



TECHNICAL WHITE PAPER

AtScale for AI: Technical Overview

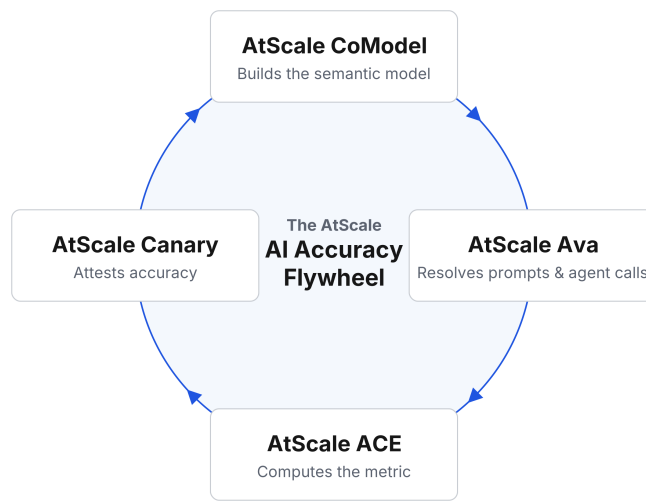
Contents

Introduction	3
The problem: Enterprise AI accuracy, cost and the inability to automate	4
Why common approaches fall short	5
The AtScale AI Accuracy Flywheel	7
 Part 1: How the AtScale loop works	 9
Model: Business context	9
Model: Semantic models	11
Model: AtScale CoModel	13
Measure: Accuracy testing with AtScale Canary	15
Measure: Cost analysis with AtScale Canary	16
Compute: AtScale Ava (Ask, Verify, Answer)	18
Compute: The semantic graph	22
Compute: AtScale ACE, the AI Computation Engine	25
Compute: Aggregates and caching	28
Compute: Determinism	29
Compute: Semantic memory	30
Repeat: AtScale governance	32
 Part 2: Deployment and connectivity	 35
Deployment	35
AI client connectivity	36
Data platform connectivity	36
BI connectivity	37
Development: From business context to production	38
Runtime: The life of one question	38
 Part 3: AtScale AI accuracy, cost and sovereignty	 41
How AtScale makes AI accurate	41
The path to 100%	42
How AtScale lowers AI cost	43
How AtScale makes it safe to automate with AI agents	43
How AtScale protects your AI sovereignty	44
 Conclusion	 44
Notes	45
Index	47

Introduction

Most enterprises aren't getting results from AI yet: only 20% are growing revenue with it,¹ and no more than 10% have scaled AI agents in any business function.² In the enterprise, accuracy is the barrier. An AI answer about revenue, margin or churn has to be exactly right before anyone can act on it. AtScale for AI is AtScale's answer to that problem. It's built on the metric engine AtScale has run in large enterprises for more than a decade, redesigned for AI.

The big idea is a feedback loop: model, measure, compute, repeat. AtScale builds a semantic model of the business, measures the accuracy and cost of AI answers against known-correct results, computes every answer from the model, and turns every miss into a fix for the next turn. Each turn leaves the system a bit more accurate.



Each stage has its own component.

AtScale CoModel, now in private preview, drafts the semantic model from the business context a company already has. It writes the model in SML, AtScale's open-source, YAML-based Semantic Modeling Language, and a modeler approves each change. AtScale Canary, also in private preview, tests AI answers against known-correct results, using the method of the public BIRD-Interact benchmark.³ ACE, the AtScale AI Computation Engine, compiles the model into a semantic graph and computes each answer inside the customer's data platform, including the multi-pass metrics LLMs get wrong. It also builds aggregate tables on its own, so the many queries an AI agent sends don't all scan raw data. AtScale Ava asks what a question means when it's unclear and points AI agents at the right metric. AtScale governance versions every definition like code and carries the company's sign-in and permissions through to every AI agent.

In AtScale's preliminary BIRD-Interact runs, the weakest model with AtScale beat the strongest model on its own, and Claude Sonnet 5 went from 46% of the answerable business questions to 93%. The companion paper, [AtScale's BIRD-Interact Analysis](#), has the full results.⁴ That's not 100%, and it won't stay at 93% unless something keeps testing it. Running the loop

continuously is how an enterprise closes the gap and stays there, without running up warehouse costs.

If you remember one thing from this paper, make it the loop: **model, measure, compute, repeat.**

The problem: Enterprise AI accuracy, cost and the inability to automate

Three problems keep enterprise AI from scaling: accuracy, cost and the inability to automate. Accuracy comes first. A wrong AI answer about revenue or margin usually looks right: the query runs, the number's plausible, and the mistake shows up only after someone has used it.

Cost follows from how those answers get made: an LLM that writes its own SQL spends tokens reading schemas and retrying failed queries, and it sends a new query to the warehouse for every question. Spending's hard to forecast. The inability to automate follows from both. If a company can't measure how often an AI agent's wrong, a person has to review what the agent produces, and the work it was meant to take over stays manual.

All three come from the same source: seven failure points in how AI works with enterprise data, both in the prompts people type and in the requests automated AI agents send.^{3, 4}

- 1. Ambiguous human prompts and agent calls.** Prompts from people and from AI agents are often short and underspecified (“show me revenue,” “which accounts are risky?”). LLM accuracy drops 39% on average when a request arrives underspecified, and models rarely recover from the first wrong reading.⁶ Models seldom ask what was meant. An agent that misreads a prompt carries the wrong result into its next action.
- 2. No systematic semantic definitions.** A metric's formula lives in documentation or in an analyst's head, so the model infers it from column names. A large bank can define revenue four ways (net, recurring, consumption, trading), each in a different system.
- 3. Complex computation.** Joins across tables at different grains fan out and count rows twice. Totals come out inflated. Distinct counts, ratios, semi-additive measures such as balances, and multi-pass calculations are where generated SQL fails most often.
- 4. Inconsistent and complex data.** Even a clean, governed gold layer is complex. The same value can be spelled several ways across sources (“Chicago,” “CHICAGO” and “CHI”),

required fields can be empty, and column names rarely say what they hold, so an LLM grouping or filtering on its own guesses wrong.

5. **Nondeterminism.** The same question can produce different SQL on different runs, so two users get two numbers. Documenting what a term means doesn't fix that, because the model still writes the computation each time. AtScale's fix is in [Determinism](#).
6. **Ungoverned access.** Permissions and audit trails live in the warehouse and the BI tools. An LLM writing its own SQL either bypasses them or needs them rebuilt for each AI agent, and no record shows which definition produced a number.
7. **No accuracy measurement.** Teams rarely test AI answers against known-correct results, so there's no measured error rate, and no alert when a model, prompt or schema change breaks an answer that used to be right.

To be accurate, affordable and safe to automate, AI has to handle all seven at once. A better prompt helps with ambiguity but leaves the math to the model. A catalog adds definitions, and the model still writes the query.

Why common approaches fall short

Teams building AI on business data usually choose one of five architectures. Each one helps with part of the problem, and each leaves the hardest decisions with the model.

Hand-written skills. Skills are files of metric definitions a model loads for a task. Anthropic's data team wrote them by hand and lifted Claude from 21% to about 95% on its analytics evals. As the data model changed, the files drifted, and accuracy fell to about 65% within a month.⁷ Someone has to keep every file current, and the model still writes the SQL.

Plain text-to-SQL. The other common starting point is to point a model at the raw tables and let it write SQL. The model has to work out joins, grain and metric logic from column names on every question, and on BIRD-Interact's 410 business questions no model got much more than a third right this way.⁸

Waiting for a better model. Frontier models keep improving, so it's tempting to wait for the next one. On the same benchmark, adding AtScale lifted every model's first-question accuracy by 55% to 108% on a cold first run, so the weakest model with AtScale, Grok 4.3 at 39.0%, beat the strongest model without it, Gemini 3.1 Pro at 36.1%.⁸ A better model still has to work out the joins and the metric math on every question. With AtScale, it doesn't.

Ontologies, catalogs and knowledge graphs. These describe what business terms mean and what data exists, and they hand that description to the model as context. That helps the model find the right columns. Context doesn't compute anything, though: the model can ignore or misread it, and it still writes the query.

Semantics inside one database. Warehouse- and lakehouse-native semantic features define metrics inside one platform. That's fine if all your data lives there. Most enterprises use more than one, though, and those definitions only work in the platform they're written for. The company ends up keeping a copy of each metric in every platform, each locked to one vendor. The features are also simple by design: each semantic view carries its own copy of the metrics, so a large company ends up with hundreds of overlapping views instead of one shared model.

Each of these improves one input to the model. None of them takes joins, grain and metric logic out of the model's hands at query time.

The AtScale AI Accuracy Flywheel

The seven failure points share a cause: in each one, the model is making a decision it can't make reliably. Improving one input doesn't change that, because the next decision is still the model's. AtScale for AI takes those decisions away from the model and gives each one to a component built for it.

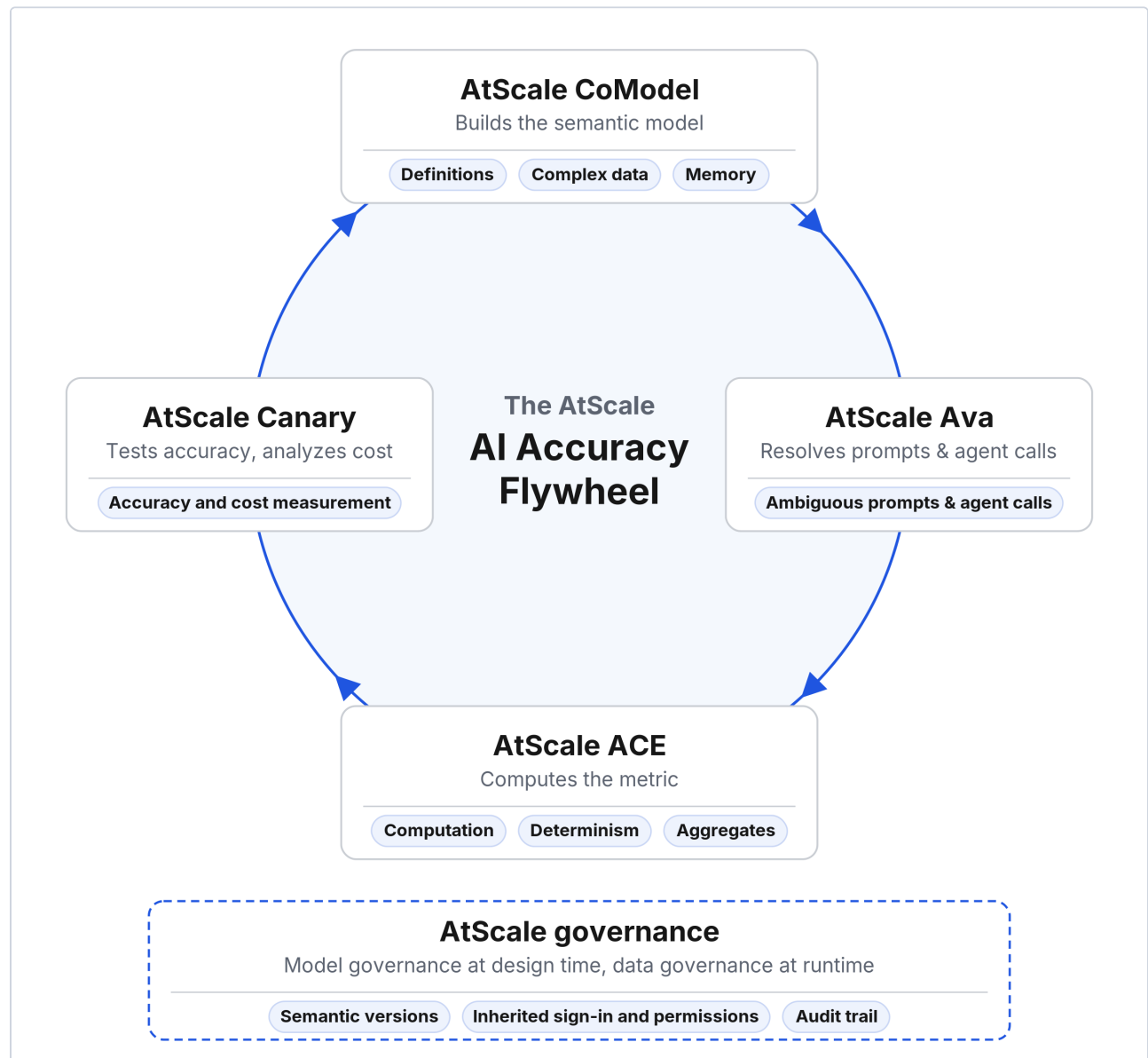


Figure 1. The AI Accuracy Flywheel. Each tag names a failure point, or a capability, from [The problem](#), placed on the component that closes it.

Ava works out what an ambiguous request means before anything runs, whether a person typed it or an AI agent sent it (failure point 1). The semantic model defines each metric in one place

and maps every spelling of a value to one, and CoModel keeps it current with the business, reading semantic memory before every draft (2 and 4). ACE plans every join, grain and pass from the semantic graph and answers repeat questions from aggregates it maintains (3 and 5). AtScale governance applies one set of permissions and logs every query (6), and Canary measures accuracy and cost against gold-standard questions (7).

The AtScale components run as a loop, or flywheel, and that's what's new. An AI system without a test suite answers a question and moves on, with nothing to say whether the answer was right. Here Canary measures every answer against a known-correct result, its misses go back to CoModel as fixes, and the queries that pass become certified queries in semantic memory, so each turn starts from a better model than the last. [Part 1](#) follows the loop one stage at a time, and [Part 2](#) covers deployment, connectivity and two end-to-end walkthroughs.

Part 1: How the AtScale loop works

AtScale for AI runs as a loop with four stages, and this part takes them in order: [model](#), [measure](#), [compute](#) and [repeat](#). Each component in [Figure 1](#) links to the section that covers it, and [↑ Flywheel](#) in any page footer brings you back.

