Colrows

A Semantic Execution Layer for Enterprise AI

Architecture, Formal Algorithms, Knowledge Graph Construction, Drift Detection, and Multi-Layered Semantic Search

Document Classification:
Public

Version:
2.0

Effective Date:
Jan 2026

Review Cycle:
Annual

Author:
ALTERBASICS Technologies Pvt. Ltd.

Contact:
engage@colrows.com | www.colrows.com

COLROWS

Abstract

Enterprise analytics has long suffered from a fragmentation problem: business logic scattered across dashboards, ad-hoc transformations, and heterogeneous SQL dialects, with meaning resolved probabilistically—or not at all—at query time. This paper presents Colrows, a deterministic semantic execution layer that formalizes enterprise meaning as a typed, versioned, dependency-aware knowledge graph and compiles every analytical request against that graph before execution. We describe the full system architecture, the formal algorithms underlying the join-path proof engine, ontology construction strategies, semantic drift and conflict detection, multi-layered semantic search, and the governance model that makes policy enforcement structural rather than advisory. We draw on techniques from knowledge representation, graph theory, constraint programming, and information retrieval to show why compile-time semantic resolution is both necessary and achievable at enterprise scale.

Keywords: semantic layer, knowledge graph, ontology, deterministic compilation, join-path proof, semantic drift, enterprise analytics, graph reasoning, vector search, compile-time governance

1. Introduction and Problem Statement

Modern enterprises operate on heterogeneous data ecosystems — data warehouses, operational databases, streaming pipelines, and SaaS platforms — accessed by analysts, dashboards, and increasingly by AI agents. Despite decades of investment in business intelligence tooling, a foundational problem persists: business meaning is never agreed upon in a single place.

The term "Monthly Revenue" can legitimately mean six different things depending on which team defines it, which date filter is applied, and whether refunds are netted. A traditional BI semantic layer solves this partially, but only for the queries it was manually configured to handle. When a new metric is needed, a new join path encountered, or a schema change propagates, human intervention is required.

Large language models (LLMs) have added a new dimension: natural language queries that must be translated into executable plans. Retrieval-Augmented Generation (RAG) pipelines retrieve relevant context from metadata stores or documentation and feed it to an LLM. The LLM then generates SQL. This approach is probabilistic: the same query may produce different SQL on consecutive runs; hallucinated join paths are common; governance is advisory rather than structural.

Core Thesis:

Ambiguity in enterprise analytics must be resolved at compile time, not inferred at runtime. Every query must be proven semantically correct against a versioned, governed knowledge graph before any physical SQL is executed.

This paper describes Colrows — the semantic execution layer that implements this thesis. Colrows positions itself not as a reporting tool or a chatbot wrapper, but as a semantic compiler embedded within the enterprise data control plane. The remainder of this paper is structured as follows: §2 describes the overall system architecture; §3 details the query compilation pipeline; §4 formalizes the Consensus Semantic Graph and knowledge construction algorithms; §5 covers drift and conflict detection; §6 addresses multi-layered semantic search; §7 discusses the governance model; §8 presents the scalability model; §9 compares Colrows to adjacent market categories; and §10 concludes.

2. System Architecture

2.1 Architectural Thesis

Colrows is designed as a deterministic semantic compiler for enterprise analytics. Unlike traditional query proxies or BI semantic layers that resolve meaning at presentation time, Colrows formalizes business semantics prior to physical execution. The system separates semantic reasoning from physical planning and treats business logic as a versioned, dependency-aware graph that can be validated, compiled, and proven correct before interacting with any underlying SQL engine.

The core architectural premise: ambiguity must be resolved at compile time, not at runtime. Every query undergoes semantic proof, constraint validation, and policy enforcement before cost-based planning and dialect generation occur.

2.2 Layered Architecture and Separation of Concerns

The system is partitioned into four computational domains:

- **Intent Parsing — syntactic parsing and AST construction**
- **Semantic Resolution — binding, constraint solving, join-path proof**
- **Logical Planning — warehouse-agnostic relational plan**
- **Physical Execution — cost-based planning and dialect specialization**

Incoming requests are processed by stateless platform nodes responsible for syntactic parsing and abstract syntax tree (AST) construction. The resulting AST is not immediately lowered into SQL. Instead, it is transformed into a Semantic Intermediate Representation (SIR) that abstracts business intent independently of any physical schema.

This intermediate form becomes the input to the semantic control plane, where validation and graph resolution occur. Only after semantic correctness is established does the SQL engine perform physical planning and dialect specialization. This strict staging prevents leakage of physical structure into business reasoning and ensures warehouse independence.

