Strategy

WHITEPAPER

AI Token Cost and Accuracy Benchmark

Evaluating Strategy Mosaic Against Direct Text-to-SQL Execution

Benchmark Evidence on Query Accuracy, Coverage, and AI Token Cost Reduction

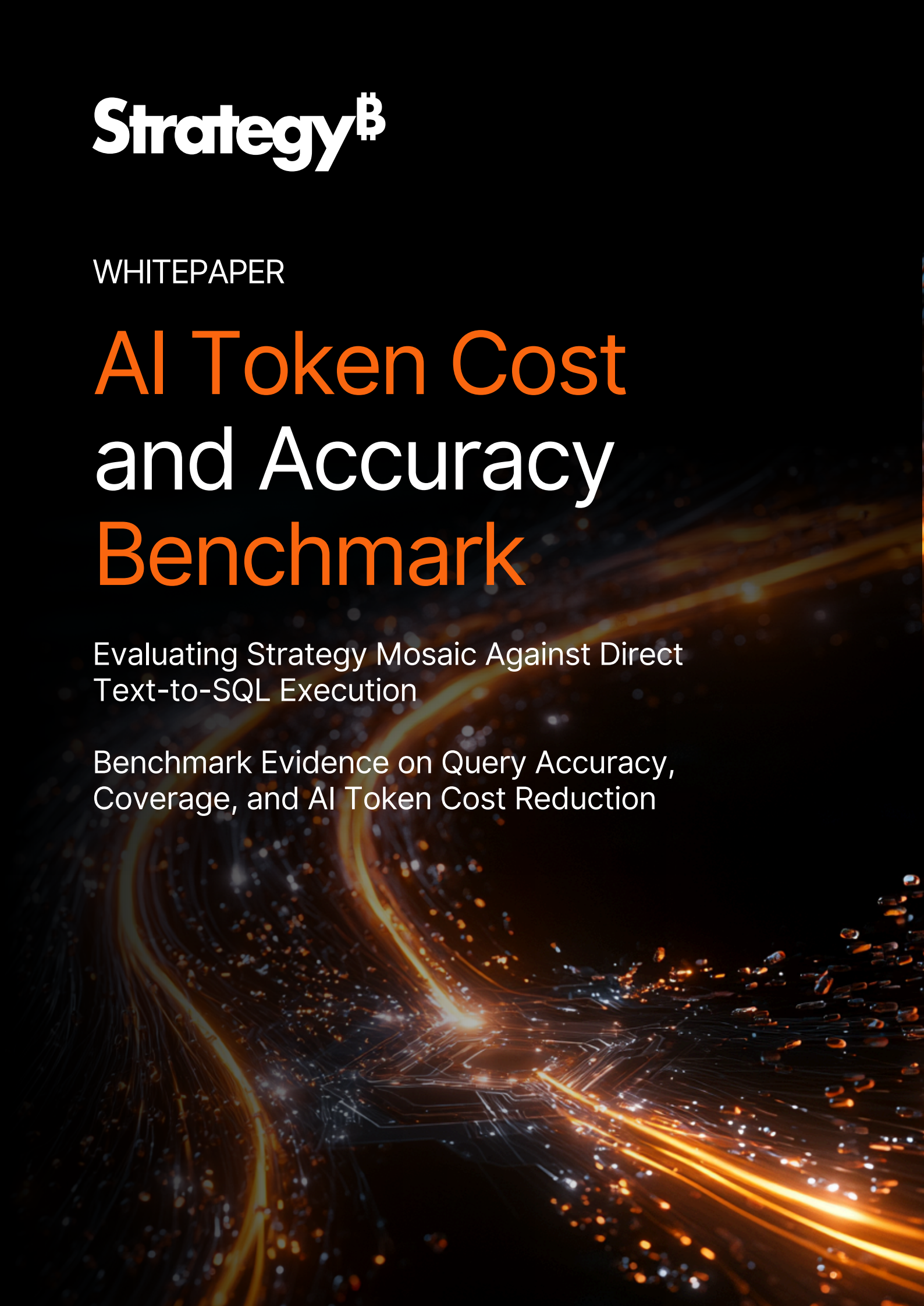


Table of Contents

Executive Summary

As organizations deploy AI agents to query enterprise data, the accuracy and cost of those queries depend on one thing: what context the AI receives. AI doesn't know your business. It doesn't know what your metrics mean, how your data is structured, or which rules apply, so it guesses. The answer looks right but often isn't. Strategy Software's Mosaic is the context layer that fixes this, the semantic layer that gives AI the business context it needs to be reliable, governed, and effective.

Mosaic was benchmarked against direct PostgreSQL querying across 17 insurance claims analytics questions on a real-world insurance data model with 28 tables. The findings are clear on both dimensions:

Accuracy: Mosaic answered all 17 questions correctly. PostgreSQL answered 15. The 2 failures were not random, they were structurally predictable wrong answers on complex multi-table queries, returning financially inflated numbers that looked valid but were not.

Cost: Mosaic significantly reduces AI token consumption and per-query costs. The same semantic model that prevents wrong answers also reduces what organizations pay to run AI analytics at scale.

Query	Direct SQL Returned	Correct Answer	Error
Policy 336: coverage limit	**$40,688,000**	**$5,086,000**	8x inflated
Policy 336: catastrophe payout	**$23,000**	**$13,600**	5x inflated
Policy 337: catastrophe payout	**$9,000**	**$0**	100% misattributed from Policy 336

Each query executed successfully and returned a number. **Only Mosaic returned correct answers in all three cases.**

The sections that follow explain the root causes of both failure modes, the benchmark methodology, and the architectural reasons Mosaic addresses them. For organizations evaluating AI analytics infrastructure, the evidence points to one conclusion: the semantic layer is what makes AI reliable on enterprise data.

1. Token Cost and Wrong Answers in AI-Driven Analytics