Model: Business context

Ontologies and context engines are organizing tools, and they're useful ones. They describe what business terms mean and how concepts relate, and they hand that description to the model as context. None of that computes anything, though. The model still decides how to use the context and still writes the query, so the result can change from one run to the next. Andrew Brust of Blue Badge Insights calls this kind of context hints: "Hints are great. But hints don't lead to AI accuracy."⁹

AtScale uses business context as input to the semantic model. CoModel reads the context a company already has and turns it into SML objects that ACE computes from:

- **Data sources and schemas:** tables, columns, keys, sample values and, increasingly, column annotations in the database. Even a clean schema is hard to read. A column called `amt_2` doesn't say whether it's net or gross, keys often aren't declared, and one table can have a row per order while the next has a row per line item. Someone who works with the data every day knows all that. A model reading the schema for the first time has to guess.
- **Catalogs, ontologies and glossaries:** business terms, definitions, ownership and the relationships between concepts. They're rarely complete, two glossaries can define the same term differently, and an entry says what a term means without saying which column or formula computes it.

- **Documentation:** metric definitions, runbooks and data dictionaries. It's spread across wikis and shared drives, written for people, and often a version or two behind the systems it describes.
- **BI dashboards and reports:** existing BI models, measures and filters, which show how the business already computes its numbers and which questions it asks of them. They're the closest thing to a working definition, but the logic is written in each tool's own language, such as DAX in Power BI or LookML in Looker. Two dashboards often compute the same metric two different ways.
- **Spreadsheets:** the workbooks teams maintain by hand, with the formulas and lookup tables that never made it into a BI tool. A formula can depend on a hidden tab, a hard-coded adjustment or a mapping one analyst updates every month, and the person who built it may have moved on.

No one source is complete, and they often disagree. CoModel reads them together, flags the conflicts only a person can resolve, and keeps reading as they change, so the model doesn't drift from what the business means. Once a definition's in the semantic model, ACE computes every query from it (see [ACE](#)).

Context from dashboards and spreadsheets

A company's dashboards and spreadsheets are some of its best business context, because they hold two things a schema doesn't: the calculations people already trust, and the questions they already ask. A Power BI report on gross margin carries the measure, the filters and the fiscal calendar Finance signed off on. A spreadsheet that recomputes churn by region each month carries a formula someone has kept by hand for years.

CoModel reads both. The calculations become candidate metrics, dimensions and filters in SML, for a modeler to review. The questions are just as useful: each dashboard shows what the business asks for, at what grain and with which filters, and that becomes a list of questions the semantic model has to answer and Canary can test. When two dashboards compute the same term differently, such as revenue net of returns in one and gross in another, that's a definition to fix in the model and a clarifying question for Ava to ask.

Reports and spreadsheets also hold results, the numbers the business already trusts. Those results can become gold answers for Canary tests (see [Canary](#)), so the questions a dashboard answers today become test cases the semantic model has to pass.

Model: Semantic models

An AtScale semantic model is a set of linked objects written in SML, AtScale’s YAML-based Semantic Modeling Language. Every object has a unique name and a type, plus a plain-language description that AI agents read, and objects reference each other by name.

Object	What it defines	Example
Dataset	A mapping to physical data: a table, a view or a SQL query	The SALES table in the warehouse
Metric	A measure and its aggregation, including distinct counts and semi-additive measures	Revenue, a sum; Customers, a distinct count
Calculation	A derived metric with its own formula, stored in the model	Gross margin %, (Revenue – Cost) ÷ Revenue
Dimension	A business entity and its attributes, with optional hierarchies and drill paths	Geography, with Region › Country › Store
Relationship	How a dataset joins a dimension, by key, including many-to-many	Sales to Geography, on store ID
Model	The top-level object that links metrics, dimensions and relationships into one queryable unit	Finance

Table 1. SML object types, with examples from a retail Finance model, the one [Figure 7](#) follows

Each model lives in a Git repository, one directory per model and one file per object, so changes go through code review like any other software. Teams can package core definitions and share them.

Each model is presented two ways. SQL clients such as Tableau, and AI agents, see one wide, flat table of metrics and attributes, with every column annotated, so they select columns and never write a join. Multidimensional clients such as Excel, Power BI and older OLAP tools see cubes and send MDX or DAX.

An AtScale semantic model is more than a list of definitions. It’s written for AI and BI at once, it describes itself to the AI agents that read it, and it compiles into a graph ACE can plan over. A

definition says what revenue means. An AtScale model also tells ACE how to compute it, at any level, for any question.

Define once, reuse everywhere

SML objects reference each other by name, so a definition written once can be shared down to a single object. A dimension such as Geography or the fiscal calendar, a metric such as Revenue, or a calculation such as Gross margin % lives in one place, and every model that needs it reuses it. When the definition changes, every model that uses it picks up the change.

That matters at enterprise scale. When semantics are defined per report or per view, as they are in Power BI semantic models and warehouse semantic views, definitions multiply into hundreds of near-copies, each with its own version of revenue. Sharing at the object level keeps one version, and it helps the loop: a fix CoModel makes to a shared metric reaches every model that uses it in the same turn.

Design Center

AtScale Design Center is the visual workspace for reviewing and deploying models. It edits the same SML files in the same Git repository, so a change made on the canvas and a change CoModel proposes show up in one history ([Figure 2](#)). Modelers open a model from any branch, check how its tables join, and deploy it. CoModel handles most authoring, and Design Center is where a person looks over the whole model before it ships.

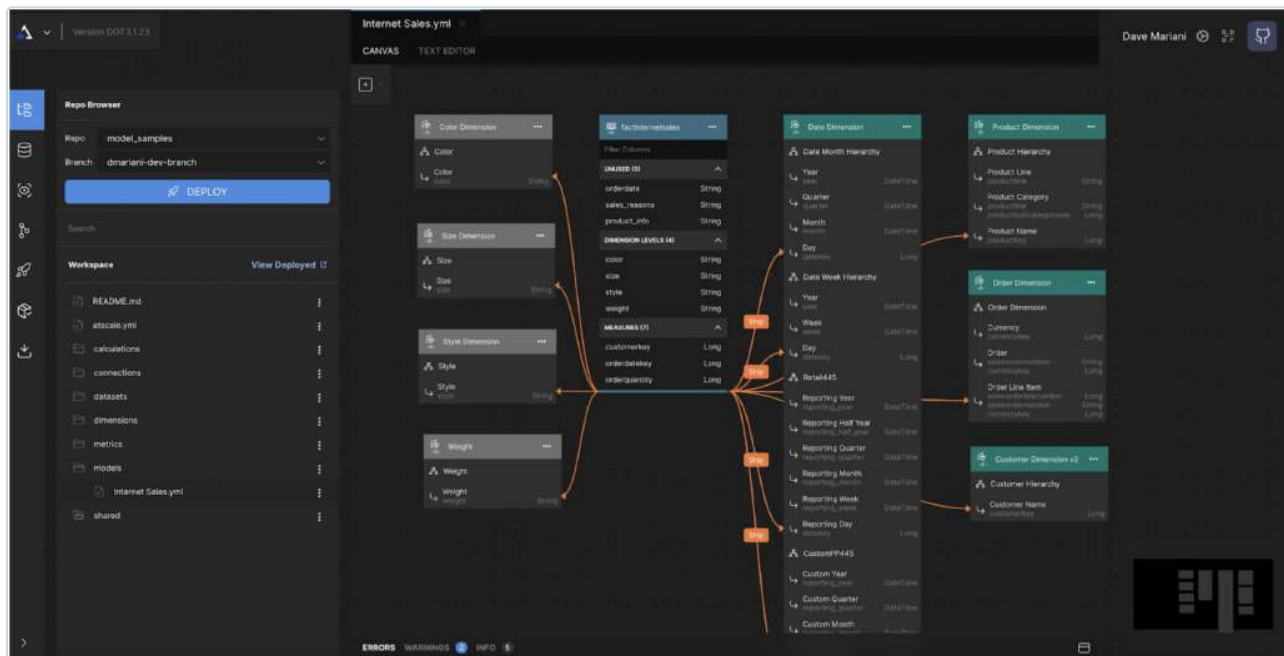


Figure 2. The Design Center model canvas. Blue panels are fact tables, green panels are dimensions and orange arrows are relationships.

Apache Ossie

[Apache Ossie](#), formerly the Open Semantic Interchange (OSI), is a vendor-neutral standard for semantic models written in declarative YAML. The project moved to the Apache Software Foundation in 2026 so that no single company controls it, and more than 50 organizations take part. SML and Ossie describe the same kinds of objects.¹⁰ A shared format lets definitions move between tools. ACE is what computes them.

Model: AtScale CoModel

CoModel is AI-assisted semantic modeling that runs on the customer's own LLM; AtScale doesn't ship or host a model. Its AI agents draft the model from the context a company already has, and the AI data engineer guides the work and approves every change ([Figure 3](#)).

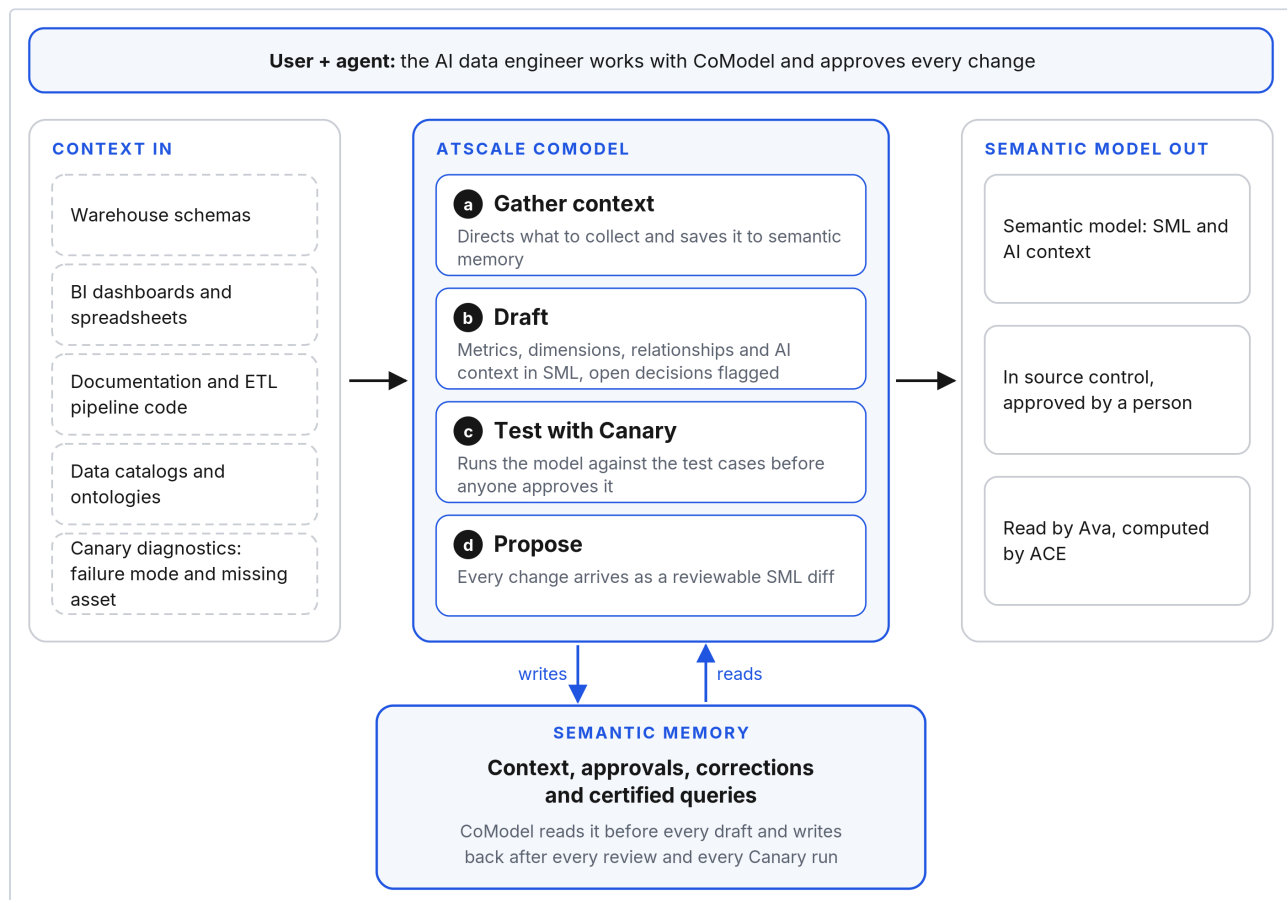


Figure 3. AtScale CoModel. AI agents gather, draft, test and propose, the AI data engineer approves every change, and semantic memory underneath records what each pass learns.

The AI agents work in four steps, lettered to match [Figure 3](#):

- Gather context.** The AI agents collect the [business context](#) above, plus ETL pipeline code, the team’s modeling standards and any Canary diagnostics, and save what they learn to semantic memory.
- Draft.** They write metrics, dimensions, relationships and AI context in SML, and flag the calls only a person can make, such as which of two revenue definitions the business reports.
- Test with Canary.** They run the draft against the test cases before anyone approves it.
- Propose.** Every change arrives as a reviewable SML diff. Nothing ships until the AI data engineer approves it, and approved changes are stored in the team’s Git repository, under AtScale governance, where Ava reads them and ACE computes from them.

CoModel’s much faster than modeling by hand. For BIRD-Interact, a CoModel preview modeled all 22 databases behind the 600 questions in a couple of hours, with object descriptions drawn from BIRD’s own corpus. The AI agents also compare the model with the warehouse

schema and flag drift, such as a renamed or dropped column. Reviewed diffs and runtime query history go back into semantic memory, which the agents read before every draft, and when Canary finds a miss, CoModel turns it into a proposed SML change.

Measure: Accuracy testing with AtScale Canary

Canary gives an AI system something few teams have today: a measured error rate on their own questions, rerun whenever anything changes, so the team knows its accuracy instead of guessing at it. It works the way the BIRD-Interact benchmark does, but runs against enterprise data: the prompts people and AI agents actually send, the company’s own warehouse tables, and gold SQL for each metric computation (Figure 4).

