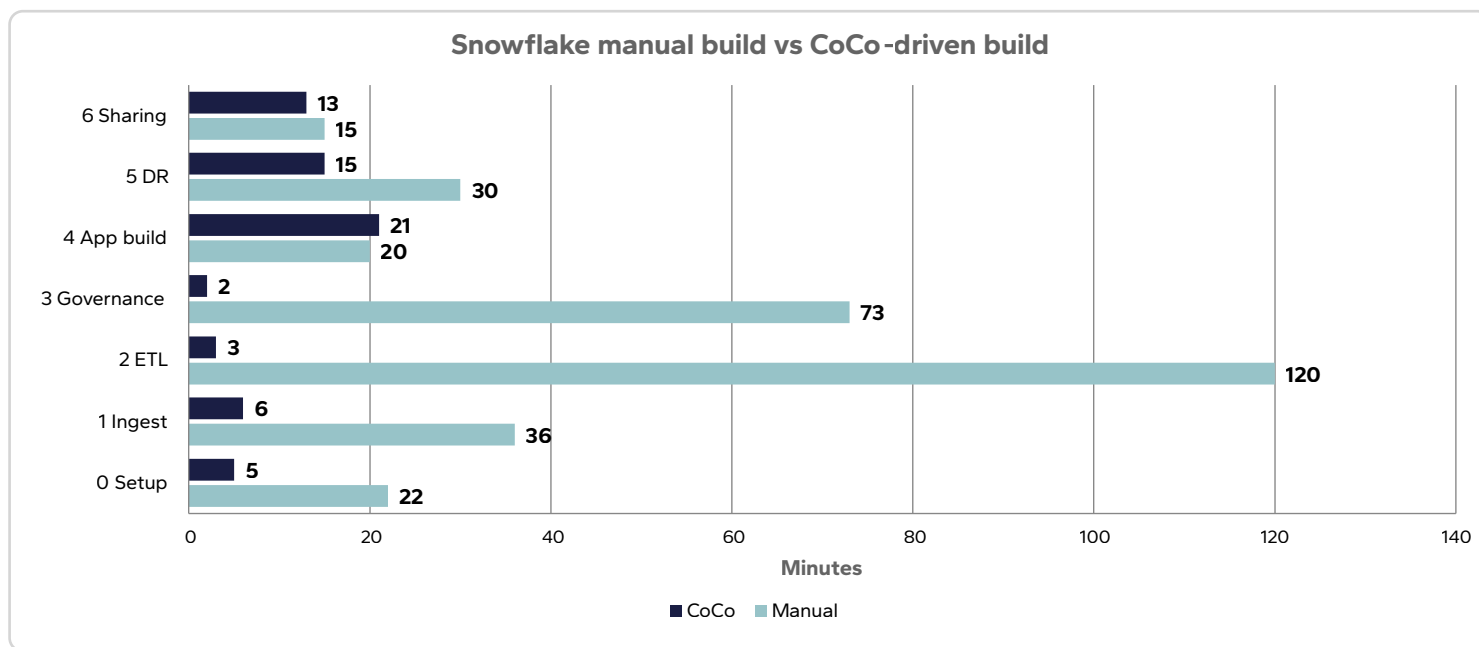


Figure 8: Snowflake Manual vs CoCo Time Comparison

(Phase bars total 65 minutes; the 68-minute total includes approximately 3 minutes of agent-initiated data enrichment and documentation not assigned to a workflow phase.)

CoCo's automated approach compressed the most time-consuming manual work almost entirely: ETL fell from 120 minutes to 3, governance from 73 minutes to 2, and data ingestion from 36 minutes to 6. ETL and governance alone account for 61% of the manual build, which is why the total moves so dramatically.

Disaster recovery and data sharing took longer than CoCo's other tasks aside from the app build, at 15 and 13 minutes respectively, since they involve handling credentials and cross-account context rather than primarily authoring SQL. CoCo, however, was still faster than the manual approach on both.

Application development was the one exception, at 21 minutes versus 20, but this task was completed with CoCo in both scenarios. Since application development was automated in both scenarios, similar timing is to be expected and the one-minute gap is primarily run-to-run variance.