The dominant pattern for AI analytics today is text-to-SQL: the AI model receives raw schema information, generates a SQL query, and executes it against the database. Sequeda *et al.* (2023) establish text-to-SQL as the standard deployment baseline, and it is widely used in production environments today. It has two distinct failure modes that become more costly at scale.

1.1 Token Cost

Every text-to-SQL query requires the AI model to first receive the full schema: table names, column names, data types, and foreign key relationships. For a 28-table schema this adds approximately **4,428 tokens of schema context per query**. For production enterprise schemas with 100–500+ tables, this grows to 100,000+ tokens per query.

At current LLM pricing of $3.00 per million input tokens, a ten-person analytics team running 200 queries per day against a 200-table schema incurs over **$15,000/month** in schema context overhead alone, before any SQL query is executed against the data. This cost is paid fresh on every query; schema context is not cached across stateless LLM calls.

Current LLM pricing	Number of analysts	Number of queries
$3.00 per 1 M input tokens	10	**200** queries per day 200-table schema

Schema context overhead:

$15K/ month

1.2 Wrong Answers from Complex Queries

When an AI model generates SQL against a raw schema, it must independently reason about join paths, bridge tables, and aggregation grain. That business logic is entirely absent from the schema definitions. For simple single-table queries this works. For queries spanning multiple fact tables, the model frequently selects the wrong join path and returns numbers that are materially wrong but structurally valid.

> There is no error, no warning, and no way for the user to know the answer is wrong without already knowing the correct answer.

In this benchmark, two multi-fact-table questions returned results from PostgreSQL that were 5–10× inflated or attributed to the wrong policy. Neither failure triggered a SQL error. Both results would have passed routine review by an analyst who did not already know the correct answer.

1.3 Join Complexity and Business Logic

Analytical schemas in real enterprises are layered: star schemas, snowflake schemas, ACORD insurance models, Kimball dimensional designs, all involving multiple fact tables joined through carefully chosen bridge tables. The correct join path is not derivable from column names alone; it is encoded in business rules.