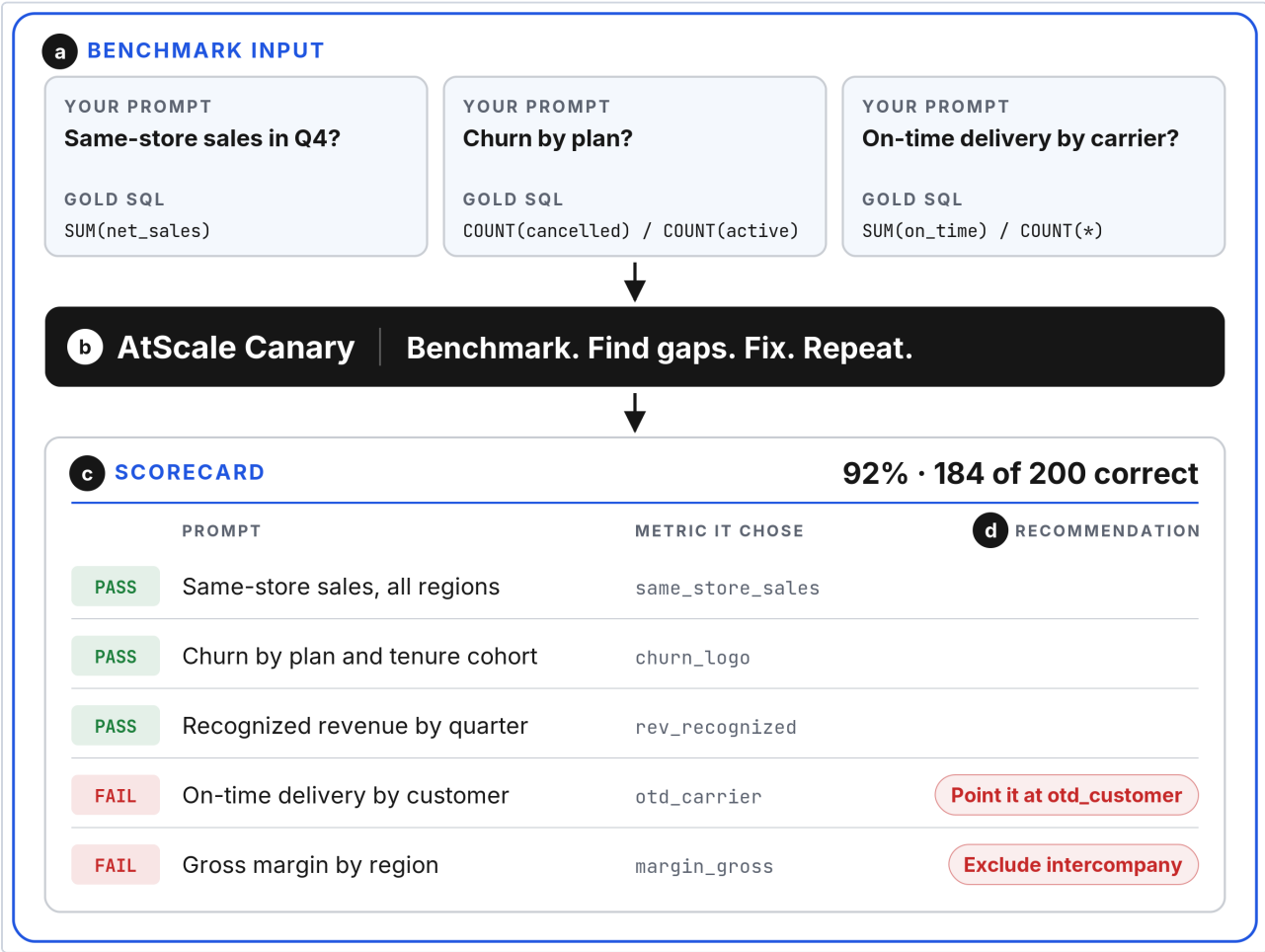


Figure 4. AtScale Canary: (a) questions and gold SQL go in, (b) Canary runs them, (c) a scorecard comes out and (d) each miss gets a recommended fix. Scores are illustrative.

Canary works in four steps, lettered to match [Figure 4](#):

- a. **Benchmark input.** Each test case pairs a prompt, written the way a person or AI agent would send it, with a gold-standard SQL query and the result set it returns. The AI accuracy engineer sets the pass threshold, and prompts from production runs are proposed as new test cases.
- b. **Run.** Canary sends every question through the same path production traffic takes, through Ava or the agent tools and into ACE.
- c. **Score.** Canary compares each result set with the gold result and marks the test pass or fail. The scorecard reports the pass rate for the suite and records which metric the AI selected for every test.
- d. **Recommend a fix.** For each miss, Canary suggests a fix for the data engineer to explore. In [Figure 4](#), “on-time delivery by customer” picked the carrier metric and should point at the customer metric, and gross margin needs intercompany sales excluded. Each miss goes back to CoModel as a proposed fix, and each query that passes becomes a certified query in semantic memory (see [Semantic memory](#)).

When a model, prompt or schema changes, Canary reruns the suite to catch regressions. The output is a measured pass rate, and each failure arrives as a work item with its prompt, the metric the AI selected and a proposed fix.

Measure: Cost analysis with AtScale Canary

Accuracy is half of what a team needs to know before it picks an LLM. The other half is what that accuracy costs. A large frontier model may be needed for multi-step finance questions and unnecessary for a daily sales lookup, and model versions and per-token prices change often.

Canary measures that trade-off. A team defines a test suite for each workload: the prompts the workload sends, each paired with gold-standard SQL. Canary runs the suite through each candidate model and records a pass or fail for every test. Comparing each model’s pass rate with its token cost per run shows the lowest-cost model that meets the workload’s accuracy threshold.

When a provider releases a new model version or changes its pricing, the team reruns the suite and changes models only if the results justify it.

Cost per correct answer tells a team far more than cost per query. A cheap model that answers wrong costs more than it saves once someone acts on the wrong number. Because AtScale does the computing, smaller, lower-cost models can clear the threshold (see [How AtScale lowers AI cost](#)).

What a model comparison shows

AtScale’s BIRD-Interact runs show what a Canary comparison looks like. Seven models from six vendors answered the same business questions through AtScale, and AtScale recorded each one’s accuracy and cost ([Table 3](#)).⁸

Model	Vendor	Accuracy with AtScale	Cost per right answer, without AtScale	Cost per right answer, with AtScale
Gemini 3.1 Pro	Google	62.2%	\$0.74	\$0.63
Claude Sonnet 5	Anthropic	56.6%	\$0.56	\$0.52
Claude Opus 4.6	Anthropic	53.7%	\$0.87	\$0.73
GPT-5	OpenAI	50.5%	\$0.79	\$0.50
Kimi K2.5 (open weight)	Moonshot AI	49.3%	\$0.56	\$0.49
Nemotron 3 Ultra (open weight)	NVIDIA	45.9%	\$0.54	\$0.30
Grok 4.3	xAI	44.6%	\$0.17	\$0.14

Table 3. Accuracy and cost per right answer for seven models in AtScale’s preliminary BIRD - Interact runs⁸

An open-weight model kept pace with a frontier model. Kimi K2.5 came within about a point of GPT-5 on accuracy, at the same cost per right answer. Claude Sonnet 5 was seven points more accurate for a few cents more. Whether that’s worth paying for depends on the workload.

The cheapest model wasn’t the most accurate. Grok 4.3 cost the least per right answer and scored lowest. Gemini 3.1 Pro scored highest, at a higher cost. A daily sales lookup may not need Gemini. A quarterly finance question might.

Gemini asked the best questions. It asked more clarifying questions than any other model. Without a semantic layer, those questions used up its budget. With AtScale, they got precise answers, and Gemini went from last to first on follow-up questions. That's a 345% lift. Where people ask loose, conversational questions, it's the model to test first.

Canary shows which model fits which use case. Finance, sales and support don't ask the same kinds of questions. Canary tags every test with its use case, so its results can show which LLM gets finance's ratios right, which one's cheap enough for sales lookups and which handles vague questions best. A team can then pick a different model for each.

They're preliminary results on a public benchmark, not a recommendation for any model. Canary applies the same method to a company's own questions, and reruns it whenever a model or its price changes.

Compute: AtScale Ava (Ask, Verify, Answer)

Questions reach AtScale by two paths, both through AtScale Ava ([Figure 5](#)). Ava's MCP client sits on both. It resolves a person's prompt from an AI client such as Claude or ChatGPT, and it points an AI agent's request at the right certified metric before the agent calls the tools in Ava's MCP server. On both paths, ACE receives one resolved metric request, computes it, then pushes the SQL down to the data platform.

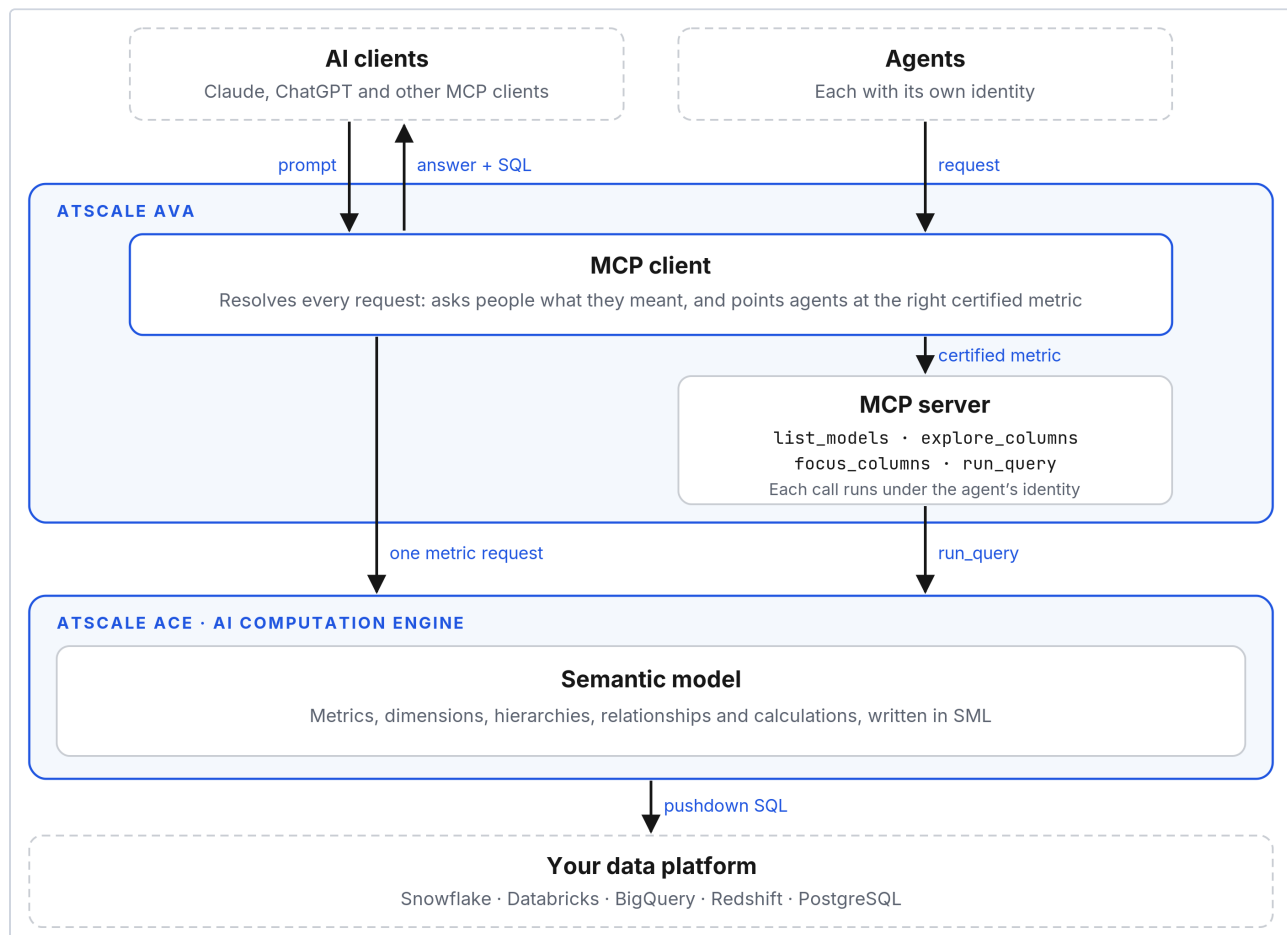


Figure 5. Two paths through AtScale Ava. Its MCP client resolves both people’s prompts and AI agents’ requests, and both paths hand ACE one metric request.

The human path: Ava resolves the prompt

Ava maps a plain-language question to one certified metric before anything gets computed, and asks when a term could mean more than one thing. Asked for “average deal size this quarter,” Ava finds that the model defines average order value three ways (net of discounts for Finance, gross for the order system, net of returns for Sales comp) and asks which one the user means. The answer goes into the model as a rule or synonym, so the same question isn’t asked twice.

The AI agent path: Ava and the agent tools

An AI agent’s request passes through Ava too. Ava matches the terms it uses to certified metrics, and the agent then calls the agent tools: `explore_columns` and `focus_columns` to find them, `run_query` to have ACE compute them, and `get_outbound_queries` to retrieve the SQL behind any answer. Each agent runs under its own identity, sees only the rows its policies allow and never writes its own SQL against raw tables.

Why disambiguation lives in the semantic layer

Disambiguation needs the list of certified metrics a term can map to, with each one's definition. Only the semantic layer holds that list. An LLM on its own sees column names, and a catalog or ontology describes terms without computing them, so neither can tell a person that “deal size” has three valid answers or what each one returns. Ava can ask the right question because the model already knows the possible answers.