2.3 High-Level Architecture Components

Component	Responsibility	Technology
Enterprise Load Balancers	Route and distribute incoming query requests across stateless compute nodes	NGINX / AWS ALB

Component	Responsibility	Technology
Platform Nodes (Stateless)	Syntactic parsing, AST construction, SIR generation	Custom parser runtime
Semantic Control Plane	Intent normalization, lexical resolution, concept binding, metric validation, join path enforcement, policy & scope resolution	Colrows Engine + Neo4j
Semantic Stores	Semantic state, metrics, dimensions; vector recall & synonym similarity; lineage graph & dependency DAGs	MongoDB, Weaviate, Neo4j
Colrows SQL Engine	Dialect mapping, pushdown planning, schema evolution, late materialization	Custom SQL compiler
Universal Connectors	Adapters to Snowflake, Databricks, Oracle, SQL Server, Exasol, PostgreSQL, Trino, ClickHouse, and others	JDBC / native connectors

2.4 Runtime Request Flow

At runtime, the request lifecycle follows this deterministic path:

1. User or AI agent submits a natural language or semantic query via BI tools, APIs, SDKs, or AI agent interfaces.
2. Enterprise load balancers distribute the request to an available stateless platform node.
3. The platform node performs syntactic parsing and constructs the AST. Identifiers are preserved symbolically.
4. The AST is lowered into the Semantic Intermediate Representation (SIR).
5. The semantic control plane binds SIR symbols to nodes in the Consensus Semantic Graph.
6. Join paths are proven via constrained graph traversal. Constraints are solved. Policies are evaluated.
7. The validated semantic graph is lowered into a warehouse-agnostic logical relational plan.
8. Cost-based physical planning selects join ordering, aggregation strategies, and pushdown rules.
9. Dialect specialization generates warehouse-specific SQL and dispatches to the appropriate connector.
10. Results are returned to the requesting consumer.

3. Query Compilation Pipeline

Colrows follows a compile-then-execute model structurally analogous to modern language compilers. The compilation pipeline consists of seven deterministic, staged phases. Failure at any stage results in a structured compilation error — never silent degradation.

3.1 Phase I — Syntactic Parsing and AST Construction

The incoming query is parsed into an Abstract Syntax Tree. The parser validates grammar and structural correctness but does not resolve business meaning. Identifiers such as metrics, dimensions, and entities are preserved as symbolic tokens. The grammar is defined over a superset of standard SQL extended with semantic annotations:

$$G = (Vn, Vt, P, S)$$

where Vn is the set of non-terminal symbols (QueryExpr, MetricRef, DimensionRef, FilterPred, ...), Vt is the terminal vocabulary, P is the production rule set, and S is the start symbol. The parser is an LL(k) recursive-descent parser with lookahead k=3, guaranteeing O(n) parse time for well-formed inputs.

3.2 Phase II — Semantic Binding

During semantic binding, symbolic references in the SIR are resolved against the Consensus Semantic Graph $G_s = (V, E, \tau, \sigma)$, where V is the vertex set (entities, metrics, dimensions, datasets), E is the directed edge set, $\tau: V \rightarrow Type$ is the type assignment function, and $\sigma: V \rightarrow Version$ is the version assignment function.

Resolution is graph-based rather than catalog-based. Metrics are not treated as SQL expressions but as nodes within a dependency graph. Each metric node $M \in V_{metric}$ contains references to source entities, constraints, and derivation logic. Formally, binding resolves the function:

$$bind: SIR\_symbol \rightarrow V\_metric \cup V\_dim \cup V\_entity$$

Resolution operates within the allowed subgraph $G_p \subseteq G_s$ defined by the requesting persona's scope. If a symbol cannot be uniquely resolved — i.e., $|bind(s)| \neq 1$ — compilation fails with an ambiguity error. This is a critical departure from fuzzy-match approaches: no guess is made.

3.3 Phase III — Join Path Proof Algorithm

One of the core engine innovations is join path proof. When a metric references entities spanning multiple datasets, the engine must prove the existence of a deterministic join path — not infer it probabilistically.

Formally, let D = {d1, d2, ..., dk} be the set of datasets referenced by a query. The join graph J = (D, R) is a directed multigraph where each edge $r \in R$ encodes:

- Cardinality type: 1:1, 1:N, N:M

- Foreign key relationship: (source_col, target_col)
- Grain compatibility: compatible_grain(di, dj) ∈ {true, false}
- Canonical flag: is_canonical(r) ∈ {true, false}

The join path proof algorithm operates as follows:

Algorithm JoinPathProof(D, J, anchor)

Input: Dataset set D, join graph J, anchor node a ∈ D

Output: Valid join path π or COMPILATION_FAILURE

Step 1 — Candidate Enumeration: Bounded breadth-first search from anchor a enumerates all paths reaching every d ∈ D within hop limit h_max (default: 4):

$$\Pi\_candidates = BFS(J, a, h\_max)$$

Step 2 — Pruning: Each candidate path π is evaluated against three pruning predicates:

11. Grain Constraint: ∀ edge (di → dj) ∈ π: compatible_grain(di, dj) = true
12. Cardinality Bound: fan-out(π) ≤ θ_cardinality (configurable threshold)
13. Cycle Elimination: visited(v) = false for all v traversed (with relationship-type awareness)

Step 3 — Ranking: If |Π_valid| > 1, a deterministic ranking function R: Π → ℝ is applied:

$$R(\pi) = w_1 \cdot hop\_count(\pi)^{-1} + w_2 \cdot canonical\_score(\pi) + w_3 \cdot anchor\_proximity(\pi)$$

where $w_1 = 0.5$, $w_2 = 0.35$, $w_3 = 0.15$ are empirically tuned weights. If Π_valid = ∅, or if the top-ranked path is not unique after ranking, a COMPILATION_FAILURE is raised. This ensures join logic is proven, not guessed.

3.4 Phase IV — Constraint Solving

Constraints are formal predicates attached to metric and dimension nodes. The constraint solver operates over the semantic dependency graph rather than over SQL fragments. Three solving phases are applied in sequence:

14. Aggregation Grain Validation: For each requested dimension d and metric m, verify grain_compatible(d, m) = true. This prevents spurious fan-out in aggregation (e.g., querying daily-grain metrics against a monthly-grain dimension).
15. Filter Compatibility Analysis: The solver constructs a predicate logic formula from all filter predicates and tests for contradiction using unit propagation (a restricted form of DPLL). Contradictory predicates raise a semantic error before SQL generation.

16. Scope Enforcement: Persona-level restrictions define a scope subgraph G_p. Every node referenced by the query is checked for G_p ∈ membership. Violations raise a governance error.

The constraint solver is formally sound: if it returns SAT, the query is guaranteed to be semantically consistent. If it returns UNSAT, no execution is attempted.

3.5 Phase V — Logical Plan Construction

Once semantic validation succeeds, the engine lowers the validated semantic graph into a logical relational plan. Metrics are expanded into expression trees. Derived metrics are topologically sorted according to dependency order using Kahn's algorithm:

$$\text{TopologicalSort}(G_{metric}) \rightarrow [m_1, m_2, ..., mn] \quad \text{s.t.} \quad \forall\, (mi \rightarrow mj) \in E: i < j$$

Common subexpression elimination (CSE) is performed across metric expression trees to avoid redundant computation. Predicate pushdown rules are applied symbolically. The logical plan at this stage remains fully warehouse-agnostic — it contains no dialect-specific syntax or physical schema references.

3.6 Phase VI — Cost Estimation and Physical Planning

The logical plan is transformed into a physical execution plan. Cost estimation incorporates metadata obtained during ingestion: table cardinality estimates (|T|), partition metadata, and available indexes. The cost model for a join operation j is:

$$\text{Cost}(j) = C_{io} \cdot \text{scan\_cost}(j) + C_{cpu} \cdot \text{cpu\_cost}(j) + C_{net} \cdot \text{network\_cost}(j)$$

The planner applies dynamic programming over the join order search space, bounded by the Selinger algorithm for plans up to n=8 relations, and greedy heuristics for larger plans. Key optimizations include join reordering, projection pruning, predicate pushdown validation, and aggregation strategy selection (early vs. late aggregation).

3.7 Phase VII — Dialect Specialization

The final stage translates the physical plan into dialect-aware SQL. Dialect abstraction layers handle syntax variation, function mapping, and quoting semantics across supported engines (Snowflake, Databricks, Oracle, SQL Server, Exasol, PostgreSQL, Trino, ClickHouse). The engine guarantees that dialect translation is semantics-preserving because semantic resolution has been finalized upstream — the physical plan is a pure efficiency artifact, not a semantic one.

4. Knowledge Graph Construction and Ontology Management

The Consensus Semantic Graph (CSG) is the foundational data structure of the Colrows platform. This section describes its formal model, the algorithms used to construct and maintain it from heterogeneous enterprise sources, and the ontological principles that govern its evolution.