Text-to-SQL generation requires the AI to infer those business rules from table names on every query. The wider the schema, the more plausible join paths exist, and the more likely the AI selects the wrong one.

The semantic layer encodes the correct path once and applies it consistently.

2. How the Semantic Layer Addresses These Failures

A semantic layer sits between the AI and the database. Rather than exposing raw table structures, it presents business-oriented concepts such as “total premium,” “claim payout,” and “loss ratio,” backed by validated join logic. Mosaic's implementation provides three architectural protections against the failure modes described above.

2.1 Fan-Out Protection

Fan-out, in plain terms, is what happens when a join multiplies rows before an aggregation gets to sum them.

A 5-year policy generates 5 rows in policy_coverage_detail. When the AI joins that table to a claims table through a shared key, each claim row matches all 5 coverage rows, and SUM() counts every dollar five times.

The query returns a valid number. The number is wrong by a factor of 5.

Mosaic isolates fact tables at the semantic model level.

When a query requires both coverage limits and claim payments, Mosaic computes each aggregate independently, then joins the pre-aggregated results at the policy grain.

The multiplicative error **never has the opportunity to form.**

2.2 Canonical Bridge Routing

A canonical path is the single authoritative route through a schema to connect two entities.

In the benchmark dataset, claims can be connected to policies via two paths: through insurable_object_identifier (shorter, but ambiguous) or through the claim_coverage bridge table (canonical, which enforces correct policy-year attribution). Without a semantic model, an AI will frequently take the shorter path because it looks correct from the column names alone. That path produces plausible results for simple scenarios and systematically wrong results when multiple policy years or shared insurable objects are involved.

Mosaic encodes the canonical path once in the semantic model. Every subsequent query, regardless of analyst, AI model, or interface, follows the same validated route.

2.3 Compressed Semantic Context

53%

Token spend reduction from compressed schema context

Rather than sending the full raw schema, Mosaic exposes a semantic API (get_semantics) that returns business-friendly field names, pre-computed relationships, and filtered context relevant to the question. This compresses schema context from 4,428 tokens to **2,073 tokens, a 53% reduction**, while simultaneously making that context more precise and less prone to misinterpretation.

The SQL generated against this context is correspondingly simpler. Analysts write GROUP BY queries instead of multi-table JOIN chains. Average SQL token count drops from 174 to 132 per query, a 24% reduction. Both effects compound directly into per-query cost.

2.4 Simple vs. Complex Queries: Why the Gap Widens

For straightforward questions, the performance difference between Mosaic and raw SQL is primarily economic: Mosaic is 37% cheaper. For multi-fact-table questions, the difference is categorical: Mosaic returns the correct answer; PostgreSQL does not. This distinction matters more over time. Enterprise data models are growing in complexity as more data sources, stricter governance requirements, expanding AI initiatives, and distributed analytics all push schemas toward the multi-fact-table patterns where raw SQL fails.

Organizations often pilot AI analytics on simple questions and see comparable results from both approaches. They then deploy at scale, and the complex, high-stakes queries begin to fail.

Query Type	Example	PostgreSQL	Mosaic
Simple: single table	Premium per policy	Correct	**Correct, 37% cheaper**
Moderate: 2-table join with clear FK	Claims by location	Correct	**Correct, 37% cheaper**
Complex: multi-fact-table aggregation	Coverage limit + total claims per policy	Wrong (8-10x fan-out inflation)	**Correct, 37% cheaper**
Complex: bridge-table attribution	Catastrophe payout by policy	Wrong (5x inflation + misattribution)	**Correct, 37% cheaper**

3. Benchmark Methodology

All results in this paper derive from live execution against real data. No metrics are theoretical or estimated; where direct measurement was not possible (due to MCP protocol limitations on server-side timing), data points are clearly labeled with their derivation method.