Why disambiguation gets simpler on a semantic model

LLM accuracy drops 39% on average when a request arrives underspecified,⁶ and ambiguity causes more than half of the failures in one study of a commercial text-to-SQL system.¹¹

Resolving what a prompt means is one of the biggest things a team can do for accuracy, and the semantic model makes that step smaller ([Figure 6](#)). On one BIRD-Interact insider-trading question, Kimi K2.5 writing against the raw tables produced 21 lines of SQL with four subqueries, five tables and six joins, and each is another place to misread what the user meant. Against the AtScale model, the same question is three lines that name a single metric, its model and a filter.¹²

WITHOUT A SEMANTIC LAYER

AI-generated SQL against the raw tables

Four subqueries, five tables and six joins. Reformatted one clause per line; the model wrote it as 21 lines.

```
WITH event_driven_traders AS (
  SELECT DISTINCT tr.tr_anchor
  FROM trade_records tr
  JOIN corporate_events ce
    ON tr."REC_KEY" = ce."REC_EVT"
),
aggressive_event_speculators AS (
  SELECT edt.tr_anchor
  FROM event_driven_traders edt
  JOIN traders t
    ON edt.tr_anchor = t."TR_KEY"
  WHERE t.trader_fin_data->>'risk_lvl'
    = 'Aggressive'
),
volatile_event_speculators AS (
  SELECT aes.tr_anchor
  FROM aggressive_event_speculators aes
  JOIN trade_records tr
    ON aes.tr_anchor = tr.tr_anchor
  JOIN sentiment_analytics sa
    ON tr."REC_KEY" = sa."REC_SA"
  WHERE ABS(
    COALESCE((sa.sentiment_data->>'news_score')::real, 0)
    - COALESCE((sa.sentiment_data->>'soc_sent')::real, 0)
  ) > 1.0
),
```

```
omi_calculation AS (
  SELECT tr.tr_anchor,
    (ob.order_metrics->>'mod_freq')::real AS mod_freq,
    (ob.order_metrics->>'cancel_pct')::real AS cancel_pct,
  CASE
    WHEN ob.order_metrics->>'cancel_pct' IS NULL
    THEN NULL
    WHEN (ob.order_metrics->>'cancel_pct')::real = 1.0
    THEN NULL
    ELSE (ob.order_metrics->>'mod_freq')::real
      / (1.0 - (ob.order_metrics->>'cancel_pct')::real)
  END AS omi
  FROM volatile_event_speculators ves
  JOIN trade_records tr
    ON ves.tr_anchor = tr.tr_anchor
  JOIN order_behaviour ob
    ON tr."REC_KEY" = ob."REC_NODE"
  WHERE ob.order_metrics->>'mod_freq' IS NOT NULL
)

SELECT AVG(omi) AS average_omi
FROM omi_calculation
WHERE omi IS NOT NULL;
```

WITH ATSCALE

3 lines against the model

One metric, one model, one filter

```
SELECT "Average Order Modification Intensity (OMI)"
FROM "Insider Trading Surveillance"
WHERE "Volatile Event Speculator" = 'Yes';
```

What Ava has to resolve

Which metric: Average Order Modification Intensity, defined once in the model
Which population: the Volatile Event Speculator attribute, defined once
The joins, JSON fields and OMI formula live in the model, so there's nothing else to ask or guess

Figure 6. One BIRD-Interact question, written by Kimi K2.5 against the raw tables and against the AtScale model.

With the model in place, Ava only has to pin down which metric and which population the user means, and it can offer the options the model defines. An AI agent's choice is just as easy: it picks certified metrics by name instead of guessing at joins and formulas. That short clarifying step often decides whether the answer is right.

Compute: The semantic graph

AtScale compiles the semantic graph from the SML files each time a model deploys. It combines a modeling method that records how each metric is built, how each table joins and what one row of each table stands for; the compile step that turns those files into nodes and edges; and a planner in ACE that answers every query by walking that graph. The graph makes the decisions an LLM gets wrong when it writes its own SQL, and it makes them once ([Figure 7](#)).

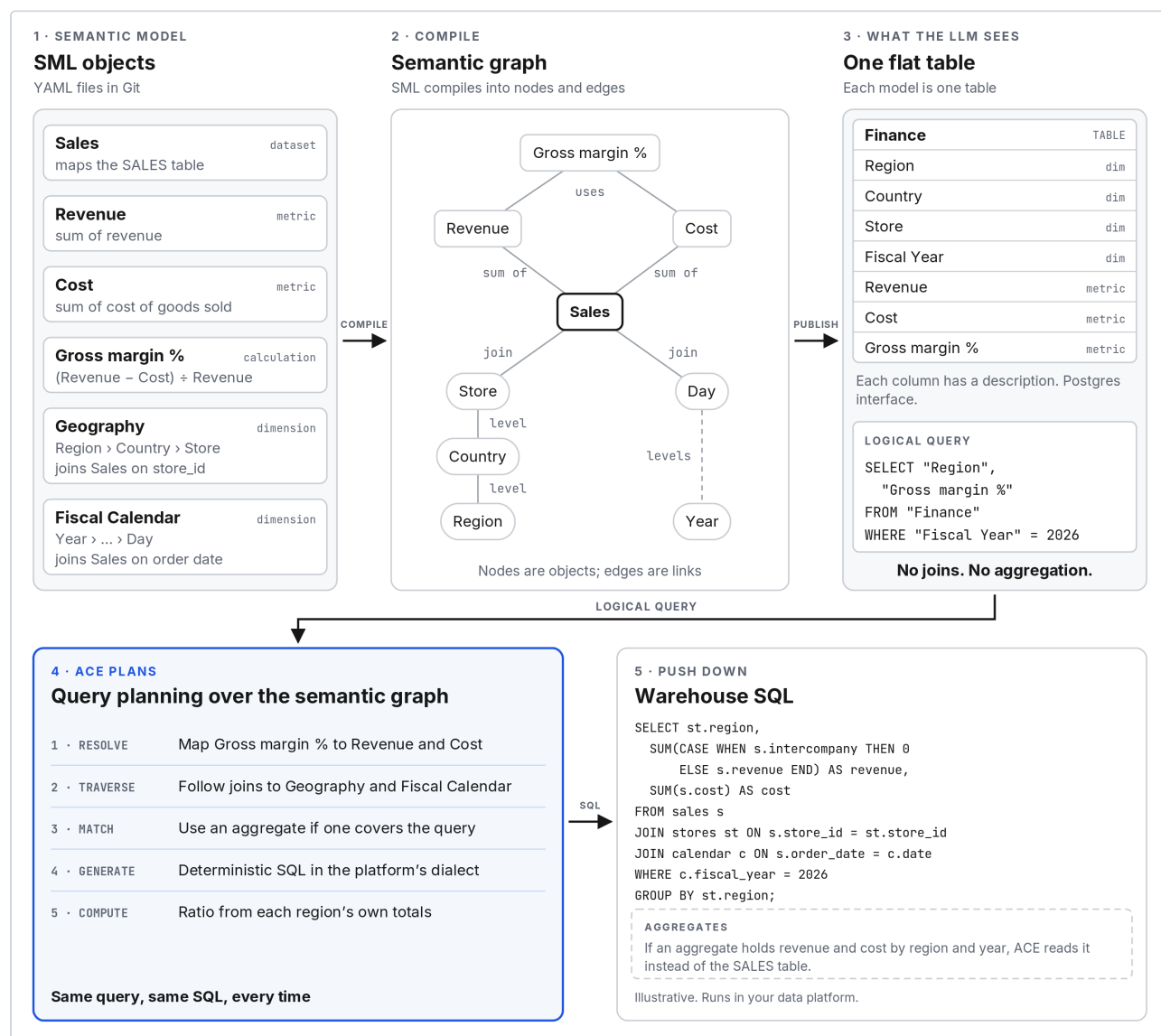


Figure 7. How one question uses the semantic model: the SML compiles into a graph, the LLM sees one flat table, and ACE plans over the graph. The SQL is illustrative. [Appendix A](#) has it at full page size.

An LLM writing SQL has to work out, on every question, how the tables join, what one row of each table stands for and which metrics need more than one pass. Those three decisions cause most wrong numbers ([failure point 3](#)), and a model can make them differently each time it runs. The graph decides all three once, at deploy time. ACE reads the answers from it instead of inferring them, and the LLM doesn't have to make them at all. The rest of the architecture depends on the semantic graph: every approach in [Why common approaches fall short](#) leaves these three decisions to the model, and the graph is where AtScale takes them away.

Objects become nodes (datasets, metrics, calculations, dimensions, hierarchy levels), and links become edges (relationships, formula references, roll-ups). The LLM doesn't see that graph: it sees one flat Finance table and writes a two-column logical query. ACE plans over the graph,

resolving Gross margin % to Revenue and Cost, following the relationships to geography and the fiscal calendar, and writing the joins, the intercompany exclusion and the per-region ratio in the platform’s own dialect. The model carries the business logic, so the LLM has almost nothing to get wrong.

Joins that don’t double count

Some tables have one row per order. Others have one row per item on an order. When a query combines the two, each order’s total gets copied onto every one of its item rows.

Order	Item	Order total
1001	Shirt	\$100
1001	Hat	\$100
1001	Socks	\$100

Add up that last column and a \$100 order becomes \$300. The query still runs and returns a believable number, with no error to flag it. LLMs writing raw SQL make this mistake often, because nothing in the column names warns them.

The semantic model records which tables have one row per order and which have one row per item. Data teams call this the table’s grain, and every relationship in the semantic graph carries it.

ACE uses the grain to decide the order of operations. Ask for revenue and units sold by region, where revenue comes from the orders table and units come from the order-lines table. ACE doesn’t join the two tables first and add up afterward. It plans two passes instead:

1. **Total each table at its own grain.** ACE sums order totals by region from the orders table, and units by region from the order-lines table, as two separate subqueries.
2. **Join the totals.** It joins the two small results on region, where each side has exactly one row per region, so nothing gets copied.

Each order’s \$100 is counted once, whatever columns the question combines. The LLM never writes the join, and ACE builds the same plan for every query that mixes tables of different grains.

Planning is a walk over the graph

A logical query names a few nodes, usually a metric and a dimension, sometimes with a filter. ACE starts from those nodes and follows the edges to everything else the query needs: the datasets behind each metric, the join path to each dimension, the hierarchy level to group by and the calendar to filter on. For gross margin by region in fiscal 2026, the walk starts at Gross margin % and finds the two metrics it's built from, Revenue and Cost. Both are sums over the Sales table, so the walk goes there next. From Sales it follows one join to Store, then up the hierarchy to Region, and another join to Day, then up to Fiscal Year. The model defines each of those paths.

The graph also tells ACE when a metric can't be computed in one pass, such as a ratio, a distinct count or a balance. [Metrics LLMs get wrong](#), under ACE, covers those metrics and walks through a three-pass plan.

Why the LLM sees a flat table

AtScale flattens the graph into one wide table of metrics and attributes before the LLM sees it. Each part of the graph the LLM can't see is a decision it can't get wrong: which tables to join, at what grain, with which aggregation, over which calendar. The LLM picks columns and filters, and ACE computes the metric.

Compute: AtScale ACE, the AI Computation Engine

ACE computes the metric a person or an AI agent asks for, from the definition in the semantic model, inside the customer's data platform, the same way every time. ACE runs the calculation itself, including multidimensional, time-series, semi-additive and hierarchical metrics. [Figure 8](#) shows its parts, and [The semantic graph](#) shows how ACE plans over it. It also builds its own aggregate tables (see [Aggregates and caching](#)).

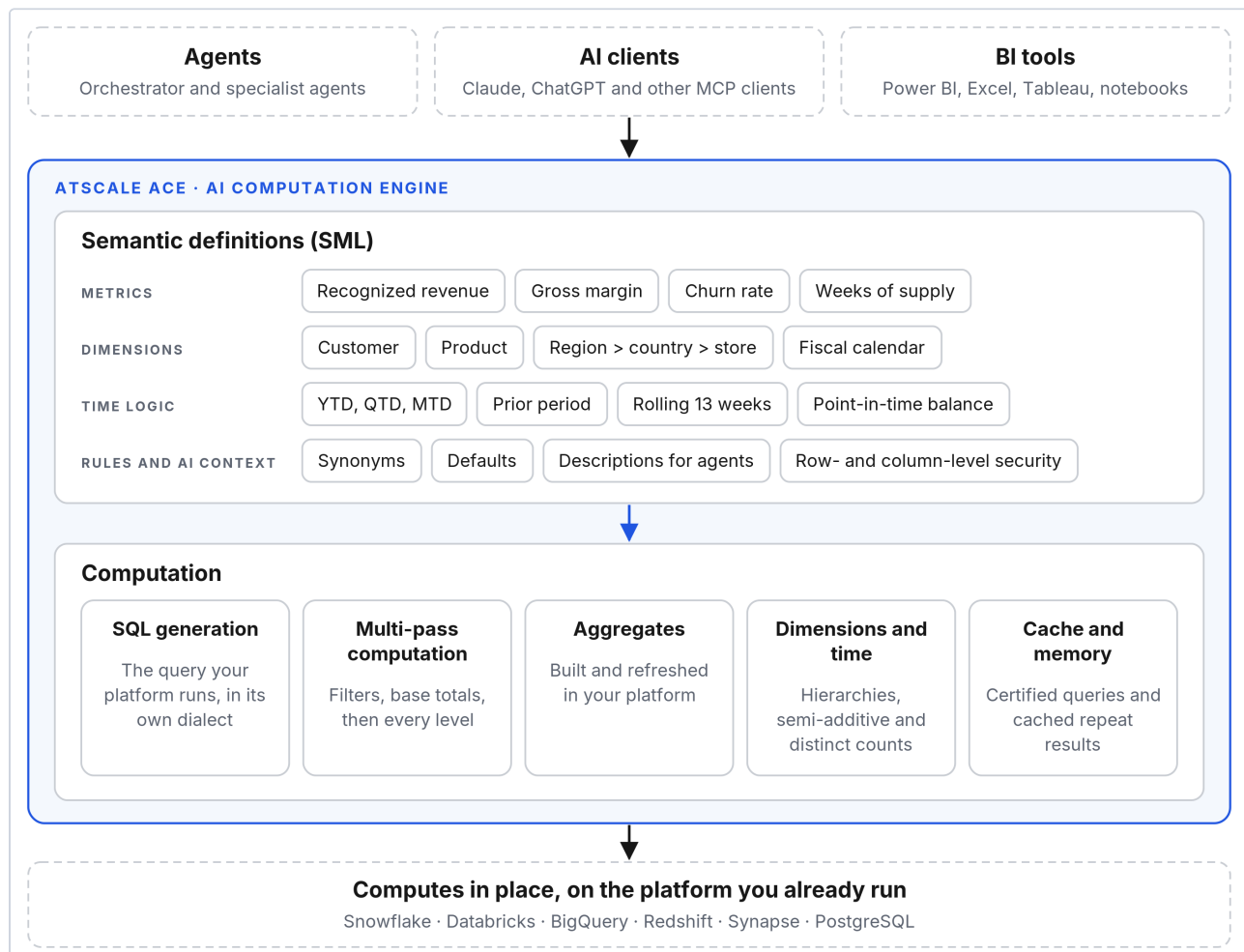


Figure 8. Inside ACE. One set of semantic definitions, computed the same way for every caller.

