# Semantic Caching for OLAP via LLM-Based Query Canonicalization (Extended Version)

Laurent Bindschaedler[1],*

[1]Max Planck Institute for Software Systems, Saarbrücken, Germany

**Abstract**

Analytical workloads exhibit substantial semantic repetition, yet most production caches key entries by SQL surface form (text or AST), fragmenting reuse across BI tools, notebooks, and NL interfaces. We introduce a safety-first middleware cache for dashboard-style OLAP over star schemas that canonicalizes both SQL and NL into a unified key space—the *OLAP Intent Signature*—capturing measures, grouping levels, filters, and time windows. Reuse requires exact intent matches under strict schema validation and confidence-gated NL acceptance; two correctness-preserving derivations (roll-up, filter-down) extend coverage without approximate matching. Across TPC-DS, SSB, and NYC TLC (1,395 queries), we achieve 82% hit rate versus 28% (text) and 56% (AST) with zero false hits; derivations double hit rate on hierarchical queries.

**Keywords**

OLAP, Caching, Canonicalization, Signatures, Semantics, Star-schema, Natural language, LLMs

## 1. Introduction

Interactive analytics systems increasingly serve diverse clients: BI dashboards, notebooks, and text-to-SQL interfaces accepting natural language (NL) queries. These use cases amplify latency and compute concerns as LLMs are embedded deeper into analytical pipelines [1]. Query caching is a natural lever because OLAP queries scan large fact tables but return small aggregated results [2], making results inexpensive to materialize and reuse.

Analytical caches typically key entries by SQL surface form: normalized query text or Abstract Syntax Tree (AST)-derived canonical representations [3]. NL clients must first translate to SQL, then apply conventional caching. These approaches fragment reuse across heterogeneous clients (Section 2): the same OLAP question issued from different tools or phrased differently in NL generates distinct cache keys, causing redundant backend execution.

This paper presents a semantic caching middleware designed for a specific category of queries: dashboard-style OLAP aggregations over star or snowflake schemas with a single fact table and dimension joins. We focus on this well-defined OLAP subset (Section 3.1) to make equivalence checkable; queries outside this subset bypass the cache and execute directly on the backend. The main idea is to canonicalize both SQL and NL into a shared, structured cache key, known as an *OLAP Intent Signature*. This signature captures the semantics that affect the numeric result: measures, grouping levels, filters, time windows, and post-aggregation operators. SQL is mapped deterministically from the AST, while NL is translated using a large language model (LLM) constrained to a JSON schema.

The design prioritizes correctness. By default, cache reuse aims for *exact-intent* hits under strict validation. To improve hit rates without relying on approximate matching, we introduce two safe derivations with explicit preconditions: (i) rolling up from finer-grain cached results for additive measures [4], and (ii) filtering down from cached supersets when the cached result includes the necessary filter attributes [5]. Intent signatures increase reuse by consolidating SQL variants and NL paraphrases into a single key; derivations extend coverage across drill patterns (roll-up, filter tightening).

*DOLAP 2026: 28th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data, co-located with EDBT/ICDT 2026, March 24, 2026, Tampere, Finland*

*Corresponding author.

bindsch@mpi-sws.org (L. Bindschaedler)
https://binds.ch (L. Bindschaedler)
0000-0003-0559-631X (L. Bindschaedler)

[0]Short version published at DOLAP 2026. This extended version adds detailed background, derivation proofs, expanded evaluation, and discussion.

We evaluate on three decision-support workloads (TPC-DS, SSB, NYC TLC) totaling 1,395 queries with systematic SQL variants and NL paraphrases. We compare against (1) normalized text-based caching, (2) AST-based SQL canonicalization, and (3) NL-to-SQL+AST pipelines, measuring hit rate, backend savings, lookup overhead, and false-hit rate. To assess NL canonicalization reliability, we additionally evaluate on adversarial NL queries and BIRD [6] human-authored questions.

Across the three primary workloads, our cache achieves 82% hit rate versus 28.2% (text) and 55.6% (AST), reducing backend compute by 85–90%. Safe derivations raise hit rate from 37% to 80% on hierarchical workloads. NL accuracy degrades under ambiguity (44% adversarial, 51% BIRD [6]), but layered safety prevents incorrect reuse; at threshold 0.5, precision reaches 76.9% at 36.5% coverage. Section 5.3 details NL risks.