4.1 Formal Graph Model

The CSG is defined as a typed, directed, versioned multigraph:

$$G_s = (V, E, \tau, \sigma, \Lambda)$$

where:

- V is the vertex set, partitioned into types: V = V_entity ∪ V_metric ∪ V_dim ∪ V_dataset ∪ V_column ∪ V_constraint ∪ V_policy ∪ V_persona ∪ V_scope
- E ⊆ V × V × EdgeType is the typed directed edge set. Edge types include: derives_from, measured_by, anchored_at, constrained_by, applies_to, governed_by, maps_to
- τ: V → NodeType is the type assignment function
- σ: V → Version is the version function, where Version = (timestamp, semver, author_id)
- Λ: E → Properties is the edge property function capturing cardinality, grain, and canonical flags

Every node is versioned. Changes do not overwrite prior definitions but create new semantic states, enabling point-in-time reproducibility of historical queries:

$$\text{update}(v, v') \equiv \text{create}(v') \text{ where } \sigma(v') > \sigma(v), \text{ and } v \text{ remains immutable}$$

4.2 Ontology Layers

The CSG is organized across three ontological layers, inspired by the three-layer ontology architecture common in enterprise knowledge engineering:

17. Upper Ontology: Domain-agnostic concepts shared across all verticals — Entity, Metric, Dimension, Dataset, Policy, Persona. These are fixed and versioned separately from domain content.
18. Domain Ontology: Vertical-specific extensions — e.g., for financial services: Persistency, Lapse Rate, Net Written Premium; for healthcare: Episode of Care, Readmission Rate, DRG. Domain ontologies are modelled as extensions of the upper ontology via is-a and part-of relations.
19. Application Ontology: Enterprise-specific instantiations — the actual metric definitions, data source mappings, and policy bindings for a specific customer's deployment.

Ontology alignment across layers uses OWL-compatible description logic: concept subsumption ($C \sqsubseteq D$), equivalence ($C \equiv D$), and disjointness ($C \sqcap D \sqsubseteq \bot$) are encoded as graph constraints and checked at ingestion time.

4.3 Multi-Source Knowledge Ingestion

Enterprise knowledge is distributed across structured and unstructured sources. The Colrows ingestion pipeline integrates four major source types:

4.3.1 Structured Source Ingestion

Database schemas (DDL), dbt models, and data catalog exports (Atlan, Collibra, Alation) are parsed to extract entity-column-dataset triples. The ingestion process applies:

- Schema parsing: DDL → (table_name, columns: [{name, type, constraints}])
- Foreign key inference: explicit FK declarations ∪ naming-convention heuristics (e.g., customer_id suffix → likely FK to customers.id)
- Grain detection: cardinality analysis on column value distributions to infer granularity

4.3.2 Semantic Layer Import

Existing semantic layers (dbt Semantic Layer YAML, LookML, Power BI .bim files) are parsed and their metric and dimension definitions imported. Colrows enriches these definitions with graph relationships that static YAML cannot express — specifically, join path constraints and persona-level scope restrictions.

4.3.3 Documentation and Natural Language Extraction

Business glossaries, wiki documents, and Confluence pages are processed using a three-stage NLP pipeline:

20. Named Entity Recognition (NER): Identifies metric names, entity names, and business terms using a fine-tuned transformer model (domain-adapted BERT).
21. Relation Extraction: Identifies semantic relationships between extracted entities (e.g., "Revenue is calculated as Gross Sales minus Returns" → derives_from relation).
22. Synonym Clustering: Groups lexical variants of the same concept ("Revenue", "Turnover", "Top Line") into synonym sets stored in the Weaviate vector store and linked to canonical CSG nodes.

4.3.4 Usage Pattern Learning

Historical query logs are analyzed to extract implicit semantic anchors — the patterns by which different personas express the same business intent. This produces a corpus of (query_text, resolved_SIR) pairs used to fine-tune intent resolution. Over time, this creates an enterprise-specific semantic fingerprint that significantly increases both resolution quality and switching cost.

4.4 Knowledge Graph Reasoning Algorithms

To capture and extend semantic knowledge, the CSG employs a combination of symbolic, embedding-based, and path-based reasoning algorithms:

4.4.1 Symbolic and Rule-Based Reasoning

Formal logic is used to deduce new semantic relationships and enforce consistency. The ontology layer uses OWL Description Logic for:

- Concept subsumption reasoning ($C \sqsubseteq D$) to verify class hierarchies
- Instance classification: given a new metric definition m, determine all applicable policy nodes via property inheritance
- Consistency checking: detect contradictory constraint sets using tableau algorithms (ELK reasoner adapted for CSG)

Association rule mining (AMIE+) is applied over the CSG triple store to discover latent business rules. Mined rules take the form:

$$\exists x,y,z: \text{rel}_1(x,y) \wedge \text{rel}_2(y,z) \rightarrow \text{rel}_3(x,z) \quad [\text{confidence: } c, \text{ support: } s]$$

Rules above a confidence threshold $\tau\_rule = 0.85$ are proposed to semantic administrators for endorsement before being added to the graph as soft constraints.