How ACE processes a request

1. **Validate.** ACE checks the request against the model before anything runs. A metric and a dimension that don't join return an error, with no warehouse cost.
2. **Plan.** ACE walks the semantic graph to find the join paths and grain the requested columns need. The plan is deterministic: a given logical query always produces the same physical query, and AtScale's query log records both.
3. **Route.** ACE answers repeat questions from cache and sends everything else to the smallest aggregate that can answer it. If no aggregate fits, the query runs against the base tables.
4. **Generate and push down.** ACE writes SQL in the platform's own dialect, tuned for that platform. AtScale enforces model permissions and row security on every query, and where the platform supports user impersonation, the query runs as the requesting user.

5. **Return.** The result goes back with its definition and the SQL that produced it, so any answer can be traced.

Metrics LLMs get wrong

Many enterprise metrics can't be computed in one pass, because the answer depends on a number that has to be computed first. Take a school with two classes. Ten students averaged 80 on a test and thirty averaged 60. The school average is 65: add all 40 scores, then divide. Averaging the two class averages gives 70, a plausible number that's wrong, and nothing in the output says so. Gross margin, average order value and average visit satisfaction work like the school average, as does any ratio.

Other metric types have their own rules. A distinct count is recounted at each level, since a customer who bought on Monday and on Tuesday shows up in both daily counts. A balance takes the last value in the period, so a quarter's bank balance is the March 31 balance, not the sum of three month-ends. Real metrics combine these rules: none of the nine metrics in AtScale's *Nine Industries, Nine AI Agents* ebook is a column with a SUM around it.¹³ Net interest margin, for one, puts an averaged balance under a ratio.

ACE reads these rules from the semantic graph (see [The semantic graph](#)) and plans the passes. Gross margin by region for fiscal 2026, the question in [Figure 7](#), takes three ([Figure 9](#)): keep the fiscal 2026 orders and zero out intercompany sales, total revenue and cost by region, then divide each region's totals.

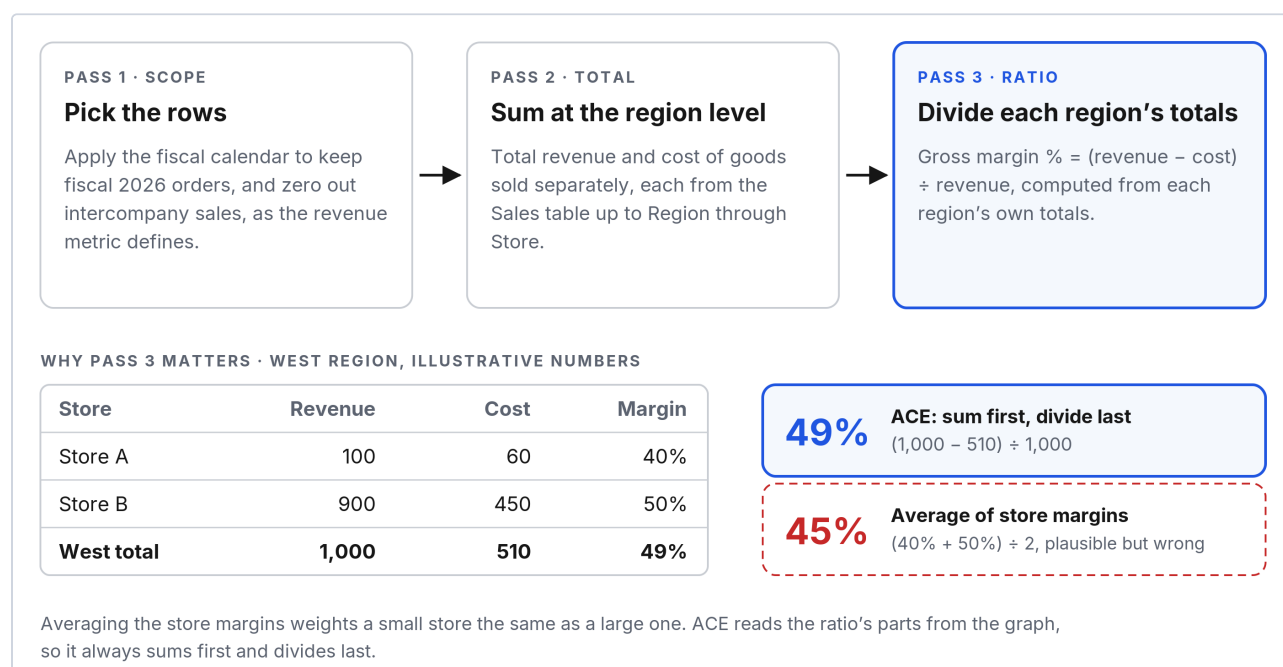


Figure 9. A three-pass plan for gross margin by region. Summing first and dividing last gives 49%; averaging the store margins gives a plausible 45%. Numbers are illustrative.

Dimensional structure and time series

The model declares how every measure behaves, and ACE follows it at every level. Hierarchies and drill paths, such as region to country to store, roll up from the model's relationships.

Time logic lives in the model too: year-, quarter- and month-to-date, prior period, rolling windows, point-in-time balances, fiscal and retail 4-5-4 calendars, and currency conversion. When someone changes the time window, ACE recalculates, and no pipeline or skill file needs rebuilding.

Compute: Aggregates and caching

Aggregates are one of the clearest differences between ACE and a semantic layer that only stores definitions. A definitions layer hands each query to the warehouse as written, so every question runs against the base tables. ACE decides, query by query, whether a smaller table it already built can answer it, and it builds that table when the workload calls for one. That matters more with AI. An AI agent often sends several queries to answer one question, and each of them can be served from an aggregate instead of a scan of raw data.

ACE builds aggregate tables inside the customer's platform. Demand-based aggregates come from the query patterns that actually arrive, predictive aggregates from the model's design, and user-defined aggregates from the modeler. They refresh on a schedule, on an ETL trigger or through the REST API, either incrementally or in full, and ACE retires them as the workload shifts. Each aggregate is stored as a set of dimensions and metrics, so one aggregate can serve many different queries. Repeat questions come back from cache.

An aggregate table stores some of the graph's metrics at some of its dimension levels, such as revenue and cost of goods sold by region and fiscal year. When ACE plans a query, it checks whether an aggregate covers every node the walk touches. If one does, ACE reads the aggregate instead of the fact table. The answer doesn't change, because the aggregate was built from the same graph.

The aggregate system has run in production for 13 years, and it does more than store summaries. ACE builds new aggregates from existing ones instead of rescanning base tables, and it builds dimensionally modified aggregates that make time-series calculations faster and cheaper.

A cache replays an answer someone already ran. ACE computes any valid combination of the model's metrics and dimensions, including combinations that haven't been queried before. Caching and aggregates keep it from recomputing work it's already done, which cuts warehouse compute and the tokens an AI agent would otherwise spend retrying queries against raw tables.

ACE doesn't move the data. Raw data stays in the customer's platform, and aggregates live in a schema the customer controls. [Data platform connectivity](#) lists the supported platforms.

Compute: Determinism

LLMs are probabilistic. Ask one the same question twice and it can write two different SQL queries, so two people asking the same thing can get two different numbers (failure point 5). That's fine for drafting an email. It isn't fine for a revenue number. Enterprises need a deterministic system, where the same question always returns the same number. AtScale keeps the two apart.

Where the probabilistic step ends

One step stays probabilistic: the LLM choosing which certified metrics, dimensions and filters a question needs. Ava narrows even that step, asking when a request could mean more than one thing and saving the answer as a rule (see [Ava](#)). Everything after that choice is deterministic. The semantic graph fixes the joins, grain and passes (see [The semantic graph](#)), and ACE compiles the same logical query to the same SQL every time (see [ACE](#)).

What deterministic means

The same logical query always produces the same SQL, so on the same data it'll return the same number. When the data changes, the number changes for that reason alone, and the query log shows the SQL behind both. Determinism doesn't mean the LLM always picks the same columns for a loosely worded question. That's the step Ava and the semantic model work to narrow.

Why it matters

Determinism is what lets a business rely on AI. Two people asking for gross margin by region see the same number, today or next month. An auditor can rerun a reported figure from the logged SQL and, on the same data, get it again. An AI agent can act on a number without a person rechecking it, because the number won't change unless the data does.

It's also what makes the loop work, because a query Canary certifies today returns the same answer tomorrow, so semantic memory can safely reuse it.

Compute: Semantic memory

Semantic memory remembers queries that produced correct results and offers them when a similar request arrives. It doesn't fill the model's context window, and it needs no retraining and no change to the model or engine. Accuracy rises with use, and AI agents take fewer steps to get there.

1. **Record.** For each question, AtScale stores the question and the logical query generated for it.

2. **Certify.** AtScale scores each stored query. One confirmed as correct becomes certified, and the rest stay candidates.

3. **Reuse.** When an AI agent takes on a similar question, semantic memory offers the certified query, so the agent starts from one that's already known to work.

One BIRD-Interact question shows how (Figure 10).⁵ On an insider-trading task, the right answer depends on a pair of filters that the simulated user only gives when asked. In the cold first run, with memory empty, Claude Sonnet 5 needed nine tool calls, a clarifying question to the simulated user and 12.5 BIRD COINS to get it right. That correct answer certified the query. In runs 2 and 3, semantic memory offered the certified query, and the AI agent used the same filter pair the user had chosen in run 1. It skipped the clarifying question and the test query, answered in five calls for 8.5 coins.

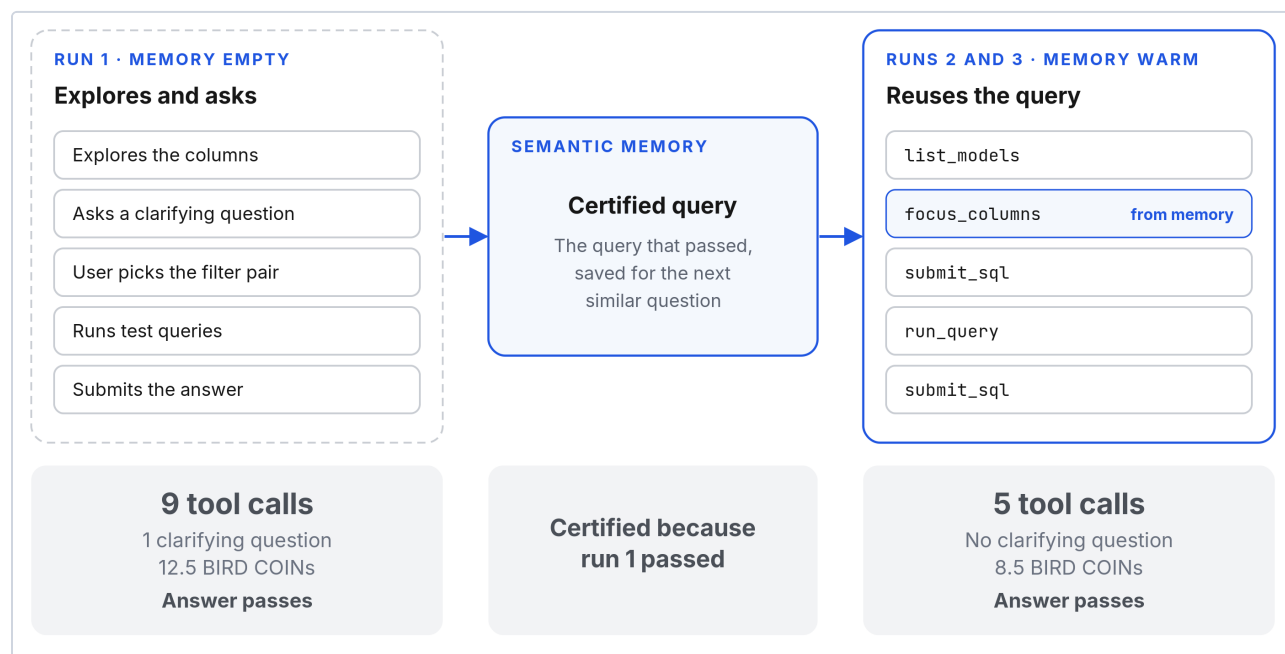


Figure 10. One insider-trading task across three runs. The cold run's passing answer becomes a certified query, and the warm runs reuse it.

In the benchmark, a query is certified when the grader passes the answer, so AtScale reports the cold first run beside every result. In production, Canary tests supply that certification. [AtScale's BIRD-Interact Analysis](#) shows how accuracy rose across runs as memory filled.

Repeat: AtScale governance

AtScale governance works at two levels, and AI needs both. Model governance, at design time, controls how a definition changes: what revenue means, who approved it and when. Data governance, at runtime, controls which rows a person or an AI agent sees when they ask a question. [Figure 11](#) shows both.

Model governance: How definitions change

Model governance treats every definition like code. When what revenue means changes, the change arrives as a reviewed pull request, versioned in Git, and every client, from Claude to an AI agent to Power BI, gets the same certified definition and the same security rules ([Figure 11](#)).

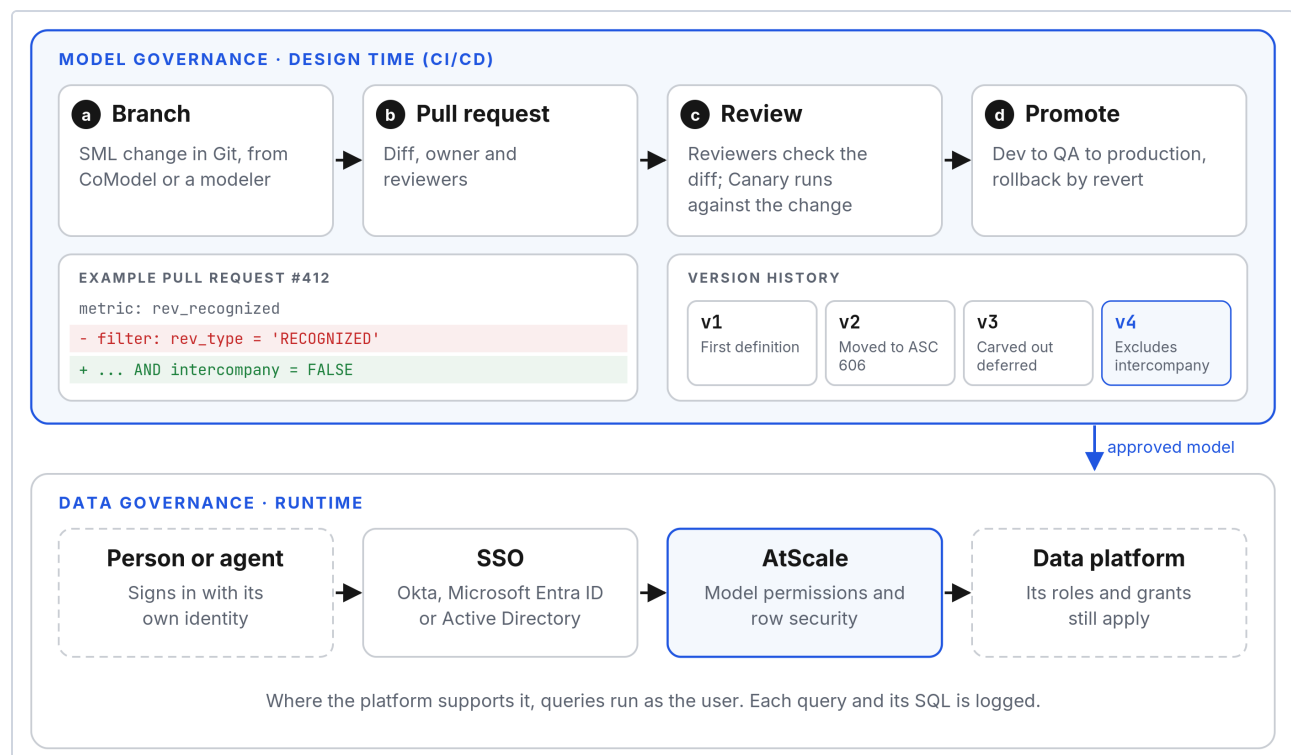


Figure 11. AtScale governance: model governance at design time (a to d) and data governance at runtime.