This paper makes the following contributions:

- **Portable intent-signature layer:** An *OLAP Intent Signature* that canonicalizes SQL and NL into a unified key space, enabling cross-client reuse without a single semantic API or platform-specific model.
- **Safety-first reuse with quantified NL failure modes:** Schema validation, confidence gating, and conservative bypassing. We quantify the NL accuracy gap (44% adversarial, 51% BIRD [6]) and show how safety mechanisms trade coverage for precision.
- **Correctness-preserving derivations:** Roll-up and filter-down from cached results, guarded by explicit preconditions.

## 2. Background and Motivation

Query caching is effective for analytical systems [7, 8] because OLAP queries scan large fact tables but return compact aggregated results, making stored results cheap to reuse. However, caching benefits depend on the cache key: semantically identical requests mapped to different keys yield no reuse.

### 2.1. Sources of Cache Fragmentation

Production caches key by SQL surface form—normalized text or AST-derived representations [3]. Three sources fragment reuse in modern analytics. First, *heterogeneous clients* (BI tools, notebooks, templating systems) generate different SQL for the same question through formatting, alias, and predicate-order variation. AST normalization reduces but does not eliminate this. Second, *NL interfaces* introduce phrasing diversity without a stable syntactic anchor [9, 6]; even NL-to-SQL translation shifts rather than resolves the problem, since different phrasings yield different SQL. Third, *approximate semantic matching* (e.g., embedding-based lookups [10]) cannot ensure correctness: "total sales last quarter" and "average sales last quarter" have high similarity but differ by 2–10× in result. The net effect: *high semantic repetition does not imply high cache reuse.*

We focus on *query-level* caching (complete results), not cell-level OLAP caches [11] that store individual cube cells; our middleware works with any SQL-compatible backend.

### 2.2. Motivating Insight

Dashboard-style OLAP aggregations over star schemas share stable semantic structure [2]: the numerical output is determined by measures, grouping levels, filters, time windows, and post-aggregation operators. These components are well-defined regardless of whether the query arrives as SQL or NL, and are verifiable against a schema. We key the cache by an *OLAP Intent Signature* encoding these components, collapsing many surface forms into a single semantic key.

Since NL-to-intent mapping can be error-prone, we design the cache around a safety-first pipeline: strict schema validation, confidence-gated reuse, and correctness-preserving derivations (roll-up [4], filter-down [5]) rather than approximate matching.

## 3. System Design

This section describes a middleware cache that answers OLAP-style aggregation queries. It maps each request (SQL or NL) to a structured *OLAP Intent Signature*, the cache key. The design is correctness-first: reuse requires exact intent equality with strict validation. Extensions beyond exact matches are limited to correctness-preserving transformations under stated assumptions.

### 3.1. Scope and Assumptions

We focus on a high-value class of dashboard queries, specifically aggregations over a star or snowflake schema [2] with a single fact table and joins to dimension tables along schema-defined foreign keys. The queries may consist of `WHERE`, `GROUP BY`, `HAVING`, and optionally `ORDER BY` and `LIMIT`. We deliberately exclude features that complicate semantic equivalence, such as window functions, set operations, correlated subqueries, recursive CTEs, and lateral joins.

**Scope Coverage.** In TPC-DS, 14 of 99 query templates (14%) qualify; the rest use window functions, CTEs, or set operations outside our scope. SSB (100%) and NYC TLC (100%) are fully covered, reflecting their dashboard-oriented design. Queries outside scope execute directly on the backend with no cache interaction, ensuring the system never returns incorrect results for unsupported query patterns.

To make equivalence checkable, we assume that join semantics are dictated by the schema: given the fact table and referenced dimension attributes, there is a unique join path. Requests containing ambiguous join paths (e.g., role-playing dimensions, many-to-many joins, self-joins, or multiple valid paths) are rejected and bypass caching.

**Terminology.** We adopt standard OLAP terminology [12]: a *dimension* is a conceptual grouping (e.g., Time, Geography), while a *level* is a specific granularity within a dimension hierarchy (e.g., Year > Quarter > Month). *Roll-up* aggregates from finer to coarser levels; *drill-down* moves from coarser to finer levels. *Slicing* fixes a single dimension value; *dicing* selects ranges across multiple dimensions.

### 3.2. Architecture

Figure 1 illustrates the system architecture. The middleware operates as an intermediary between clients, such as BI tools, notebooks, and NL interfaces, and the backend OLAP engine.

