Yes, there is a material difference, but it is mainly about *where orchestration happens, which client environment you work in, who the user is, and where the governance boundary sits*, rather than a hard wall of mutually exclusive capabilities. The reason is structural

the Snowflake MCP server is itself largely a delivery mechanism that surfaces Snowflake-native Cortex features to external clients like Claude. So "Claude Code + MCP to Snowflake" and "just Snowflake Cortex" are not two separate universes of capability. They are two access patterns over substantially overlapping machinery.

Explainer · June 4, 2026

What the MCP path actually is

The Snowflake MCP (Model Context Protocol) server exposes Cortex Search, Cortex Analyst, Cortex Agent, SQL Execution, Object Management, and Semantic View querying to MCP clients such as Claude Desktop and Cursor. In other words, the MCP path largely *re-surfaces* Cortex rather than replacing it. Snowflake's own documentation frames the server as providing tooling for Snowflake Cortex AI, object management, and SQL orchestration — explicitly a delivery mechanism for Snowflake-native Cortex AI, not a distinct feature set.

There are two distinct ways to stand this up, and the distinction matters for governance:

- A **Snowflake-managed/hosted server**, created as a Snowflake object, OAuth-authenticated, with its control plane running inside Snowflake's perimeter.
- An **open-source Snowflake-Labs server** run locally or in a container, configured via YAML.

The "data leaves Snowflake" argument plays out differently across these two: the managed server keeps the control plane in-perimeter, while a self-hosted server runs outside it.

Where the use cases genuinely diverge

Developer-workflow surface. The clearest "different use case" the MCP path unlocks is ad-hoc analysis sitting *alongside code inside an IDE* (Cursor, Claude Desktop), deployed at the individual or team level rather than enterprise-wide. This is a developer-at-the-keyboard motion, not an enterprise platform deployment.

Claude Code as an agentic coding environment. Claude Code is an agentic coding tool — file editing, git, bash, multi-language support, reasoning — that, via MCP, gains live access to Snowflake's execution and metadata layer: catalog search, schema exploration, query execution, semantic model queries, and Cortex Agent discovery. Those agentic coding capabilities are not something the native SQL/Python Cortex functions offer on their own. If your workflow is "write and run code while a model reasons over your repo *and* your Snowflake schema," that is the MCP path's natural home.

Native Cortex as a SQL-native AI layer. On the other side, native Cortex AI functions cover text generation, summarization, translation, sentiment, entity extraction, classification, document parsing, RAG, and text-to-SQL — all callable directly in SQL or Python (e.g., `SELECT snowflake.cortex.complete(...)`). This is a fundamentally different layer from the agentic coding-tool surface and shouldn't be conflated with it. If your work is "embed AI into a query or a data pipeline," this is the native answer.

The native coding analog. Snowflake CoCo (formerly Cortex Code) is Snowflake's own native AI coding agent aimed at data engineering, analytics, and platform operations — usage analytics, cost insights, administration — with deep access to account metadata. It is the in-platform counterpart to Claude Code + MCP. Two things are not established from the available documentation, however: the renamed product's exact interface surface (including whether it itself supports MCP or IDE extensions), and whether CoCo actually *executes* administrative actions such as privilege changes versus merely *surfacing* insights and assisting. Treat its "deep access" as capability described, not autonomous action confirmed.

Governance is a primary dividing line — but the "inside vs. outside" axis is itself fuzzy

This is where the two approaches differ most substantively, though the dividing line is less clean than it first looks. Snowflake Cortex runs AI inside Snowflake's security and governance perimeter, and Snowflake states that data is not sent to Anthropic for training. That is the core architectural distinction on the Cortex side: in-platform inference under Snowflake's controls.

But the axis is contested rather than settled. The managed MCP server's control plane stays inside Snowflake's perimeter, while native CoCo/Cortex tooling now reaches into IDE-style and desktop surfaces — so "native = inside, MCP = outside" is an oversimplification. The cleaner statement is about *where outputs land*, not where the platform branding sits.

In the managed-MCP pattern, governance is enforced at three layers: Snowflake roles and permissions, a semantic view defining what data is visible, and the agent's allowed tool list. Crucially, those controls must be *deliberately configured*, not assumed. The control plane is broader than a single on/off toggle for SQL execution, and even with `execute_sql` omitted, other tools (for example, Cortex Analyst) can still return sensitive result sets to the external client. So "use the MCP path and it's automatically read-only and safe" is not a posture you get for free.

Data egress depends on the mode. The managed server keeps its control plane in-perimeter, but any interaction through an *external* MCP client necessarily returns result sets to that client's environment. The self-hosted open-source server adds another variable, since it runs outside the perimeter entirely. What actually leaves Snowflake therefore depends on which deployment mode you choose, and the sources do not fully characterize the egress in each.