Changes move through four steps, lettered to match [Figure 11](#):

- a. **Branch.** CoModel or a modeler edits SML on a branch of the model's Git repository. Test cases and semantic memory are versioned with the model, so they stay in step.
- b. **Pull request.** The change arrives as a diff with an owner and reviewers. In [Figure 11](#), pull request #412 adds `intercompany = FALSE` to the filter on recognized revenue.
- c. **Review.** Reviewers check the diff before it merges, and Canary runs the test suite against the changed model when it's available.
- d. **Promote.** An approved change moves by branch from dev to QA to production. Each metric keeps its version history, and a bad change is rolled back by reverting it. In this paper, a certified metric is one that's defined in the model and merged to production after review.

Data governance: Who sees what at runtime

For an enterprise, data governance decides whether AI can touch production data at all. Security teams have spent years on identity, roles, grants and row-level rules in the data platform. An LLM that writes its own SQL either goes around them or forces a team to rebuild them for every AI agent (failure point 6). AtScale connects the two: a request arrives in business terms and reaches the warehouse as SQL that runs under the rules the company already has.

Nothing in that security model moves. People keep signing in through the company's identity provider, such as Okta, Microsoft Entra ID or Active Directory, over OIDC, SAML or Kerberos. Production deployments require one. The data platform's roles and grants still apply: where the platform supports user impersonation, ACE runs each query as the requesting user, so a query can't return rows that user couldn't read in the warehouse. AtScale logs each query and the SQL it sent, so a number traces back to the definition and the SQL that produced it.

On top of the platform's rules, AtScale applies model permissions and row security to every query from every client. An analyst in Excel, a dashboard in Tableau and an AI agent asking the same question see the same rows, because the rule lives in one place instead of in each tool.

AI agents follow the same rules as people. Each agent signs in with its own identity, calls certified metrics through agent tools that only run SELECT queries against one model at a time, and sees only the rows its policies allow. Adding an agent doesn't mean writing a new security policy for it.

It also leaves the company free to choose. The rules stay in the company's identity provider, data platform and semantic model, so switching LLMs or adding an AI tool doesn't hand the security model to an AI vendor or mean rebuilding it (see [How AtScale protects your AI sovereignty](#)). [Deployment](#) covers what an installation needs.

Part 2: Deployment and connectivity

This part covers how AtScale runs in a company’s environment: deployment, connectivity, and two walkthroughs of the loop, one at design time and one at runtime. [Appendix B](#) has the full reference architecture.

Deployment

AtScale runs in the customer’s own environment, and raw data never leaves the customer’s data platform. There are three ways to run it ([Table 2](#)).

Option	Where it runs	Notes
Self - hosted on Kubernetes	The customer’s cloud account or data center	Installed with a Helm chart, which handles scaling and failover
Google Cloud Marketplace	The customer’s Google Cloud project	Deployed as a Kubernetes application from the marketplace
AtScale for Snowflake	The customer’s Snowflake account	Runs as a single -tenant Snowflake Native App, bought through Snowflake Marketplace

Table 2. AtScale deployment options

- **What gets installed.** ACE, which plans and computes every query, Design Center, the visual workspace for models, and the AtScale MCP server, which connects AI clients and AI agents. No LLM is included: AtScale works with the models the customer already uses.
- **What it needs.** A supported data platform, an identity provider such as Okta or Microsoft Entra ID, and a Git repository for the models. The [system requirements](#) page has the details.
- **What AtScale keeps.** Model versions, query history and the aggregates ACE builds, in a schema the customer controls. It doesn’t keep a copy of the raw data.
- **Editions and pricing.** AtScale comes in Standard and Enterprise editions, through Snowflake Marketplace and Google Cloud Marketplace. Enterprise is priced by deployed semantic objects (DSOs), the metrics, dimensions and models published to production. Users are unlimited, and there are no per-seat or per-query fees.¹⁴

AI client connectivity

The AtScale MCP server lets any MCP client ask a published model questions in plain language. Each query runs under the user's own permissions, and the server can show the exact SQL that ran. The [MCP server documentation](#) covers setup and the full tool list.

AtScale's BIRD-Interact runs (see [AtScale's BIRD-Interact Analysis](#)) tested seven models from six vendors through the same MCP server:

- **Anthropic:** Claude Sonnet 5 and Claude Opus 4.6 ([AtScale and Claude](#))
- **Google:** Gemini 3.1 Pro
- **OpenAI:** GPT-5 ([AtScale and OpenAI](#))
- **xAI:** Grok 4.3 ([AtScale and Grok Build](#))
- **NVIDIA:** Nemotron 3 Ultra, an open-weight model ([AtScale and NVIDIA](#))
- **Moonshot AI:** Kimi K2.5, an open-weight model

The [full list of AI tools](#) covers the rest. Because models are written in SML, definitions also move between tools and platforms in an open format.

Data platform connectivity

ACE computes every query inside the customer's data platform, in that platform's own SQL dialect, so the data stays where it is. Supported platforms include:

- Snowflake
- Databricks
- Google BigQuery
- Amazon Redshift
- PostgreSQL

The [supported tools and platforms](#) page and the [data platforms and BI tools](#) page have the full list.

BI connectivity

The model that answers AI clients also serves pivot tables, and it's the same definition and the same number in each one ([Figure 12](#)). There aren't extracts to schedule or per-tool models to rebuild. Excel and Power BI connect live over XMLA, Tableau and Looker connect over SQL, and notebooks use the Python interface. AtScale also serves Google Sheets, ThoughtSpot, MicroStrategy, Superset, Klipfolio and Hex.

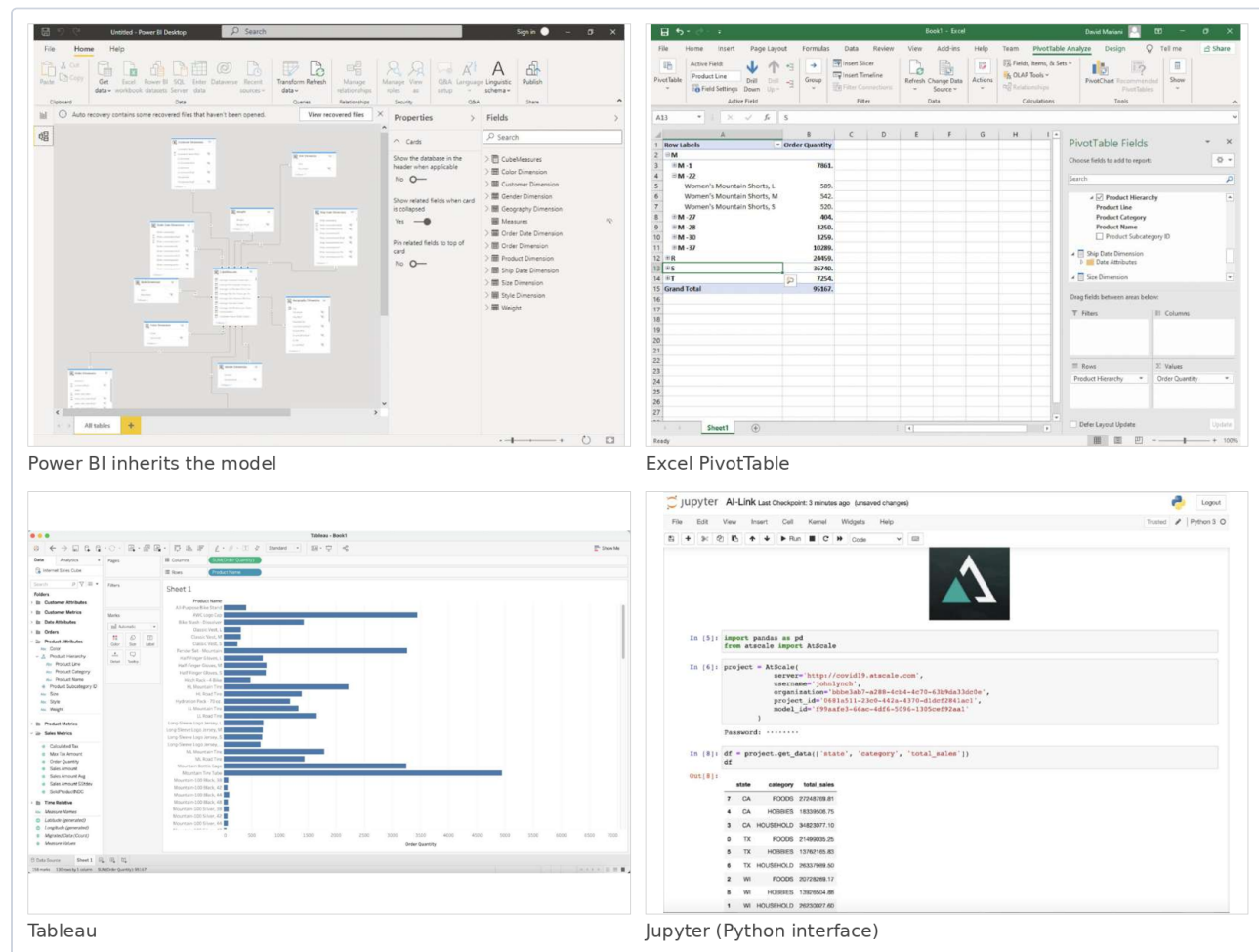


Figure 12. One model, four tools: Power BI, Excel, Tableau and a Jupyter notebook all query the same AtScale model.

Snowflake Semantic Views and the XMLA endpoint

Snowflake Semantic Views store business definitions inside Snowflake, and Snowflake's AI features, including Cortex Analyst, read them. Excel and Power BI send MDX and DAX over XMLA, so they can't query a Semantic View directly. The Snowflake XMLA endpoint, built on ACE, translates those queries into Snowflake SQL and runs them against the Semantic View,

with no extract to refresh. AtScale Enterprise takes the same engine to other platforms, BI tools and AI agents. snowflake.at-scale.com has the details and a side-by-side comparison.

Development: From business context to production

Development runs the same loop before anything reaches production ([Figure 13](#)). CoModel drafts from the business context, Canary tests the draft and sends each miss back as a fix, and a modeler reviews what passes before it ships through AtScale governance. When a model, prompt or schema changes, the loop turns again.

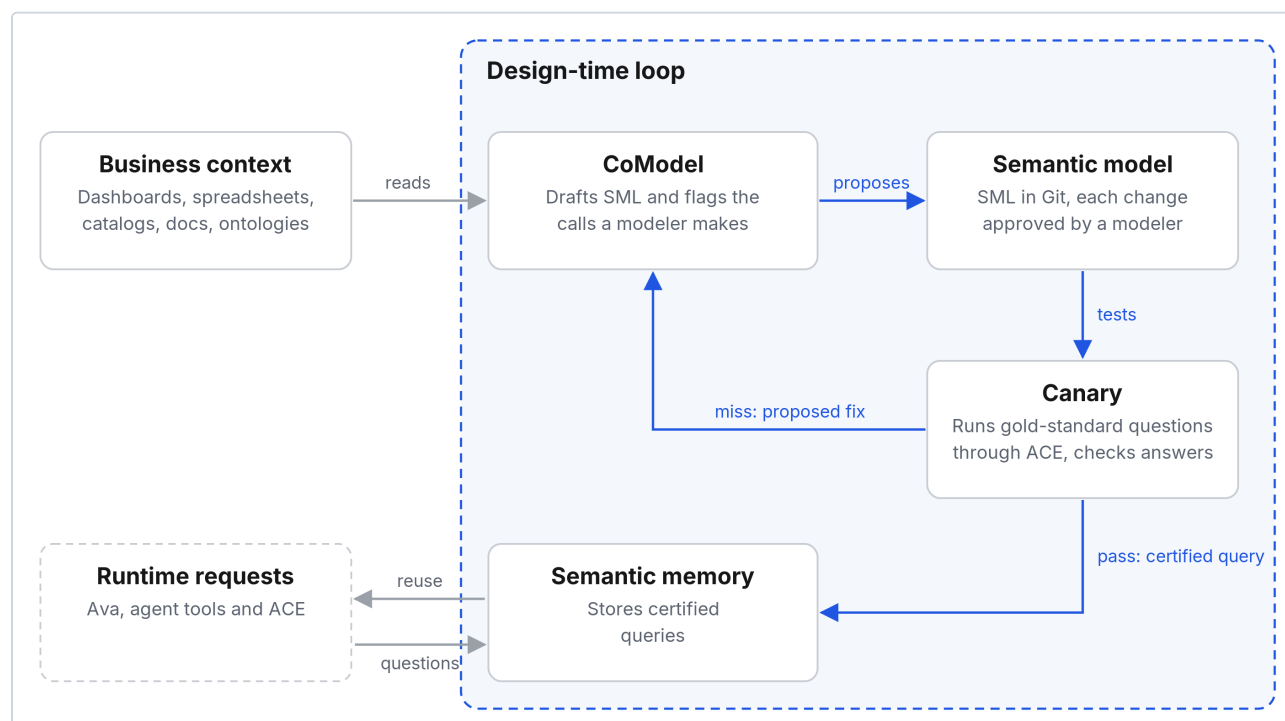


Figure 13. The design-time loop. Misses go back to CoModel as fixes; passing queries become certified queries in semantic memory.