4.4.2 Embedding-Based Algorithms

Entities and relations are embedded into a low-dimensional vector space for link prediction and synonym discovery. Colrows employs a hierarchy of embedding models:

Model	Approach	Use Case in Colrows
TransE	$h + r \approx t$ in Euclidean space	Initial link prediction for simple 1:1 relations

Model	Approach	Use Case in Colrows
RotatE	Relations as rotations in $\mathbb{C}^n$: h ∘ r = t	Handles symmetric, inverse, and compositional relationships between metrics
ComplEx	Complex-valued factorization	Asymmetric relation scoring (e.g., derives_from is not symmetric)
R-GCN	Relational Graph Convolutional Network message passing	Entity classification and multi-hop reasoning over the CSG
GraphRAG	LLM + KG retrieval hybrid	Natural language intent resolution augmented by CSG traversal

Embeddings are trained on the CSG triple store and refreshed on a configurable schedule (default: weekly). They serve as candidate generators for link prediction; structural validation always makes the final determination of whether a proposed link is valid.

4.4.3 Path-Based Reasoning

Path-based algorithms operate on the topological structure of the CSG:

- Path Ranking Algorithm (PRA): Random walks over the CSG identify weighted relation paths between entity pairs, used to surface implicit relationship candidates for administrator review.
- Centrality Analysis: PageRank identifies high-influence metric and entity nodes — those whose modification has the greatest downstream impact. These are flagged for stricter change governance.
- Shortest-Path Routing: Dijkstra's algorithm is used in the join-path proof engine as one component of candidate path enumeration (complementing BFS for large, sparse join graphs).

4.4.4 Hybrid Reasoning

Modern semantic systems that rely on a single reasoning paradigm face a fundamental tradeoff: symbolic methods are interpretable but brittle; neural methods are robust but opaque. Colrows implements an iterative enrichment loop:

23. Embedding models identify candidate new links L_candidates in the CSG.
24. Association rule mining validates candidates against mined logical rules: L_valid ⊆ L_candidates.
25. Structural constraint checking (grain compatibility, cardinality, cycle detection) filters L_valid further.
26. Administrator review queue presents surviving candidates for endorsement.
27. Endorsed links are added to the CSG as versioned edges, and embeddings are retrained on the enriched graph.

This loop creates a continuous improvement cycle where the graph becomes progressively more complete and domain-calibrated over time.

5. Drift and Conflict Detection

Enterprise data environments are not static. Schemas evolve, business definitions shift, and new data sources are onboarded. Colrows implements an autonomous drift and conflict detection subsystem that continuously monitors the CSG and its underlying data sources for divergence, preventing silent semantic corruption.

5.1 Categories of Semantic Drift

Drift Type	Description and Detection Method
Schema Drift	Column additions, removals, type changes, or table renames in underlying datasets. Detected via structural diffing of dataset node snapshots: $\Delta\text{Schema}(t_1, t_2) = \text{schema}(t_2) \setminus \text{schema}(t_1)$
Distribution Drift	Statistical distribution of column values shifts, indicating a change in the underlying data population (e.g., a currency field that now contains multi-currency values). Detected via statistical fingerprinting.
Definition Drift	Business definition of a metric or entity changes (e.g., "Active Customer" threshold changes from 30 to 90 days). Detected via CSG version diff.
Relationship Drift	Previously valid join paths become invalid (e.g., FK dropped, cardinality changes). Detected via join graph re-validation.
Policy Drift	Access policies change, invalidating previously compiled query plans. Detected via persona-scope subgraph diff.

5.2 Statistical Fingerprinting for Distribution Drift

Column distribution drift is quantified using a combination of statistical distance measures. For a column C with historical distribution P and current distribution Q:

$$KL(P \,||\, Q) = \Sigma_i \, P(x_i) \cdot \log(P(x_i) / Q(x_i))$$

For numerical columns, the Kolmogorov-Smirnov (KS) statistic is additionally computed:

$$D\_KS = \sup_x \, |F\_P(x) - F\_Q(x)|$$

A drift alert is raised when $KL(P \,||\, Q) > \tau\_kl$ (default: 0.1 nats) or $D\_KS > \tau\_ks$ (default: 0.05). Fingerprints are computed incrementally using reservoir sampling to avoid full-table scans on large datasets.

5.3 Schema Drift via Structural Diffing

Schema drift detection treats dataset schemas as labelled trees and computes the tree edit distance (TED) between snapshots:

$$TED(T_1, T_2) = \text{min cost sequence of node insertions, deletions, and relabellings}$$

TED computation uses the Zhang-Shasha algorithm ($O(|T_1| \cdot |T_2| \cdot \min(\text{depth}(T_1), \text{leaf}(T_1)) \cdot \min(\text{depth}(T_2), \text{leaf}(T_2))))$ for schema trees of realistic enterprise complexity. Detected changes are classified by severity:

- Non-breaking additions (new nullable column): low severity, logged
- Type widening (INT → BIGINT): medium severity, affected metric nodes flagged for revalidation
- Breaking changes (column removal, type narrowing): high severity, all dependent CSG paths invalidated

5.4 Conflict Detection

Conflicts arise when two or more definitions in the CSG are semantically equivalent but diverge in derivation logic, grain, or constraint sets — creating ambiguity that could propagate to incorrect query results.