Signal65 Comment: It's worth noting that this comparison actually understates the efficiency gain AI-assisted development can offer. Both the manual and the agentic workflow utilized CoCo to build the dashboard application. A true from-scratch manual build, with no AI assistance at all, would typically take considerably longer. CoCo achieved a working React dashboard in roughly 20 minutes, indicating meaningful time savings for developers.

Upon completion, Signal65 engineers evaluated CoCo's work to ensure tasks were completed successfully. In two cases, minor adjustments were required. The first was a data-quality fix to airport IDs — a defect the manual build caught only at validation and remediated via a crosswalk table, and which CoCo fixed with a single prompt in under one minute.

The second issue detected was an error with external data sharing. While the connection to the shared account had been made, data was not appearing. The underlying cause was that CoCo had used the account name instead of the account locator in its connection string. CoCo needed three additional prompts to diagnose and fix the issue, closing the gap with the manual execution time.

In Databricks, Genie Code was given a nearly-identical prompt as CoCo, with only minor adjustments made to reflect the Databricks environment. Given the explicit requirements to ingest external data and create an external application, Genie Code was not actually able to complete the full workflow.

Further experimentation, with relaxed requirements, did show Genie Code's capabilities within the boundaries of the Databricks platform. When enabled to utilize existing data ingested from previous manual testing, Genie Code was able to perform ETL and create a dashboard application, however it was limited to deployment within the Databricks environment. It was unable to fully complete all governance steps due to the requirement to utilize a separate admin console. As with the manual approach, cross-cloud disaster recovery was not completed due to missing functionality.

In total, this experimentation took 66 minutes. While the wall-clock time was similar to CoCo's 68-minute workflow, the delivered scope differed significantly: CoCo completed all 7 phases while Genie Code completed 2 of 7 phases under relaxed requirements. This comparison is illustrative of each platform's AI capabilities, **however due to the differences in scope, wall-clock time is not directly comparable.**

Snowflake CoCo vs Databricks Genie Code Autonomous Workflow Comparison		
Platform	CoCo	Genie Code
Scope	Full Workflow (7/7 Phases)	Partial Workflow (2/7 Phases)
Wall clock	68 min – Full Completion	66 min – Partial Completion
Prompting	Minimal	Moderate
Uploaded raw data	Yes	No
App framework	React	Vanilla JS
Governance	Yes	Partial – console steps not completed
Deployable externally	Yes	No
Cross-cloud DR	Completed	Not completed

Figure 9: Snowflake CoCo vs Databricks Genie Code Autonomous Workflow Comparison

Additional Feature Differentiation

Beyond the operational differences Signal65 tested directly in the steps above, the platforms also differ across several other dimensions worth noting for a complete picture of enterprise readiness. These reflect documented platform capabilities rather than Signal65's own hands-on testing:

Capability statements below reflect vendor documentation reviewed as of October 26, 2026; all footnoted sources in this section were accessed on that date.

Open catalog: Unity Catalog supports Iceberg and exposes a REST catalog API¹, but the governance enforcement and operational metadata of the managed Unity Catalog service are not part of the open-source Unity Catalog project². Data can be read out, but the managed service's governance configuration does not transfer with it. Snowflake's answer is to keep governance open at the catalog level too, with Iceberg REST catalog, semantic views, and open governance APIs, so governance does not depend on a single vendor's control plane³.

Bidirectional interoperability: Databricks documents that writes to foreign tables are not covered by managed-table guarantees⁴. Reads from external catalogs go through Catalog Federation, which requires a storage credential and external location configured in advance⁵. Snowflake's Horizon Catalog federates in both directions using short-lived, vended credentials, so access stays scoped and expires automatically regardless of which direction data is flowing⁶.

¹ Databricks. What is Apache Iceberg in Databricks? Databricks Documentation (AWS). <https://docs.databricks.com/aws/en/iceberg/>

² Unity Catalog open source project. LF AI & Data Foundation. <https://github.com/unitycatalog/unitycatalog>

³ Snowflake. The Open Interoperability Standard: Why Bidirectional Iceberg REST Matters. Snowflake Blog. <https://www.snowflake.com/en/blog/bidirectional-interoperability-snowflake-horizon-databricks/>

⁴ Databricks. Work with Foreign Tables. Databricks Documentation (AWS). <https://docs.databricks.com/aws/en/tables/foreign>