Runtime: The life of one question

This section follows one question from prompt to answer. A finance manager types into an AI client, “What was our gross margin by region this year?” It’s the question [Figure 7](#) traces through the semantic graph.

1. **The question arrives.** The AI client sends the prompt to the AtScale MCP server, where Ava receives it. An AI agent’s request comes through Ava the same way; Ava matches it to the certified metric, and the agent picks up at step 3.

2. **Ava disambiguates the question.** “This year” could mean the calendar year or fiscal 2026, and the model defines both. The AI client stops and asks the manager which one before anything runs. The manager picks fiscal 2026, and Ava saves the answer as a rule so the next person isn’t asked.
3. **The LLM finds the metric in the semantic layer.** AtScale presents the Finance semantic model to the LLM as one flat table. Every column in it is a metric or dimension defined in the model, not a raw warehouse column. The LLM calls `explore_columns`, which returns Region, Fiscal Year and Gross margin %, each with its model description. If a similar question has passed before, semantic memory offers the query that worked, and the LLM starts from it.
4. **The LLM writes a logical query.** It selects two columns and filters on one, with no joins and no aggregation, and `run_query` hands it to ACE:

```
SELECT "Region", "Gross margin %"
FROM "Finance"
WHERE "Fiscal Year" = 2026
```

5. **ACE validates and plans.** ACE checks every column against the model, then walks the semantic graph. Gross margin % is built from Revenue and Cost, both are sums over the Sales table, Sales joins Store and rolls up to Region, and it joins Day and rolls up to Fiscal Year (see [Planning is a walk over the graph](#)). If an aggregate already holds revenue and cost by region and fiscal year, ACE plans to read it instead of the Sales table.
6. **ACE computes in three passes.** Gross margin is a ratio, so it can’t be summed or averaged across stores. ACE keeps fiscal 2026 orders and zeroes out intercompany sales, totals Revenue and Cost by region, then divides each region’s totals ([Figure 9](#)). Averaging the store margins instead would return a believable wrong answer: 45% for the West region instead of 49%.
7. **The platform runs the SQL.** ACE writes the three passes as SQL in the platform’s own dialect, usually as one statement ([Figure 7](#), panel 5). It applies row security and, where the platform supports impersonation, runs the query as the manager. The data never leaves the customer’s warehouse. ¹⁵

8. **The answer comes back with its SQL.** The AI client gets one row per region, along with the metric's definition, and `get_outbound_queries` returns the exact SQL that ran. AtScale logs the query and its SQL, so the number can be traced later.

9. **Memory learns.** Once a Canary test certifies the answer, the query joins semantic memory, and the next question about margin by region starts from a query that's known to work.

Over the whole trip, the LLM chose only which columns to select and which year to filter on. The model supplied everything else, from the joins and the grain to the three passes and the SQL, and it supplies the same ones on every run. The LLM picked the metric, ACE computed it, and semantic memory kept the query that worked.

Part 3: AtScale AI accuracy, cost and sovereignty

AtScale for AI is designed to get the number right without overspending. Accuracy and cost improve together, because the LLM makes fewer decisions and runs fewer queries.

How AtScale makes AI accurate

[Table 4](#) maps each failure point from the problem section to the AtScale component that addresses it, and to the section that explains how. In AtScale’s BIRD-Interact analysis, the misses that remain fall under failure points 1 to 4, and [The path to 100%](#) covers how the loop closes them.

Failure point	AtScale fix	Section
1. Ambiguous human prompts and agent calls	Ava asks one clarifying question before computing, and the answer becomes a rule. AI agents call certified metrics by name.	Ava
2. No systematic semantic definitions	Each metric is defined in one place in the semantic model, in SML. CoModel drafts the model from existing metadata.	Business context , Semantic models , CoModel
3. Complex computation	ACE plans joins, grain and multi-pass math over the semantic graph, the same way on each run.	The semantic graph , ACE
4. Inconsistent and complex data	AtScale works on the clean gold layer the data team already maintains. The semantic model maps every spelling to one value and describes what each column holds.	Semantic models
5. Nondeterminism	ACE compiles a given logical query to identical physical SQL on each run, and semantic memory reuses certified queries.	Determinism , ACE , Semantic memory
6. Ungoverned access	AtScale enforces model permissions and row security on every query, logs each query and its SQL, and versions definitions in Git.	Governance

Failure point	AtScale fix	Section
7. No accuracy measurement	Canary runs gold-standard questions, flags each miss for CoModel to fix, and reruns the suite to catch regressions.	Canary

Table 4. Failure points 1 to 7 and the AtScale fix for each

The design reduces the number of decisions left to the model. The model selects from a declared set of metrics and attributes, and every step after that selection is deterministic. Fewer decisions mean fewer chances to get one wrong. That’s also why smaller, cheaper models do well on top of the layer.

The path to 100%

AtScale’s BIRD-Interact analysis shows how far the layer gets on its own, and where it doesn’t get far enough yet.⁴ The questions AtScale still missed map onto failure points 1 to 4 in [Table 4](#). One used a business term the model never defined, another asked for a ranking without saying by what, a third needed a metric that lived only in a document, and the rest came down to joins that counted rows twice or values spelled three ways. The flywheel has a component for each.

How a question is asked matters too. A question asked as precisely as a BI report asks it, naming the metric, the filters and the period, compiles to the same SQL the report runs, so the answer matches the report every time. That’s how a finance team producing quarterly results would use AI. The hard case is everyday conversational language from many people at once, which is loose and varies from person to person, and that’s where the remaining misses come from.

Closing them takes people and tools together. An analyst says what a term like “VPN abuser” means, and a data engineer confirms what one row of a table stands for. The tools keep each answer. Ava asks the question, and CoModel writes the answer into the semantic model. ACE computes it consistently, multi-pass metrics included, because it plans the passes from the semantic graph (see [The semantic graph](#)) and the LLM never has to work them out. Canary keeps each fix as a regression test so the same error can’t come back unnoticed, and [AtScale governance](#) makes sure a person reviews and approves every change before the model deploys to production. Each fix raises accuracy for every later question that touches it, which is how the loop keeps moving toward 100%. AtScale hasn’t yet published a rerun with these fixes applied, so 100% is still the target.

How AtScale lowers AI cost

AI analytics costs money in three places: LLM tokens, warehouse queries and the work that follows a wrong answer. All three grow when the LLM has to work out what each metric means on every request. The semantic model already holds those definitions, so AtScale cuts all three.

LLM tokens. Without a semantic layer, an LLM explores the schema, writes test queries and retries until a query runs, and each attempt adds tokens. In AtScale's preliminary BIRD-Interact runs, the cost per right answer fell for all seven models, by about 21% on average and from 7% to 44% (see [Table 3](#)).⁸

Warehouse queries. An LLM without a semantic layer sends more queries to the warehouse: schema probes, test queries and retries after failures. With AtScale, ACE rejects a request that doesn't fit the model before it reaches the warehouse, serves repeat questions from cache or an aggregate (see [Aggregates and caching](#)), and semantic memory lets the AI agent start from a query that already worked.

Wrong answers. A wrong answer doesn't appear on any bill, but it costs the business the decision made on it, the downstream work when an AI agent acts on it and the time spent finding and fixing it. Canary catches those misses before anyone acts on them.

How AtScale makes it safe to automate with AI agents

Inaccurate AI also limits what a company can automate. A person reading an answer in a chat can usually tell when a number's off. An AI agent that reorders stock, changes a price or files a report acts on the number with no one checking it first, so it needs answers that were checked before it asked.

That's what the whole flywheel is for. AI agents can only call certified metrics, the ones defined in the semantic model and merged to production after review (see [Governance](#)). Ava points each AI agent's request at the right one, and ACE computes it under the AI agent's own permissions. Canary tests every change against the company's gold-standard questions before it ships, CoModel turns each miss into a fix, and semantic memory keeps the queries that passed. A metric that reaches an AI agent has been defined, tested and approved, and a change that would break one is caught first. Every metric that passes is one more number the business doesn't have to check by hand before an AI agent acts on it.

How AtScale protects your AI sovereignty

AI sovereignty means the customer decides where its data lives, which AI it uses and who can see what. A company can keep its data in Snowflake, Databricks or BigQuery, move between models such as Grok, Kimi and Claude Opus as prices and needs change, and run more than one at a time, without rebuilding its metrics or handing its business definitions to any vendor.

AtScale also runs where the data does, in the customer's cloud account or its own data center.

The business definitions live in the semantic model, written in SML and stored in the customer's own Git repository, so no LLM vendor owns them. Any model that can reach the AtScale MCP server can use them, and switching models doesn't mean rebuilding a single metric.

The same goes for security. The company keeps its identity provider, roles and row-level rules, and AtScale applies them to every AI tool (see [Data governance](#)).

Because AtScale does the computing, a company can pick models for price, speed or where they run, and [Canary](#) shows which one's accurate enough for each job.

Conclusion

Accuracy on business data comes mostly from system design. When an LLM has to rebuild joins and metric logic for every question, it fails in predictable ways, and a stronger model only fails a little less often. AtScale moves those decisions out of the model. The semantic model holds each definition, Ava asks when a question's unclear, ACE computes the answer, semantic memory reuses what worked, and Canary measures how often it's right.

AtScale's preliminary BIRD-Interact runs, reported in [AtScale's BIRD-Interact Analysis](#), put numbers on that. Every model gained accuracy with AtScale, and the misses that remain fall under failure points 1 to 4, each with a named fix: a clarifying rule in Ava, a metric declared in the model, a relationship ACE can compile, or a value mapping for inconsistent data.

Each miss turns into a fix, and each right answer becomes a query the next question can reuse. That's the loop to remember: model, measure, compute, repeat.

Notes

1. Deloitte, [State of AI in the Enterprise, 2026 edition](#), survey of 3,235 senior leaders in 24 countries, conducted August to September 2025. ↩
2. McKinsey & Company, [The State of AI in 2025](#), November 2025, survey of 1,993 respondents. Agent figures are self-reported; 23% had scaled an agent somewhere in the company. ↩
3. Nan Huo et al., [“BIRD-INTERACT: Re-imagining Text-to-SQL Evaluation via Lens of Dynamic Interactions,”](#) arXiv:2510.05318, 2025. Benchmark code and data: github.com/bird-bench/BIRD-Interact. Leaderboard figures as of September 16, 2026: bird-interact.github.io. ↩
4. AtScale, [AtScale’s BIRD-Interact Analysis](#), companion white paper, October 2026: how BIRD-Interact works, results for seven models, the answerable-question analysis and the path to 100%. Results are preliminary and not yet verified by the BIRD team. ↩
5. David P. Mariani, Dianne Wood, John Lynch, Chuck Bear, Mark Palmer and Jay Schuren, [BIRD Conversational Analytics Benchmark on an AtScale Semantic Layer](#), AtScale draft white paper v0.1, October 2026, results as of October 8, 2026: run protocol, “Learning from cold start” and Appendix D. Results are preliminary and not yet verified by the BIRD team. ↩
6. Philippe Laban, Hiroaki Hayashi, Yingbo Zhou and Jennifer Neville, [“LLMs Get Lost In Multi-Turn Conversation,”](#) Microsoft Research and Salesforce Research, arXiv:2505.06120, 2025. ↩
7. Anthropic, [“How Anthropic enables self-service data analytics with Claude”](#). Offline eval accuracy as reported by Anthropic’s data team. ↩
8. AtScale, preliminary BIRD-Interact runs, October 3 to 7, 2026, published in the [AtScale BIRD-Interact eBook](#) and presented in [Inside the Semantic LAIR, Episode 1](#), October 2026. The 93% and 46% figures are best runs on the 317 business questions that have a correct answer, after six grading corrections applied to both arms (means of three runs: 90.6% and 45.8%). The 36.1% and 39.0% figures are first-question pass rates on all 410 business questions: Gemini 3.1 Pro without AtScale, and Grok 4.3 with AtScale on its cold first run. Final figures are from AtScale’s results workbook, [BIRD Results \(AtScale, three runs, memory on\)](#), leaderboard tabs dated October 3 to 7 and the October 6 modified-setup tab. Full results are in [AtScale’s BIRD-Interact Analysis](#). Results are preliminary and subject to change. In Table 3, accuracy is the first-question pass rate on the 410 business questions, best of three runs, and cost per right answer is API spend per run at list prices divided by first-question passes across all 600 tasks; Nemotron 3 Ultra’s cost uses runs 2 and 3 only, because its run-1 cost record is incomplete. ↩
9. Andrew Brust, Blue Badge Insights, speaking on [Inside the Semantic LAIR, Episode 1: Inside the BIRD-Interact Benchmark](#), October 2026. ↩
10. Apache Software Foundation, [Apache Ossie](#) and [project updates](#); AtScale, [“Have Meaning, Will Travel: Why Apache Ossie Needs You”](#). ↩

11. Irina Saporina and Mirella Lapata, “AMBROSIA: A Benchmark for Parsing Ambiguous Questions into Database Queries,” NeurIPS 2024 Datasets and Benchmarks Track, summarizing Wang et al. (2023) on a commercial text-to-SQL system. ↩
12. Dave Mariani, *Inside the Semantic LAIR*, Episode 1: Inside the BIRD-Interact Benchmark, October 2026: the insider-trading question run by Kimi K2.5 on the raw tables and through AtScale. The results are preliminary and may change. ↩
13. AtScale, *Nine Industries, Nine AI Agents*, ebook, 2026. One tough metric from each of nine industries, including same-store sales, net interest margin, thirty-day readmission rate and on-time delivery, and how AtScale computes each one for AI agents. ↩
14. AtScale, [Pricing](#), and “[Flexible, Consumption-Based Pricing: AtScale’s Pricing Explained](#)”, atscale.com. ↩
15. AtScale Overview and Benchmark session, September 29, 2026: [recording](#) and [notes and transcript](#). Architecture walkthrough by Jay Schuren; benchmark, demo and semantic memory by Dave Mariani. ↩