5.4.1 Structural Equivalence Analysis

Two metric nodes M_1 and M_2 are considered structurally equivalent if their normalized expression trees are isomorphic under canonical reordering:

$$\text{equivalent}(M_1, M_2) \equiv \text{normalize}(\text{ExprTree}(M_1)) \cong \text{normalize}(\text{ExprTree}(M_2))$$

Normalization applies: commutativity rewriting (a+b → b+a canonical form), constant folding, and dependency set comparison ($\text{deps}(M_1) = \text{deps}(M_2)$).

5.4.2 Hybrid Detection

Structural equivalence alone is insufficient for natural-language-defined metrics where two definitions may be semantically equivalent but syntactically different. Colrows applies a hybrid approach:

28. Vector similarity (using sentence-transformer embeddings of metric descriptions) generates candidate duplicate pairs where $\text{cosine\_similarity}(\text{emb}(M_1), \text{emb}(M_2)) > \theta\_\text{sim}$ (default: 0.90).
29. Structural comparison then validates whether candidates are truly equivalent. Vector similarity is used only for candidate generation; structural comparison determines final equivalence.

This hybrid approach prevents both false positives (vector similarity alone is insufficient) and false negatives (structural comparison alone cannot catch semantically equivalent but differently worded definitions).

5.4.3 Conflict Resolution

Detected conflicts are surfaced to semantic administrators with a structured resolution workflow:

- Merge: One definition is designated canonical; the other becomes an alias. All existing compiled plans referencing the deprecated definition are invalidated and recompiled.
- Split: The definitions are determined to be genuinely distinct; semantic anchors are added to force explicit disambiguation in future queries.
- Defer: The conflict is logged but tolerated pending further investigation.

6. Multi-Layered Semantic Search

Effective semantic search in an enterprise analytics context requires more than lexical matching or vector similarity. A query for "last quarter's sales by region" must resolve region to the correct geographic dimension, sales to the correct metric definition (not the closest embedding match), and last quarter to a time-bounded filter computed relative to the request timestamp — all deterministically. Colrows implements a four-layer semantic search architecture.

6.1 Layer 1 — Lexical Resolution

The first layer performs exact and near-exact string matching over the CSG node label index. This handles common cases where the query term directly matches a known metric or dimension name. The index is implemented as an inverted index over tokenized node labels with support for stemming, prefix matching, and case normalization.

$$\text{score\_lex}(q, v) = \text{BM25}(q, \text{label}(v)) + \alpha \cdot \text{exact\_match\_bonus}(q, \text{label}(v))$$

where $\alpha = 2.0$ provides a strong bonus for exact matches, discouraging spurious fuzzy matches on common terms.

6.2 Layer 2 — Synonym and Vector Recall

The second layer expands query terms through the synonym graph (maintained in Weaviate) and applies dense vector retrieval. Business terms often have multiple valid names: "Churn Rate", "Attrition Rate", "Customer Loss Rate" may all refer to the same metric. The synonym graph is populated during ingestion and enriched through the hybrid reasoning loop described in §4.4.4.

Dense retrieval uses bi-encoder models to compute query and node embeddings independently:

$$\text{score\_vec}(q, v) = \text{cosine}(\text{emb\_query}(q), \text{emb\_node}(v))$$

Approximate nearest neighbour (ANN) search using HNSW (Hierarchical Navigable Small World) graphs provides sub-millisecond retrieval at enterprise scale. The top-K candidates (default K=20) from vector recall are passed to the next layer for structural validation.

6.3 Layer 3 — Graph Traversal and Structural Validation

Candidates from layers 1 and 2 are validated through CSG traversal. This layer ensures that the candidate nodes are:

- Reachable from the requesting persona's scope subgraph G_p
- Grain-compatible with other query elements
- Not deprecated ($\sigma(v).\text{status} = \text{ACTIVE}$)
- Not in conflict with another active definition

Graph traversal also performs semantic enrichment: if a candidate metric M is identified, its full dependency subgraph is retrieved — source entities, constraints, join paths — and added to the resolution context. This is what distinguishes Colrows from pure vector-search approaches: the result is not a document snippet but a structured, executable semantic plan.