⁵ Databricks. Enable Snowflake catalog federation. Databricks Documentation (AWS). <https://docs.databricks.com/aws/en/query-federation/snowflake-catalog-federation>

⁶ Snowflake. Use a Catalog-Linked Database for Apache Iceberg Tables. <https://docs.snowflake.com/en/user-guide/tables-iceberg-catalog-linked-database> Snowflake. Access Apache Iceberg tables with an external engine through Snowflake Horizon Catalog. Snowflake Documentation. <https://docs.snowflake.com/en/user-guide/tables-iceberg-access-using-external-query-engine-snowflake-horizon>



Key Takeaways

Agentic AI Accelerates Efficiency

When comparing manual processes, Snowflake was found to be more efficient than Databricks, however, the operational savings were concentrated in a few key areas: governance was the largest recurring difference, while larger one-time gaps appeared in data ingestion and the AI-assisted app build. Snowflake also differentiated itself as more capable in certain areas, such as cross-cloud disaster recovery, which could not be completed on Databricks in this evaluation.

For many of the core tasks achievable on both platforms, however, the overall processes, complexity, and effort required is similar.

The real operational accelerator found in this study is the use of AI agents. Snowflake CoCo was able to automate the entire workflow in approximately 1 hour, 4.6x faster than the manual workflow. By leveraging CoCo, developers and data engineers can free up significant time typically spent on these routine tasks, and re-direct their attention to more complex, strategic initiatives. For enterprises, this increase in operational efficiency saves time and enables faster innovation.

Not All AI Strategies are Equal

Within this study, three approaches to AI assisted development were evaluated: Snowflake CoCo, Databricks Genie Code, and third party agent integration.

Signal65 found Snowflake CoCo to provide a complete agentic experience in this evaluation, capable of automating tasks both within Snowflake and externally. CoCo provides a single, deeply integrated tool that accelerates developer productivity across the entire workflow. In this evaluation, Databricks Genie Code did not complete the requested externally deployed application without manual intervention, behaving more as an in-platform assistant for this workflow. Genie Code has since introduced Agent mode capabilities that were not validated during this evaluation.

Third-party AI tools such as Claude Code can close that gap, but they introduce additional complexity and cost. Third party AI tools may additionally introduce data governance and security concerns, limiting their use by certain organizations.

Snowflake CoCo	Databricks Genie Code	Third-Party Agents
First-party agent available in the tested environment*	First-party agent available in the tested environment*	External tooling, added alongside either platform
Operates inside Snowflake and on the developer's local machine	Operated only within the Databricks workspace in this evaluation	Operates locally, connecting to the platform via a toolkit
Built the dashboard in 20 minutes, nearly hands-off	Generated code in 5 minutes but could not deploy it; 95 minutes to a working app	Built the dashboard in 83 minutes, nearly hands-off
Completed all seven workflow phases, including governance and cross-cloud DR	Did not complete governance or cross-cloud DR in this evaluation	Scope limited by what the platform itself supports
Single vendor: one tool, one contract, consistent security model	No third-party agent license used in this test; gaps require filling elsewhere	May add licensing cost, plus data governance and security review

Figure 10: AI Strategy Comparison
**Editions as tested; see Figure 3*

Governance Efficiency Matters Over Time

Across manually completed processes, governance was found to be the largest recurring point of operational differentiation between the two platforms — one-time gaps such as ingest error remediation were larger but are not expected to repeat. While Snowflake enables governance control via inline SQL, Databricks introduces additional friction with a separate admin console to create and assign users and groups. In total, the governance related tasks were completed 32% faster in Snowflake, with the entire process completed in SQL.

Unlike a one-time implementation cost, governance friction is a recurring operational cost that can be expected to repeat with each role change, policy update, or new user onboarding — an effect projected from the measured 32% gap, not separately measured.

Cross Cloud Disaster Recovery Enhances Reliability

Cross-cloud disaster recovery is a clear feature gap between the two platforms. Snowflake provides AWS to Azure failover as a configurable capability in its Business Critical Edition, while the evaluation did not reproduce equivalent cross-cloud failover on Databricks. For organizations that require this level of resilience, compromising on a less-robust failover solution is not an option.



Final Thoughts: Snowflake is Built for the Agentic Enterprise

This evaluation found Snowflake to be a robust, enterprise-ready data lakehouse platform built for the era of agentic AI: one that enhances operational efficiency, strengthens data resilience, and accelerates development through native AI tooling.