3.1 Dataset

The dataset follows an ACORD-aligned (U.S. insurance industry standard) insurance data model with 28 tables capturing policies, coverage details, claims, financial amounts, and party relationships. The schema includes the multi-row-per-policy patterns created by multi-year policy terms, exactly the structural condition that triggers fan-out errors in naive SQL.

- Source: Chat-With-Your-Data benchmark (data.world/cwd-benchmark-data)
- Semantic model: Mosaic Library (linked in original benchmark)

3.2 Test Scope

- **17 questions** spanning single-table lookups, multi-table aggregations, ratio calculations, exposure analysis, and cross-domain attribution
- **Live execution** against both PostgreSQL and Mosaic MCP connections, with no synthetic or replayed data
- **SQL generated by Claude Code MCP** for both systems, ensuring equivalent AI generation quality
- **Token counts measured** from actual API responses (schema context) and query character lengths (÷4, standard approximation)
- **PostgreSQL execution times** measured via EXPLAIN ANALYZE: 0.2ms–6.7ms, negligible relative to LLM inference time of ~13–14s for both tools

3.3 Mosaic Configuration

Three one-time configurations were required to encode business intent not inferable from column names:

Configuration	Problem	Business Impact
Party ID unification	person_identifier and organization_identifier map to the same semantic key as agreement_party_role.party_identifier, but names differ and no FK exists	Accurate party-to-policy traversal; without this, party queries misjoin or return empty results
Case + type normalization	party_identifier vs Party_Identifier (case mismatch); bigint vs integer (type mismatch) prevent auto-join	Consistent join routing across party tables without runtime cast errors
Policy number parsing	agreement_name holds 'Policy 31003000336'; policy.policy_number holds '31003000336'. Direct equality join fails due to prefix.	Premium-to-claim analysis at policy level. Computed field strips prefix via SPLIT_PART(agreement_name, ' ', 2).

These configurations are one-time setup. Once encoded, they apply automatically to every subsequent query, regardless of user, interface, or AI model.

4. Accuracy Results: Where Semantics Prevent Failure

Of 17 questions, both systems answered 15 correctly. On the remaining 2, PostgreSQL returned plausible but materially wrong numbers; Mosaic returned correct answers in both cases. These failures are not isolated to this dataset. Any query that joins two fact tables through a shared dimension key, without semantic isolation, will produce the same inflation. They are **predictable structural failures of naive multi-fact-table SQL**, and they would affect every analyst querying this data without a semantic model.

Metric	Mosaic	Without Mosaic	Difference
Correct answers (17 questions)	17 / 17 = 100%	15 / 17 = 88.2%	-11.8%
First-attempt success rate	100%	94.1%	-5.9%
Wrong answer rate	0%	11.8%	+11.8%

4.1 Question 16: Coverage Limits and Total Claims per Policy

Question: "What is the total coverage limit and total claims per policy?"

	Policy 31003000336	Result vs Correct
PostgreSQL	Limit: $40,688,000 Claims: $136,000	8–10× inflated (fan-out)
Mosaic	Limit: $5,086,000 Claims: $13,600	Correct

Root Cause

policy_coverage_detail contains one row per policy per coverage year. A 5-year policy produces 5 PCD rows. When the AI joins policy_amount (fact table) to claim_amount (fact table) through this shared key, each claim row joins to all 5 coverage year rows, multiplying claim amounts by 5 before SUM() aggregates. The query shape (one output row per policy) looks correct. The inflation is invisible without prior knowledge of the correct answer.

Why Mosaic Is Correct

The semantic model isolates fact tables. Coverage and claim metrics are computed via independent canonical paths and combined only as pre-aggregated results at the policy grain. The multiplicative error cannot form.

4.2 Question 17: Total Catastrophe Payout per Policy

Question: "What is the total catastrophe payout per policy?"