Index

A

accuracy, [4](#)
fixes, [41](#)
underspecified prompts, [20](#)
see also [benchmarks](#); [Canary](#), [AtScale](#)

ACE, AtScale (AI Computation Engine)
overview, [25](#)
aggregates and caching, [28](#)
graph planning, [25](#)
in the flywheel, [3](#)
request steps, [26](#)
XMLA endpoint, [37](#)

agent tools, [19](#), [36](#)

agents, [19](#)
access rules, [32](#)
flat-table view, [11](#)
walkthrough, [38](#)

aggregates, [28](#)
as graph slices, [28](#)

AI Accuracy Flywheel, [3](#), [41](#)
how it closes all seven, [7](#)

AI clients, [36](#), [19](#)

AI operations
choosing an LLM, [16](#)

ambiguity, see [disambiguation](#)

Apache Ossie, [13](#)

architecture
components, [35](#)
inside ACE, [25](#)
reference diagram, [53](#)

AtScale

ACE (AI Computation Engine), [25](#)
AI client connectivity, [36](#)
architecture overview, [35](#)
Ava (Ask, Verify, Answer), [18](#)
BI tool connectivity, [37](#)
Canary, [15](#)
CoModel, [13](#)
data platform connectivity, [36](#)
deployment, [35](#)
Design Center, [12](#)
governance, [32](#)
MCP server, [36](#)
pricing and editions, [35](#)
semantic memory, [30](#)
Snowflake XMLA endpoint, [37](#)

audit logs, [33](#)

automation
safe use of AI agents, [43](#)

Ava, AtScale (Ask, Verify, Answer)
overview, [18](#)
agent requests, [19](#)
human path, [19](#)
on a semantic model, [20](#)
semantic layer role, [20](#)

B

balances, [28](#)

benchmarks
gold-standard tests, [15](#)
memory runs, [30](#)
see also [BIRD-Interact](#)

BI tools, [37](#)
MDX and DAX, [11](#)
Semantic Views, [37](#)

BIRD COINs
in the semantic memory example, [30](#)

BIRD-Interact

- CoModel models, [13](#)
- insider-trading question, [20](#)
- introduced, [3](#)
- path to 100%, [42](#)
- raw baseline, [5](#)

business context, [9](#)

C

caching, [28](#), [26](#)

calendars, see [time intelligence](#)

Canary, AtScale

- overview, [15](#)
- comparing LLMs on cost and accuracy, [16](#)
- development loop, [38](#)

catalogs, [9](#)

- limitations, [5](#), [20](#)
- see also [context](#)

ChatGPT

- using with AtScale, [18](#)
- see also [GPT](#)

clarifying questions

- Ava, [18](#)
- Gemini, [17](#)

Claude

- using with AtScale, [36](#)

column names, [4](#), [9](#), [24](#)

common approaches, [5](#)

CoModel, AtScale

- overview, [13](#)
- business context, [9](#)
- development loop, [38](#)

compiled semantic graph, see [semantic graph](#)

complex data, see [data](#)

computation, multi-pass, see [multi-pass computation](#)

conflicting definitions, [9](#)

connectors, see [data platforms](#); [BI tools](#)

context, [9](#), [13](#)

- context window, [30](#)
- limitations, [5](#), [9](#)
- see also [context engines](#); [ontologies](#); [catalogs](#)

context engines

- limitations, [9](#)
- see also [context](#)

cost, [43](#)

- aggregates and caching, [28](#)
- LLM tokens, [43](#)
- warehouse queries, [43](#), [28](#)
- wrong answers, [43](#)

cost per correct answer, [43](#), [17](#)

D

dashboards

- as business context, [10](#)

data

- inconsistent and complex, [4](#), [41](#)
- where it stays, [28](#)
- see also [data platforms](#); [business context](#)

data governance

- runtime permissions and row security, [33](#)

data platforms, [28](#)

- requirements, [35](#)
- XMLA endpoint, [37](#)

Databricks, [36](#)

deployed semantic objects, see [DSOs](#) ([deployed semantic objects](#))

deployment, [35](#)

Design Center, AtScale

- overview, [12](#)

determinism, [29](#)

- fewer model decisions, [41](#)
- in ACE, [26](#)
- nondeterminism, failure point, [4](#)
- query planning, [26](#)
- what it means, [30](#)
- why it matters, [30](#)
- see also [nondeterminism](#)

development workflow, [38](#)

dimensions, [11](#)
 hierarchies, [28](#)
 levels in the graph, [22](#)
disambiguation, [19](#)
 failure point 1, [4](#)
 underspecified requests, [20](#)

distinct counts, [28](#)

documentation
 as business context, [9](#)

double counting, [24](#)

drift
 model and schema, [9](#), [13](#)
 skill files, [5](#)

DSOs (deployed semantic objects), [35](#)

E

editions, see [pricing](#)

Excel, [37](#)

F

failure points of AI, [4](#)
 fixes, [41](#)
 remaining misses, [42](#)

flywheel, see [AI Accuracy Flywheel](#)

frontier models, [5](#), [17](#)

G

Gemini
 clarifying questions, [17](#)
 using with AtScale, [36](#)

Git
 models stored in, [11](#), [44](#)
 pull requests and review, [32](#)

gold answers
 from dashboards and spreadsheets, [10](#)

gold layer, [4](#), [41](#)

gold-standard queries, [15](#), [38](#)

governance, [32](#)
 audit logs, [32](#)
 data governance, [33](#)
 identity providers, [35](#)
 impersonation, [26](#), [32](#)
 model governance, [32](#)
 row security, [26](#), [32](#)

governance, AtScale
 overview, [32](#)
 see also [governance](#)

GPT
 using with AtScale, [36](#)

grain, [24](#)

graph, semantic, see [semantic graph](#)

Grok
 using with AtScale, [36](#)

gross margin example, [22](#), [38](#)

H

hierarchies, see [dimensions](#)

I

identity and SSO, see [governance](#)

J

joins, [24](#)
 raw SQL, [20](#)

K

Kimi
 21-line SQL vs 3, [20](#)
 accuracy and cost vs GPT-5, [17](#)
 using with AtScale, [36](#)

knowledge graphs, [5](#)
 limitations, [5](#)
 see also [ontologies](#); [semantic graph](#)

Kubernetes, see [deployment](#)

L

LLMs

- ChatGPT, [18](#)
- Claude, [18](#)
- flat-table view, [25](#)
- Kimi, [20](#)
- MCP clients, [18](#)

Looker, [9](#), [37](#)

LookML, [9](#)

M

MCP server, AtScale

- overview, [36](#)
- Ava inside the server, [18](#)

MDX and DAX, [35](#)

metrics, [11](#)

- certified, [19](#)
- types, [27](#)

model governance

- how definitions change, [32](#)

model selection

- accuracy and cost, [16](#)

modeling, semantic, [11](#)

- AI-assisted, with CoModel, [13](#)
- in Design Center, [12](#)
- see also [semantic model](#); SML (Semantic Modeling Language)

models, AI, [36](#)

- see also [LLMs](#); [open-weight models](#)

models, choosing, [16](#), [44](#)

models, open-weight, see [open-weight models](#)

models, semantic, see [semantic model](#)

multi-pass computation, [27](#)

- in the graph, [27](#)
- metric types, [27](#)

N

Nemotron

- using with AtScale, [36](#)

net interest margin, [27](#)

nondeterminism

- failure point, [4](#)
- see also [determinism](#)

O

ontologies

- as business context, [9](#)
- limitations, [5](#)
- see also [catalogs](#); [knowledge graphs](#)

open-weight models, [17](#)

Ossie, see [Apache Ossie](#)

P

path to 100%, [42](#)

Power BI, [37](#)

pricing

- AtScale Enterprise, [35](#)
- deployed semantic objects, [35](#)
- editions, [35](#)

private preview

- CoModel and Canary, [3](#)

probabilistic

- LLMs vs deterministic computation, [30](#)
- see also [determinism](#)

prompts, [19](#)

- ambiguous human prompts and agent calls, [4](#)
- underspecified, [20](#)
- walkthrough, [38](#)
- see also [disambiguation](#)

pull requests, [32](#)

R

ratios, [27](#)

regression testing, [15](#), [42](#)

relationships, [11](#), [24](#)

retail example

- Finance model objects, [11](#)
- gross margin by region, [22](#), [38](#)
- three-pass plan, [27](#)
- see also [gross margin example](#)

reuse, see [sharing and reuse](#)

row security, see [governance](#)

runtime, [38](#)

S

schemas

drift detection, [13](#)

why they're hard to read, [9](#)

security and permissions

agents and identity, [33](#)

inherited from the data platform, [33](#)

semantic graph, [22](#)

aggregates, [28](#)

flat-table view, [25](#)

full-size figure, [52](#)

grain, [24](#)

multi-pass plans, [27](#)

planning, [25](#)

vs. ontologies, [5](#)

semantic memory, AtScale

overview, [30](#)

development loop, [38](#)

semantic model, [11](#)

compiled into the semantic graph, [22](#)

flat-table view, [11](#)

Semantic Modeling Language, see [SML \(Semantic Modeling Language\)](#)

semantic views

duplicated definitions, [5](#), [12](#)

Snowflake, [37](#)

semi-additive measures, see [balances](#)

sharing and reuse

define once, reuse everywhere, [12](#)

skills

limitations, [5](#)

SML (Semantic Modeling Language), [11](#)

Apache Ossie, [13](#)

compilation, [22](#)

Snowflake, [36](#)

AtScale XMLA endpoint, [37](#)

Native App, [35](#)

Semantic Views, [37](#)

sovereignty, [44](#)

choosing a model, [16](#)

spreadsheets

as business context, [10](#)

SQL, [26](#)

21 lines vs 3, [20](#)

pushdown, [18](#)

retrieving, [36](#)

T

Tableau, [11](#), [37](#)

text-to-SQL, [5](#)

ambiguity, [20](#)

grain errors, [24](#)

multi-pass errors, [27](#)

time intelligence, [28](#)

U

use cases

choosing a model for each, [17](#)

W

warehouses, see [data platforms](#)

X

XMLA endpoint, [37](#)

BI tools, [37](#)

Appendix A: The semantic graph at full size



Appendix B: Reference architecture

[Figure B1](#) puts the components from Part 1 into one architecture. At design time, CoModel and Canary build and test the semantic model, and model governance reviews each change and promotes it to production. At runtime, Ava resolves each request, ACE computes the answer from the semantic graph inside the customer's data platform, and data governance applies AtScale's permissions and the platform's own roles to every query. Each box links to the section that covers it.

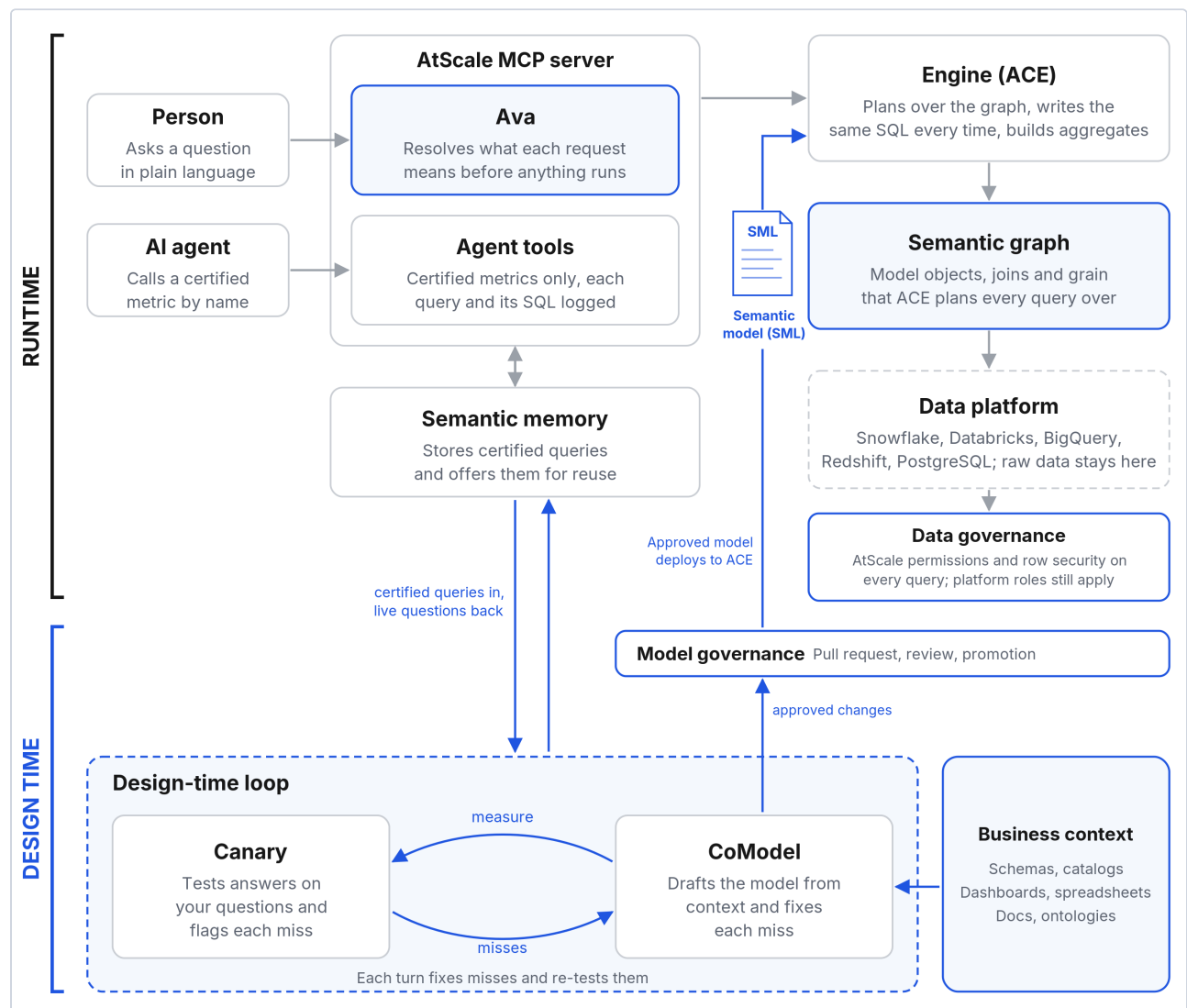


Figure B1. The AtScale for AI architecture.