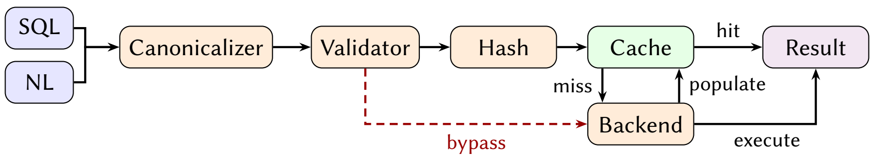

**Figure 1:** Architecture: queries are canonicalized, validated, and hashed as cache keys. Hits return cached results; misses execute on backend. Validation failures bypass the cache (dashed path). Derivation checks (roll-up, filter-down) occur within cache lookup (not shown).

For each request, the following steps are taken: (1) canonicalize the request into an intent signature, (2) validate the signature against the schema and safety rules, (3) look up the signature hash in the cache, (4) on a miss, execute on the backend and store the result under the signature. Because canonicalization and validation are distinct steps, reuse decisions are auditable: the signature is an explicit contract that determines whether a cached result may be returned.

### 3.3. OLAP Intent Signature

We define a canonical signature as a JSON object containing all semantics that can affect the numerical output of the query. The components of the signature include:

- **Measures**: aggregation function and base expression (for example `SUM(f_sales)`), including `DISTINCT` flags.
- **Grouping levels**: list of hierarchy levels in `GROUP BY`, sorted alphabetically to a canonical order (for example `geo.region`, `time.month`). SQL GROUP BY order is semantically irrelevant; we canonicalize to ensure equivalent queries produce identical signatures.

- **Filters**: a set of normalized predicates over non-temporal dimensions and facts, including canonical literal formats.
- **Time window**: explicit start and end boundaries on the time dimension, normalized to a canonical representation. We separate time windows from general filters for two reasons: (i) temporal predicates require special canonicalization (resolving “last quarter” to concrete dates, handling timezone normalization), and (ii) time boundaries are critical for cache invalidation—when new data arrives, entries with open-ended time windows (“last 30 days”) must be refreshed while closed windows (“Q1 2024”) remain valid.
- **Post-aggregation operators**: `HAVING`, `ORDER BY`, `LIMIT`.
- **Metric identity (optional)**: a metric identifier from a governed layer, if available.
- **Scope (optional)**: for multi-tenant deployments, a tenant or user identifier ensures cache isolation across security boundaries.

The optional fields are relevant for governed or multi-tenant deployments; our evaluation uses ungoverned single-tenant schemas.

We serialize the signature into a canonical JSON string (with sorted keys and normalized lists) and compute a hash (SHA-256) to obtain a fixed-length cache key. This ensures different surface forms map to the same signature (Figure 2).

**Query Model and Join Handling.** Join paths are implicit—determined by the schema’s foreign keys—and not included in the signature. The canonicalizer verifies that all referenced columns exist and join paths are unambiguous. Queries with self-joins, role-playing dimensions, or many-to-many relationships bypass the cache, ensuring that matching signatures produce identical results.

**NL:** “Show total revenue by region for electronics in Q1 2024”

**SQL:**

```
SELECT r.region_name,
     SUM(s.amount) AS revenue
FROM sales s
  JOIN regions r ON s.region_id = r.id
  JOIN products p ON s.product_id = p.id
WHERE p.category = 'electronics'
  AND s.sale_date >= '2024-01-01'
  AND s.sale_date < '2024-04-01'
GROUP BY r.region_name
```

**OLAP Intent Signature:**

```
{
  "measures": [{"agg": "SUM",
    "expr": "sales.amount"}],
  "levels": ["regions.region_name"],
  "filters": [{"col": "products.category",
    "op": "=", "val": "electronics"}],
  "time_window": {
    "start": "2024-01-01",
    "end": "2024-04-01"}
}
```

**Figure 2:** Example: an NL question and equivalent SQL query produce identical OLAP Intent Signatures.

## 3.4. Canonicalization

SQL requests are parsed into an AST and normalized deterministically (identifier resolution, predicate ordering, literal canonicalization). NL requests are mapped to signatures using an LLM constrained to produce strict JSON; the LLM also returns a confidence score (0–1) used for safety gating. Low-confidence signatures bypass caching to avoid incorrect reuse.