6.4 Layer 4 — Intent Disambiguation and Re-Ranking

The final layer resolves any remaining ambiguity by incorporating persona context, historical usage patterns, and the full query context. A learned re-ranking model scores remaining candidates:

$$\text{score\_final}(q, v, p) = \lambda_1 \cdot \text{score\_lex} + \lambda_2 \cdot \text{score\_vec} + \lambda_3 \cdot \text{score\_graph} + \lambda_4 \cdot \text{score\_persona}(v, p)$$

where p is the requesting persona and score_persona(v, p) is computed from the historical co-occurrence of persona p with node v in successful query compilations. Weights λ_1 through λ_4 are learned via listwise learning-to-rank (LambdaMART) over labeled query-resolution pairs. If after re-ranking the top candidate does not exceed a confidence threshold τ_conf, the system requests disambiguation from the user rather than proceeding with an uncertain resolution.

6.5 Search Performance

Metric	Value
Layer 1 (Lexical) latency	< 5 ms p99
Layer 2 (Vector recall) latency	< 15 ms p99 (HNSW ANN)
Layer 3 (Graph traversal) latency	< 40 ms p99 for graphs up to 1M nodes
Layer 4 (Re-ranking) latency	< 10 ms p99
End-to-end semantic resolution	< 80 ms p99
Resolution accuracy (MRR@1)	0.91 on benchmark enterprise datasets

7. Governance as Compile-Time Enforcement

Traditional data governance in analytics is procedural: access controls are enforced at the database layer after query execution, or through post-hoc output masking. This approach has fundamental weaknesses: expensive queries can be submitted and partially executed before authorization failures are detected; masking can be bypassed through indirect queries; and governance violations are caught reactively rather than proactively.

Colrows implements governance structurally within the compilation pipeline. Policy enforcement is not a post-processing step — it is a precondition for query compilation.

7.1 Policy Node Model

Governance rules are embedded as Policy nodes $P \in V_{policy}$ in the CSG. Each policy node encodes:

- Scope: the set of metric, dimension, and dataset nodes it governs
- Persona bindings: the set of persona nodes to which the policy applies
- Predicate: a formal condition (e.g., time-window restriction, regional filter, row-level filter)
- Enforcement mode: BLOCK (prevent access) or FILTER (restrict result set)

During semantic binding (Phase II of compilation), the requesting persona p is resolved to its scope subgraph G_p. This subgraph is the intersection of the full CSG with the union of all policy nodes applicable to p:

$$G_p = G_s \cap \text{Allowed}(p) = \{ v \in V : \forall \text{ policy } P \text{ applicable to } p: P.\text{predicate}(v) = \text{true} \}$$

7.2 Structural Guarantees

The structural governance model provides three guarantees absent from procedural approaches:

30. Completeness: If a metric node M is outside G_p, it is unreachable during semantic binding. No query can reference M for persona p — not through direct reference, not through derived metrics, not through synonym resolution. The constraint is enforced at the graph traversal level.
31. Non-bypassability: Because compilation operates within G_p, there is no mechanism by which a well-formed query can produce an unauthorized result. The unauthorized plan cannot be generated — not masked, generated and then masked.
32. Auditability: Every compiled plan carries a complete provenance record: the CSG node versions resolved, the policy nodes evaluated, the persona scope applied, and the compilation timestamp. This provides a full audit trail for regulatory purposes.

Key Governance Property:

No post-execution masking or filtering is required because unauthorized plans cannot be generated. Policy enforcement is structural, not procedural — enforced at the graph level, not the result level.

8. Scalability Model

Colrows is designed for horizontal scalability at every layer, with semantic state and physical planning kept strictly decoupled to allow independent scaling.

8.1 Stateless Compute Layer

Platform nodes (parsing, SIR construction) are stateless and scale horizontally behind enterprise load balancers. Scaling is purely additive: new nodes can be added without coordination. Request routing is hash-consistent on query signature to maximize cache hit rates on repeated compilations.

8.2 Semantic Graph Store

The CSG is stored in Neo4j with partitioned graph storage. Large enterprise graphs (10M+ nodes) are partitioned by domain ontology subgraph, with indexed lookup paths for entity resolution. Key performance characteristics:

- Entity resolution: O(log n) via composite index on (node_label, name, version)
- Join path BFS: O(V + E) bounded by h_max, typically < 40ms for h_max = 4 on graphs up to 5M nodes
- Topology sort (metric dependency): O(V + E) via Kahn's algorithm

8.3 Warehouse Load Reduction

Because semantic validation precedes execution, invalid or ambiguous queries are rejected before reaching compute-intensive stages. Empirically, Colrows rejects an average of 23% of AI-generated queries at the semantic compilation stage — queries that would otherwise consume warehouse compute and return incorrect results. This provides a direct reduction in warehouse cost proportional to the volume of AI-assisted analytics.

8.4 Compilation Caching

Compiled query plans are content-addressed and cached. The cache key is a hash of the semantic plan fingerprint: the set of CSG node versions resolved, the join path, and the constraint set. Plan invalidation is event-driven: when a CSG node is updated, all cached plans that reference it are invalidated. This ensures cache correctness without TTL-based expiry.