	Policy 31003000336	Policy 31003000337
PostgreSQL	$23,000 (5× inflated)	$9,000 (misattributed from Policy 336)
Mosaic	$13,600 (correct)	Not returned (correct — no catastrophe payouts)

Root Causes (Two Compounding Errors)

Fan-out: insurable_object_identifier appears across multiple PCD rows, multiplying claim_amount rows before aggregation.

Bridge misselection: Connecting claims to policies via insurable_object_identifier bypasses the claim_coverage bridge, which contains the authoritative policy-year linkage. Claims belonging to Policy 336 leak into Policy 337 because both policies share an insurable object.

Why Mosaic Is Correct

The semantic model enforces the path: claim → claim_coverage → policy_coverage_detail → policy. This is the only valid attribution path. Both overcounting and cross-policy leakage are structurally prevented.

5. Token Economics

Without business context, AI has to rediscover how your data works on every query. It loads the schema, reasons about relationships, and generates SQL from scratch each time. That burns tokens before any actual analysis begins. Mosaic addresses this directly because business logic is pre-defined in the semantic layer, AI receives only what it needs.

5.1 Mosaic MCP (Semantic Layer via Model Context Protocol)

Mosaic MCP gives the LLM the business context it needs, not the full raw schema. Mosaic provides the governed definitions, relationships, and join logic so the SQL is accurate and precise. The LLM produces the query, but Mosaic defines what that query should look like, abstracting JOIN complexity into simple GROUP BY queries.

Token Component	Mosaic MCP	PostgreSQL	Reduction
Schema context / query	2,073 tokens	4,428 tokens	**53%**
SQL tokens (avg)	132 tokens	174 tokens	**24%**
Total input tokens / query	**~2,405 tokens**	**~4,802 tokens**	**50%**
Cost / query	**$0.01246**	**$0.01966**	**37%**
Total cost: 17 questions	**$0.2119**	**$0.3342**	**$0.12 saved**

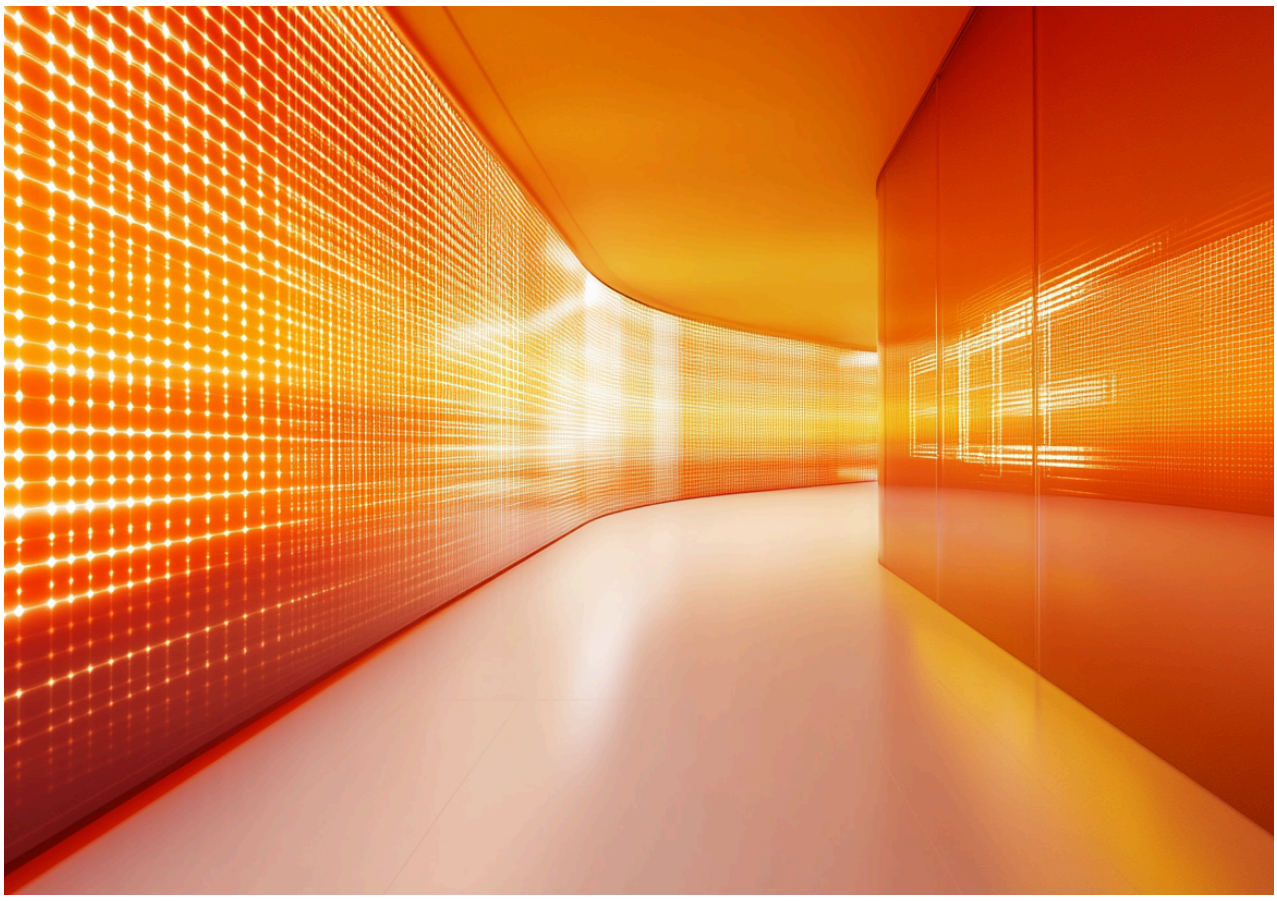
Note: Cost calculated at $3.00/M input tokens + $15.00/M output tokens. Schema context measured from live get_semantics API responses and DDL character counts ÷4.