**SQL → Signature.** The AST is normalized deterministically (identifier resolution, commutative predicate ordering, literal canonicalization) and intent components extracted. Identical signatures imply identical semantics under the schema conditions in Section 3.1.

**NL → Signature.** The LLM prompt includes the schema and a controlled vocabulary of measures and dimensions. The returned confidence score (0–1) is an uncalibrated heuristic used for coarse gating (Section 3.7); more principled alternatives (calibration, log-probabilities, self-consistency) are deferred to future work. Even simple gating substantially improves precision (Table 3).

### 3.5. Validation and Cache Lookup

Before reuse, we validate that all referenced measures/dimensions exist, time windows resolve to concrete boundaries, and join paths are unambiguous. Validation failures bypass the cache and execute on the backend, prioritizing misses over incorrect reuse.

Before reusing any elements, we perform the following validations: (1) all referenced measures/dimensions exist and pass type checks, (2) the specified time window resolves to concrete boundaries, (3) the implied join path is unique within the schema, and (4) unsupported constructs trigger bypass.

If any of these validations fail, the request will still execute on the backend. However, it may only be stored under the *executed* signature if the signature is well-formed and the policy allows it. This approach enforces a conservative default: prioritize misses over incorrect reuse. When the cache reaches capacity, entries are evicted using a Least Recently Used (LRU) policy, which discards the entry that has gone longest without being accessed.

### 3.6. Correctness-Preserving Reuse Beyond Exact Hits

Exact intent matching is the default reuse mode. We also support two safe derivations:

**Roll-Up from Finer-Grained Cache.** If a cached entry has identical measures and filters but finer grouping levels, we can re-aggregate the cached result. Roll-up is permitted only for composable aggregations (SUM, COUNT, MIN, MAX); it is rejected for AVG, COUNT DISTINCT, or ratios.

**Filter-Down from Cached Supersets.** If a cached entry uses a superset filter and contains the attributes needed to apply a tighter filter, we filter the cached result. Derivations are disabled when ORDER BY or LIMIT is present.

**Roll-Up.** Re-aggregation from a finer-grained cached entry is permitted only for composable aggregations [2, 13, 14] (`SUM`, `COUNT`, `MIN`, `MAX`); non-additive measures (`AVG`, `COUNT DISTINCT`, ratios) are rejected. Preconditions: (i) composable aggregation, (ii) summarizable hierarchy (each child maps to exactly one parent at each level), (iii) NULL-preserving semantics.

**Filter-Down.** The cached result must contain the filter attributes needed for the tighter predicate; otherwise the derivation is rejected. Preconditions: (i) filter attributes present in cached columns, (ii) SQL NULL semantics preserved, (iii) no `ORDER BY`/`LIMIT`.

Derivations are disabled under `ORDER BY` or `LIMIT` because re-aggregation or post-filtering can alter top-k membership. Drill-down (finer ← coarser) is unsupported: query-level caching lacks the detail data. Drill-down queries populate the cache but do not reuse coarser entries.

### 3.7. Safety Policy for NL-Driven Reuse

NL canonicalization can be schema-valid yet semantically incorrect (e.g., mapping an ambiguous term to the wrong column). We control NL reuse via layered policies:

- **Schema validation:** required for all reuse to ensure structural correctness.
- **Confidence-gated reuse:** low-confidence signatures bypass caching.
- **Heuristic ambiguity checks:** reject common ambiguity patterns, such as unresolved relative time, underspecified spatial terms, and aggregation-word mismatches.
- **Optional lightweight verification:** a configurable check on NL-originated hits that can catch some classes of mismatch, primarily time-window errors.

These layers are designed to prioritize misses over false hits, ensuring that the reuse behavior aligns with the requirements for analytics correctness.

## 4. Prototype Implementation

We implement the middleware in Python. The *canonicalizer* parses SQL with `sqlglot` and normalizes deterministically (identifier resolution, commutative predicate ordering, literal canonicalization); NL requests are mapped via an LLM endpoint emitting schema-constrained JSON with confidence scores.

The *validator* checks measure/dimension existence, time-window resolution, and join-path uniqueness; failures bypass the cache.

The *cache store* persists results as Parquet files indexed by signature hash, with a SQLite metadata index for derivation candidate lookup (entries matching requested measures with superset dimensions or superset filters). The executor uses DuckDB as backend. NL-string → signature mappings are memoized to avoid repeat LLM calls. Configuration knobs include confidence threshold, heuristic toggles, derivation enable/disable, and snapshot strategy for invalidation.