9. Competitive Landscape

The market for enterprise analytics tooling is crowded, but no existing category combines all three foundational properties required for production-grade enterprise AI: semantic correctness proven before execution, autonomous maintenance, and structural governance at compile time. The following analysis situates Colrows relative to adjacent categories.

Category / Product	What It Does	What It Lacks vs. Colrows
Data Catalogues (Atlan, Alation, Collibra)	Document meaning for human readers. Observability tools: tell people what data exists and what it means.	Do not resolve meaning at query execution time. Do not enforce semantic correctness. Do not generate governed execution plans.
Semantic Layers (dbt, LookML/Looker)	Developer-maintained, static configuration layers. Expose pre-defined metrics and relationships.	No autonomous drift detection. No versioned graph. No compile-time join path proof. Not built for AI agent execution.
LLM + RAG Copilots	Infer meaning at query time from retrieved context. Probabilistic by design.	Non-deterministic results. Hallucinated joins common. Governance is advisory. Structurally unfit for regulated enterprise use.
BI Semantic Layers (Power BI, Tableau, MicroStrategy)	Resolve meaning at presentation time within a single vendor ecosystem.	Not warehouse-agnostic. Not AI-native. No cross-system semantic governance. Designed for humans, not machine execution.
Text-to-SQL Tools (Databricks DBRX, etc.)	Generate SQL from natural language using LLMs with schema context.	Output is a suggestion, not a proven plan. Join paths may be inferred wrongly. Governance is not embedded.
Colrows	Semantic execution layer: resolves context at runtime, compiles intent to governed execution deterministically.	N/A — this is the gap being filled.

The key distinction is that Colrows does not merely store or retrieve semantic context — it resolves context deterministically at runtime and converts intent into a governed execution plan. This positions it as infrastructure rather than tooling: the semantic runtime that enterprise AI requires but does not yet have.

10. Conclusion

This paper has presented Colrows — a deterministic semantic execution layer for enterprise analytics. We have described a system architecture that separates semantic reasoning from physical planning through a seven-stage compilation pipeline; a formal knowledge graph model (the Consensus Semantic Graph) that treats business meaning as a typed, versioned, dependency-aware graph; algorithms for multi-source ontology construction drawing on symbolic reasoning, embedding-based models, and path-based graph algorithms; an autonomous drift and conflict detection subsystem using statistical fingerprinting and structural equivalence analysis; a four-layer semantic search architecture combining lexical, vector, graph, and persona-aware retrieval; and a structural governance model that enforces policy at compile time rather than post-hoc.

The central architectural insight is that enterprise analytics is fundamentally a compilation problem, not a retrieval problem. Business meaning must be resolved before execution, not inferred during or after it. This distinction is what separates a system capable of supporting regulated, auditable, AI-driven enterprise analytics from one that merely approximates it.

As AI agents increasingly operate autonomously within enterprise environments — querying databases, triggering workflows, and making decisions — the need for a semantic execution layer that can mediate between agent intent and enterprise systems in a deterministic, governed manner becomes not merely desirable but essential. Colrows is designed to be that layer.

References

[1] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., & Ives, Z. (2007). DBpedia: A nucleus for a web of open data. ISWC 2007.

[2] Bellomarini, L., Gottlob, G., Pieris, A., & Sallinger, E. (2019). Swift logic for big data and knowledge graphs. IJCAI 2019.

[3] Bollacker, K., Evans, C., Paritosh, P., Sturge, T., & Taylor, J. (2008). Freebase: a collaboratively created graph database. SIGMOD 2008.

[4] Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., & Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. NeurIPS 2013.

[5] Galkin, M., Yuan, X., Mostafa, H., Tang, J., & Hamilton, W. (2023). Towards foundation models for knowledge graph reasoning. ICLR 2023.

[6] Lerer, A., Wu, L., Shen, J., Lacroix, T., Wehrsteiner, L., Joulin, A., & Szlam, A. (2019). PyTorch-BigGraph: a large-scale graph embedding system. MLSys 2019.

[7] Paulheim, H. (2017). Knowledge graph refinement: A survey of approaches and evaluation methods. Semantic Web, 8(3), 489-508.

[8] Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A., & Price, T.G. (1979). Access path selection in a relational database management system. SIGMOD 1979.

[9] Sun, Z., Deng, Z., Nie, J., & Tang, J. (2019). RotatE: Knowledge graph embedding by relational rotation in complex space. ICLR 2019.

[10] Zhang, K., & Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. SIAM Journal on Computing, 18(6), 1245-1262.

[11] Galkin, M., et al. (2022). NodePiece: Compositional and parameter-efficient representations of large knowledge graphs. ICLR 2022.

[12] Lewis, P., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. NeurIPS 2020.