Documented limits and cost considerations

The MCP path is not promised parity. The managed MCP server carries documented constraints not present in native Cortex: tool-count caps, non-streaming responses, and semantic views (not semantic models) as the query surface. Full feature parity for multi-step orchestration and advanced policies is not confirmed. Treat MCP as near-parity access to Snowflake's AI and metadata layers from an external agent, not as end-to-end equivalence with every native workflow.

Cost structure differs, but no clean side-by-side exists. Native Cortex inference is billed through Snowflake credits, and the managed MCP server object adds no new billing line beyond the underlying Snowflake compute, Cortex usage, and storage it triggers. An external Claude client, by contrast, incurs separate Anthropic model charges on top of that Snowflake compute. A definitive side-by-side cost model and any rate or concurrency limits are not laid out in the available sources.

How to decide

Choose **native Cortex** when the AI work belongs inside SQL, pipelines, or the platform itself, and when keeping inference and governance entirely in-perimeter is the priority. Choose **Claude Code + MCP** when you want an agentic coding environment that reasons over both your code and your live Snowflake metadata and execution layer, accepting that you are routing through an external client and therefore must configure the role, semantic-view, and tool-list controls intentionally. The two are not mutually exclusive: an organization can run native Cortex for production, governed, enterprise-wide workflows while individual developers use Claude Code via MCP for exploratory work against the same account. The capabilities overlap heavily because MCP surfaces Cortex; the persona, the workspace, and the governance boundary are what actually differ.

For the nerds

The deeper question is what happens to intermediate state in a multi-step agentic loop. In native Cortex Agent orchestration, intermediate results — retrieved chunks, partial SQL outputs, tool-call responses — stay inside Snowflake's compute boundary. In the MCP path, those intermediates live in the external client's context window. That matters on two fronts: context-window size and token cost become a real constraint on how many Snowflake tool calls you can chain, and any sensitive data surfaced in an intermediate step is now in the client environment, not just the final result set, which is a meaningful difference for data-classification and DLP (data loss prevention) purposes.

The semantic-view constraint on the managed server compounds this. Restricting the MCP query surface to semantic views rather than semantic models means complex analytical queries that depend on semantic-model features cannot be expressed through the managed MCP path, pushing those workloads back to native Cortex Analyst or direct SQL. Whether that is a limitation or a deliberate safeguard depends entirely on how much you trust the external client environment.

WHERE SOURCES DISAGREE

Is "inside vs. outside Snowflake" even the right axis?
One reading treats Claude+MCP as an external developer path and CoCo/Cortex as a strictly in-platform native path. Another account, drawn from Snowflake's own MCP materials, shows the managed MCP server's control plane stays inside Snowflake's perimeter while native CoCo/Cortex tooling now reaches into IDE-style and desktop surfaces — blurring the line
Unresolved: the available sources describe both an in-perimeter managed MCP option and native tooling that extends outward, so a clean architectural dichotomy is not supported

Full feature parity of Cortex Agent capabilities via MCP versus native Snowsight/CoCo.
One reading implies the MCP path "largely re-surfaces" Cortex with near-parity. Another account notes the managed server documents hard limits (e.g., tool-count caps, non-streaming responses, semantic views only and not semantic models on the managed server)
Unresolved: the documentation does not confirm end-to-end equivalence for multi-step orchestration and advanced policies

Whether CoCo can access Snowflake operations or metadata that an external MCP client cannot.
One reading holds that CoCo's native, deep access to account metadata, usage, and cost metrics may exceed what the MCP toolset exposes. Another account suggests the MCP server's enumerated tools cover most of the same ground
Unresolved: the line of demarcation between the two is explicitly not documented in the available sources

Whether CoCo performs administrative actions or only surfaces insights.
One reading describes CoCo as providing natural-language "access to usage analytics, privilege management, and cost optimization," implying it can act. Another account cautions the sources show CoCo surfacing insights and assisting, without clearly asserting it executes privilege changes or writes governance policies autonomously
Unresolved: the source language describes capability and examples but stops short of confirming direct execution

Cost, billing, and rate-limit differences between native Cortex invocation and external Claude-via-MCP usage.
One reading notes native Cortex is billed via Snowflake credits and the MCP server object adds no new billing line beyond underlying Snowflake compute/Cortex/storage. The conflicting reading is that an external Claude client incurs separate Anthropic model charges, and the full financial trade-off and any rate/concurrency limits are not laid out
Unresolved: the sources do not provide a definitive side-by-side cost model

Data-egress posture for the external-client path.
One reading emphasizes that Cortex keeps data in-perimeter and that the managed MCP server reuses Snowflake's governance, implying minimal egress. The conflicting reading notes that any interaction through an *external* MCP client necessarily returns result sets to that client's environment, so egress depends on which deployment mode (managed vs. self-hosted) is used
Unresolved: the sources confirm both an in-perimeter control plane and the inherent flow of outputs to external clients without fully characterizing what leaves the perimeter in each mode