**Reproducibility.** We will release source code, the LLM prompt template (~800 tokens), workload generation scripts, the JSON Schema specification, and the 63 adversarial queries with annotations.

## 5. Evaluation

We evaluate the intent-signature cache along four research questions: **RQ1** hit rate, cache fragmentation, and NL reuse patterns, **RQ2** correctness (false hits), **RQ3** backend savings and overheads, and **RQ4** coverage gains from correctness-preserving derivations.

### 5.1. Experimental Setup

**Workloads.** We use **TPC-DS** [15] (SF=1, 14 in-scope queries), **SSB** [16] (13 queries), and **NYC TLC** [17] (18 templates): 1,395 queries total (945 SQL, 450 NL). For NL robustness, we use 63 **adversarial NL queries** stressing ambiguity and 150 **BIRD** [6] human-authored OLAP-compatible questions (9.8% of BIRD's dev set, selected per Section 3.1). For derivation evaluation, we construct an **SSB hierarchical workload** drilling through time, geography, and product hierarchies.

**Query Variants and Ground Truth.** Per canonical intent, we generate 21 SQL variants (formatting, alias, predicate-order changes) and 10 manually authored NL paraphrases. All variants are verified to produce identical results, establishing ground truth for hit-rate and false-hit measurement.

**Baselines.** We compare: (1) **TextCache** (normalized SQL text), (2) **ASTCache** (AST-derived canonicalization), (3) **NL-to-SQL+AST** (NL translated to SQL, then ASTCache), (4) **LLMSigCache** (our system). TextCache/ASTCache are SQL-only.

**Platform.** Server: two AMD EPYC 9654 CPUs, 2 TB DDR5 RAM, Debian Linux. Backend: DuckDB. NL uses GPT-4o-mini at temperature 0; Table 5b reports model comparison. Three trials per method; NL memoization cleared between trials.

### 5.2. RQ1: Does Intent-Signature Caching Improve Hit Rates?

Table 1 reports hit rates and reduction factors. LLMSigCache achieves the highest hit rate on all three workloads (82% average), outperforming TextCache (28.2%) and ASTCache (55.6%). Relative to ASTCache, the gain is driven by NL: ASTCache cannot process NL inputs, while LLMSigCache maps NL paraphrases into the same intent key space and enables reuse.

**Table 1:** Cache performance by method. Hit rate (%); Reduction factor (queries per cache key, higher = less fragmentation). TextCache/ASTCache reduction factors are SQL-only (68% of workload).[†]

| | **Hit Rate (%)** | | | | **Reduction (×)** | | |
|---|---|---|---|---|---|---|---|
| **Method** | **NYC TLC** | **SSB** | **TPC-DS** | **Avg** | **NYC TLC** | **SSB** | **TPC-DS** |
| TextCache | 13.8 | 38.2 | 32.7 | 28.2 | 2.1 | 3.7 | 3.2 |
| ASTCache | 63.4 | 54.1 | 49.3 | 55.6 | 32.7 | 31.0 | 24.1 |
| NL-to-SQL+AST | 86.6 | 80.4 | 65.0 | 77.3 | 9.4 | 16.1 | 8.0 |
| LLMSigCache | 96.9 | 81.9 | 67.1 | 82.0 | 19.0 | 16.8 | 9.0 |

[†]ASTCache excludes NL queries (32% of workload), inflating its per-key ratio on the SQL subset.

Within LLMSigCache, NL reuse arises from both *NL-to-NL* hits (paraphrases of the same intent) and *cross-surface* hits (NL matching SQL-populated entries or vice versa). Cross-surface sharing accounts for 3–34% of NL cache hits; most NL benefit comes from unifying paraphrases under intent keys.

All methods produce **zero false hits** on controlled workloads. NL accuracy drops to 44% on adversarial queries and 51% on BIRD [6] human-authored questions, making safety essential: a 0.5 confidence threshold yields 77% precision at 37% coverage. Backend compute drops 85–90%. Safe derivations (roll-up, filter-down) raise hit rate from 37% to 80% on hierarchical workloads.

## 5.3. RQ2: Does Intent-Signature Caching Preserve Correctness?

We distinguish *cache correctness* (a hit returns the same result as backend execution) from *user-intent correctness* (the signature matches what the user meant). Zero false hits ensure cache correctness; NL accuracy determines user-intent correctness.