Snowflake demonstrated meaningful efficiencies and feature advantages, including cross-cloud disaster recovery and inline governance controls, that suit it well to a wide range of enterprise requirements. As manual workflows increasingly give way to AI agents, Snowflake's first-party, native agentic tooling positions it well for that shift.

The most striking result in this evaluation is how wide the gap was between agentic and manual work on Snowflake. A single brief to CoCo, with light supervision — plan review, action approvals, and a handful of corrective prompts — produced the entire pipeline: ingestion, transformation, governance, a working application, cross-cloud failover, external sharing, in 68 minutes. The manual workflow took 316 minutes.

In this evaluation, Databricks' Genie Code did not complete the full workflow, requiring substantial manual implementation for the tested scenario.

Snowflake is built on open standards, with portable catalogs, bidirectional federation, simplified governance, and cross cloud disaster recovery. It further provides enhanced operational efficiency and native agentic functionality capable of completing the tested workflow 4.6x faster than manual execution.

For organizations evaluating where to build their next generation of data and AI infrastructure, this evaluation points to a clear conclusion: in the workflows tested, Snowflake delivered the operational efficiency, resilience, and native agentic capability that modern data and AI teams increasingly demand.

Appendix: Evaluation Steps by Phase

The steps below outline the general sequence performed in each phase of the evaluation. They are described at a platform-neutral level; the specific implementation differences between Snowflake and Databricks encountered along the way are discussed in the sections above.

Phase 0 - Setup

- Provision compute (a warehouse or cluster) and create the working database/schema
- Install and configure the platform's command-line interface

Phase 1 - Data Ingestion

- Upload raw source files (CSV and JSON)
- Define file formats for structured and semi-structured data
- Load structured data into tables
- Load semi-structured data into tables
- Validate the ingest (row counts, spot-checks)
- Ingest and join supplemental reference data to resolve data-quality issues identified during validation

Phase 2 - ETL

- Build curated tables from the raw source data
- Update curated tables to incorporate data-quality fixes
- Flatten semi-structured data into query-ready tables (passenger, weather, and sentiment data)
- Build aggregation tables for daily and monthly reporting

Phase 3 - Security & Governance

- Run automated PII classification against the curated dataset
- Apply classification tags to flagged columns
- Create user roles for restricted and administrative access
- Create and validate column-level masking policies
- Apply row-level filtering policies to restrict data by partner/carrier

Phase 4 - AI-Accelerated Application Development

- Install and configure the platform's AI agent tooling
- Prompt the agent to build a dashboard application against the curated data
- Deploy and run the application locally

Phase 5 - Disaster Recovery & Failover

- Create a secondary account/environment in a second cloud provider
- Configure replication and failover between the primary and secondary environments
- Validate the failover configuration

Phase 6 - External Sharing

- Configure a data share scoped to a specific external partner
- Share curated data with the external account
- Validate that shared data is visible to the partner and correctly scoped

Important Information About this Report

CONTRIBUTORS

Mitch Lewis

Performance Analyst | Signal65

Russ Fellows

VP, Labs | Signal65

Cameron Moccari

Director of Operations | Signal65

PUBLISHER

Ryan Shrout

President and GM | Signal65

INQUIRIES

Contact us if you would like to discuss this report and Signal65 will respond promptly.

CITATIONS

This paper can be cited by accredited press and analysts, but must be cited in-context, displaying author's name, author's title, and "Signal65." Non-press and non-analysts must receive prior written permission by Signal65 for any citations.

LICENSING

This document, including any supporting materials, is owned by Signal65. This publication may not be reproduced, distributed, or shared in any form without the prior written permission of Signal65.

DISCLOSURES

Signal65 provides research, analysis, advising, and lab services to many high-tech companies, including those mentioned in this paper.

ABOUT SIGNAL65

Signal65 exists to be a source of data in a world where technology markets and product landscapes create complex and distorted views of product truth. We strive to provide honest and comprehensive feedback and analysis for our clients in order for them to better understand their own competitive positioning and create optimal opportunities to market and message their devices and services.



IN PARTNERSHIP WITH



CONTACT INFORMATION

Signal65 | signal65.com