5.2 Strategy AI Agents

Strategy AI Agents extend Mosaic's semantic layer into pre-built automations. They are purpose-built agents that answer business questions by routing directly through Mosaic's governed business context, with near-zero token overhead. Rather than optimizing what the LLM receives, they remove the LLM from query generation entirely Mosaic handles schema lookup, SQL generation, and execution, and the LLM receives only the natural language question.

Token Component	Strategy AI Agents	Mosaic MCP	PostgreSQL
Schema context	~0 tokens	2,073	4,428
SQL tokens	~0 tokens	132	174
Question text only	~50–100 tokens	200	200
Total / query	**~50–100 tokens**	**~2,405**	**~4,802**
vs PostgreSQL baseline	**~98% reduction**	**50% reduction**	Baseline

Strategy AI Agents deliver approximately 98% token reduction vs direct PostgreSQL and 96% reduction vs Mosaic MCP. AI token consumption is virtually zero; cost is deflected from the LLM provider to the Mosaic software platform. This architecture scales linearly: adding users, questions, or larger schemas does not increase token costs, because no schema context ever touches the LLM.



6. Where Savings Compound in Production

The benchmark numbers above are conservative by design. They were measured against a 28-table demo dataset, without caching, without multi-turn conversation history, and at only 17 questions. Each constraint compresses the measured savings relative to production reality.

Factor	Benchmark Conditions	Production Reality
Schema size	28 tables, ~7K chars	100–500+ tables → 100K+ chars; PostgreSQL overhead grows linearly with table count
Mosaic caching	Disabled (fresh queries)	Enabled → schema tokens drop to $0.00 on repeat queries within session
Query volume	17 questions, one session	Hundreds per analyst per day; 10-person team = thousands of daily queries
Conversation history	Not included	Multi-turn sessions re-send full schema + history each turn (PostgreSQL); Mosaic optimizes context

6.1 Production Estimates

With Mosaic MCP: Conservative 50–70% cost reduction in production, rising toward the upper bound as schema size increases and caching compounds across sessions.