**False Hits for Main Workloads.** Across all 1,395 controlled queries, all methods exhibit **zero false hits**. Signature specificity (5+ structured fields, SHA-256 hashing) makes collisions rare; SQL-seeded reuse (Section 6.1) restricts NL to read-only cache access for stronger guarantees.

**NL Semantic Accuracy under Ambiguity.** Cache correctness depends on NL canonicalization accuracy when NL requests are allowed to reuse cached results. Therefore, we evaluate canonicalization under two stress settings.

First, we construct 63 adversarial NL queries designed to trigger ambiguity (metric name, time references, dimension disambiguation, aggregation intent, and compositional requests). Table 2 shows that exact semantic accuracy is 44.4% on these adversarial queries, with many errors being *schema-valid but semantically wrong*.

**Table 2:** Semantic accuracy on 63 adversarial NL queries by ambiguity type.

| Ambiguity Type | Example | N | Correct | Wrong | Invalid |
|---|---|---|---|---|---|
| Metric name | “revenue” → `net_amount` vs `gross_amount` | 15 | 11 | 4 | 0 |
| Time reference | “last month” without current date context | 12 | 2 | 10 | 0 |
| Dimension | “area” → `zone` vs `borough` | 12 | 3 | 9 | 0 |
| Aggregation | “average trips” → AVG vs COUNT | 9 | 6 | 3 | 0 |
| Compositional | Multiple measures in single query | 15 | 6 | 4 | 5 |
| **Total** | — | 63 | 28 | 30 | 5 |

Second, we evaluated 150 human-authored OLAP-compatible questions from BIRD [6], using the corresponding schemas. The resulting semantic accuracy is 51.3% in this setting. The main workload paraphrases are controlled rewrites of known-correct intents with unambiguous references; BIRD and adversarial queries introduce realistic ambiguity (synonyms, implicit time references, underspecified dimensions) that is absent from the main evaluation, explaining the accuracy gap.

These results quantify a practical gap: NL canonicalization is the limiting factor under realistic ambiguity. They motivate the safety policy in Section 3.7, which trades coverage for precision via confidence-gated reuse and heuristic ambiguity checks.

Table 3a reports the coverage–precision trade-off: at threshold 0.5, precision rises to 76.9% at 36.5% coverage. Table 3b shows schema-specific heuristics that reject common ambiguity patterns (unresolved relative time, underspecified spatial terms, aggregation-word mismatches). Heuristics reduce wrong signatures from 30 to 9, improving precision from 48% to 69%, but reject 54% of queries. These are deployment-specific templates; operators define analogous rules per schema.

**Table 3:** Safety mechanism effectiveness on adversarial queries (N=63).

**(a) Confidence threshold**

| Threshold | Coverage | Precision |
|---|---|---|
| 0.3 | 63.5% | 62.5% |
| 0.5 | 36.5% | 76.9% |
| 0.7 | 20.6% | 78.3% |
| 0.9 | 12.7% | 100.0% |

**(b) Schema-specific heuristics**

| | Validation only | With heuristics |
|---|---|---|
| Precision | 48.3% | 69.0% |
| Wrong signatures | 30 | 9 |
| Bypass rate | 7.9% | 54.0% |

### 5.4. RQ3: What Are the Backend Savings and Overheads?

**Backend Savings.** LLMSigCache reduces backend compute by 85–90%, comparable to ASTCache on the SQL subset and higher overall because it also caches NL.

**Overhead.** SQL lookup adds 9–16 ms median (Table 4). NL pays ~1.3 s LLM cost on first occurrence; exact repeats are memoized. SQL hits save 150–800 ms backend time at 10–20 ms lookup cost. Derivations operate on in-memory Parquet and remain sub-second.

**Capacity Sensitivity.** Table 4 shows LRU eviction under varying cache sizes. Dashboard-like orderings (Sequential, Zipf) remain effective at 10–25% capacity; Interleaved requires larger caches.