With Strategy AI Agents: 95–98% cost reduction regardless of schema complexity, query volume, or team size. Schema size is irrelevant because no schema context ever reaches the LLM. This is a fundamentally schema-invariant cost architecture.

Production Scale Illustration

An enterprise with 200 tables, 20 analysts running 200 queries/day, at $3.00/M input tokens: PostgreSQL baseline is approximately $15,500/month in schema context overhead. Strategy AI Agents reduce this to under $450/month in question-text tokens. Annual saving: approximately $181,000, before accounting for reduced error correction and re-query costs.

7. Mosaic in the Broader Semantic Layer Landscape

Semantic layers as a category have been independently validated by multiple vendors and academic researchers.

Recent benchmarks from other providers, run against the same ACME Insurance dataset used in this study, confirm that well-implemented semantic layers achieve near-100% accuracy for queries within their modeled scope, dramatically outperforming direct text-to-SQL approaches. This is the right conclusion: the category works.

But not all semantic layer architectures are equivalent. The distinctions that matter in production are coverage, cost, and what happens at the boundaries of the model.

7.1 The Coverage Gap: What Happens When the Semantic Layer Can't Answer

Some semantic layer implementations rely on a metric engine that enforces strict entity traversal rules. When a query requires too many joins, or joins across entity types that haven't been explicitly modeled, those implementations **return an error rather than a wrong answer.** This is far better than a wrong answer. But it is also not an answer.

Published benchmarks on this same dataset show that certain metric-layer architectures require new intermediate models before complex multi-hop queries can be answered at all.

Mosaic's semantic model is designed to encode relationships between existing tables without requiring new intermediate models to be written, tested, and deployed. For organizations with complex, normalized schemas, this distinction has meaningful implications for the time between deployment and full query coverage.

Mosaic works with your schema as it exists

Most semantic layer implementations require new intermediate models before complex queries can be answered.

Mosaic encodes relationships between existing tables through configuration with no restructuring, no new model development, no deployment cycle. In complex, normalized enterprise schemas, that difference is measured in months.

7.2 Token Costs: What Competitive Benchmarks Don't Measure

Published accuracy benchmarks from other semantic layer vendors focus exclusively on correctness. None have addressed token cost reduction at the level Mosaic's testing does, and none have published anything comparable to Strategy AI Agents' external agent architecture.

Dimension	Semantic Layer (industry)	Mosaic MCP	Strategy AI Agents
Accuracy vs. raw SQL	Near-100% for covered queries	100% (17/17)	100% (17/17)
Coverage of complex joins	Varies — may require new modeling	Full coverage via config	Full coverage via config
Token cost vs. raw SQL	Not published	**37% reduction per query**	**~98% reduction — near zero**
Schema-invariant cost model	No	**Partial (caching helps)**	**Yes — schema never reaches LLM**
AI cost scales with user count	Yes	**Yes (minimized)**	**No — question text only**

7.3 Strategy AI Agents: A Different Architectural Category

Strategy AI Agents represent a fundamentally different architectural approach. Rather than optimizing how much schema context is sent to the LLM, they eliminate schema context from the LLM loop entirely. The LLM receives only the natural language question, 50 to 100 tokens, and Mosaic's MCP server handles the rest: schema lookup, query planning, execution, and result formatting.

This architecture has three properties that no other published semantic layer approach has matched simultaneously:

- **Schema-invariant cost:** A 500-table enterprise schema costs exactly the same as a 28-table demo schema in LLM tokens: zero. Accuracy gains from the semantic model are preserved while token costs are eliminated.
- **User-count-invariant cost:** Adding 100 more analysts does not increase AI token consumption. Each analyst's question costs approximately 75 tokens regardless of how many people are asking.
- **Zero wrong answers by construction:** The LLM is never asked to write SQL. It cannot produce a wrong join because it never writes a join. The Mosaic MCP produces deterministic, semantically-validated queries every time.

8. ROI Framework

Use the following to calculate projected savings based on your organization's deployment parameters.

8.1 Mosaic MCP Savings

Input Variable	Value / Formula
Monthly AI database queries (all users)	_____
Avg tokens per query (PostgreSQL baseline)	~4,800 tokens
Mosaic MCP token reduction	**50%**
Monthly token savings	[queries] × 2,400 tokens × [AI provider $/M rate]
Annual LLM cost savings	**[monthly] x 12**
Analyst time saved correcting wrong queries	[# wrong queries/month] × [avg correction time] × [hourly cost]

8.2 Strategy AI Agents Savings (Maximum Cost Deflection)

Input Variable	Value / Formula
Monthly AI database queries (all users)	_____
Strategy AI Agents token reduction	**~98%**
Monthly token savings	[queries] × 4,700 tokens × [AI provider $/M rate]
Annual LLM cost savings	**[monthly] x 12**
Schema-size independence	100-500+ table schemas cost the same as 28-table schemas: zero schema tokens
User-count independence	AI token cost does not scale with number of users

8.3 Choosing the Right Pathway

Profile	Recommended	Why
Existing LLM integrations needing optimization	**Mosaic MCP**	37% immediate savings, drop-in improvement
High query volume (BI/data teams, 100+ queries/day)	**Strategy AI Agents**	Maximum token deflection at scale
Complex enterprise schemas (100+ tables)	**Strategy AI Agents**	Schema complexity doesn't affect token costs
New AI initiatives with budget constraints	**Strategy AI Agents**	Start at near-zero AI infrastructure cost
Multi-tenant SaaS applications	**Strategy AI Agents**	AI cost does not scale with user count

9. Conclusion

Connecting AI agents directly to raw databases is not merely expensive; it is structurally unreliable for the complex multi-table queries that drive enterprise analytics decisions.

Mosaic's semantic layer addresses both dimensions simultaneously. By encoding business logic (canonical join paths, fan-out protections, party-identity unification) once in the semantic model, every subsequent query inherits those guarantees regardless of which AI model, analyst, or interface issues it. The accuracy improvement is not a tuning artifact; it is a structural property of the architecture.

The cost savings are equally structural. Mosaic MCP reduces schema context overhead by 53% and SQL complexity by 24%, delivering 37% cost reduction in the benchmark and 50–70% at production scale with caching. Strategy AI Agents remove the AI from the query execution path entirely, achieving 95–98% cost reduction regardless of schema complexity, an architecture that makes AI query costs effectively schema-invariant and user-count-invariant.

For organizations evaluating AI analytics infrastructure, the question is no longer whether to use a semantic layer. The benchmark data demonstrates its necessity for accurate results on multi-fact-table queries.

Technical Proof Points

100% accuracy	17/17 questions answered correctly across all complexity levels and query types
50% token reduction	Mosaic MCP, measured in live testing against 28-table ACORD-aligned schema
~98% token reduction	Strategy AI Agents: external agent architecture eliminates schema and SQL tokens from LLM context entirely
0% wrong answers	vs 11.8% with direct PostgreSQL; both PostgreSQL failures involved 5–10× financial inflation or policy misattribution
37% cost savings	Per query in benchmark (Mosaic MCP); rises to 50–70% at production scale with caching enabled

Source: Insurance Claims Analytics Benchmark: 17 questions, live execution, measured token counts

Data model: ACORD-aligned insurance schema (data.world/cwd-benchmark-data)

SQL queries generated and executed via Claude Code MCP. Token counts measured from live API responses and query character lengths.

Original benchmark authored by Sushma Ganapuram, Strategy Software.

Strategy Incorporated, 1850 Towers Crescent Plaza, Tysons Corner, VA 22182.

strategysoftware.com

Strategy ₿