**Table 4:** RQ3 overhead measurements.

**(a) Latency (ms) by scenario**

| Scenario | Med. | P95 |
|---|---|---|
| *SQL queries (all methods)* | | |
| Cache lookup | 9–16 | 12–28 |
| *NL queries (LLMSigCache)* | | |
| First occur. (miss) | 1,317 | 2,266 |
| First occur. (hit) | 1,304 | 2,016 |
| Repeat (memo) | <0.01 | <0.01 |

**(b) Hit rate (%) vs. cache size (NYC TLC)**

| Ordering | 10% | 25% | 50% | 75% | 100% |
|---|---|---|---|---|---|
| Sequential | 96.8 | 96.8 | 96.8 | 96.8 | 96.9 |
| Random | 4.9 | 22.9 | 47.5 | 69.9 | 96.9 |
| Interleaved | 0.0 | 5.1 | 5.1 | 5.1 | 96.9 |
| Zipf | 13.4 | 45.2 | 72.8 | 85.7 | 96.9 |

### 5.5. RQ4: Do Correctness-Preserving Derivations Extend Coverage?

To evaluate derivations, we construct an SSB hierarchical workload where queries drill up and down through time, geography, and product hierarchies. SSB's explicit dimension hierarchies are representative of dashboard drill patterns; TPC-DS and NYC TLC lack systematic hierarchy traversal, so derivations provide limited benefit on those workloads by design. Without derivations, hit rate is 37%; enabling roll-up and filter-down raises it to 80% with zero false hits.

## 6. Discussion and Deployment Considerations

### 6.1. Deployment Knobs

The main deployment risk is NL canonicalization producing *schema-valid but semantically incorrect* signatures. With only 44% accuracy on adversarial queries (Table 2), confidence-gated reuse and ambiguity heuristics are essential. Table 5 summarizes configuration profiles controlling the precision-coverage trade-off. Additional restrictions include *SQL-seeded reuse only* (preventing cache poisoning from NL) and *scoped signatures* with tenant context. In governed deployments (e.g., dbt Metrics, Cube), metric identifiers in the signature can eliminate ambiguity at the source.

**Table 5:** Configuration profiles and model comparison on adversarial queries (N=63).

**(a) Configuration profiles**

| Setting | Conservative | Balanced | Aggressive |
|---|---|---|---|
| Schema validation | Yes | Yes | Yes |
| Confidence threshold | 0.7 | 0.5 | None |
| Schema heuristics | All | Time+spatial | None |
| NL precision | 71.4% | 72.0% | 48.3% |
| NL coverage | 22.2% | 39.7% | 92.1% |
| Wrong cached | 4 | 7 | 30 |

**(b) LLM ablation**

| Model | Correct | Wrong | Invalid | Accuracy |
|---|---|---|---|---|
| GPT-4o-mini | 28 | 30 | 5 | 44.4% |
| Claude-3.5-haiku | 38 | 25 | 0 | 60.3% |

### 6.2. Cache Invalidation

Our evaluation uses static snapshots, but production systems must link entries to data freshness [18]. The prototype records snapshot identifiers per entry; entries are invalidated on schema change or when their time window intersects updated partitions. For append-only fact tables, closed time windows ("Q1 2024") remain valid indefinitely; only open-ended windows ("last 30 days") require refresh. Empirical characterization of cache churn under realistic update patterns—including closed-vs.-open window ratios, update frequency, and dimension change rates—remains future work.

### 6.3. Limitations

**Restricted Query Class.** Queries with window functions, set operations, or CTEs bypass the cache. Derived metrics (ratios, year-over-year) require expression-tree signatures or metric-layer integration.

**NL Canonicalization Quality.** NL accuracy drops on ambiguous inputs (Table 2); safety policies trade coverage for precision.

**Derivation Limits.** Roll-up requires composable aggregations and is disabled for non-additive measures and ordering/top-k queries.

**Synthetic Workload.** Controlled variants may overstate hit rates vs. production traffic. Alternative generators such as CubeLoad [19] and production trace validation would complement controlled benchmarks.

## 7. Related Work

**Semantic Caching and Decision-Support Reuse.** Classic semantic caching stores results using semantic descriptions (e.g., predicate regions) and reuses them when a new query can be answered from cached data [7]. WATCHMAN frames caching as a workload-aware manager for decision-support queries [8]. We follow the same goal but focus on a restricted *intent signature* that serves as a deterministic cache key derivable from both SQL and NL.

**Answering Aggregates Using Views.** Prior work addresses answering aggregation queries from previously computed views [5, 20, 21] and rewriting using dimension hierarchies [4] under summarizability constraints [13, 14]. Our derivations are correctness-preserving rewrites guarded by explicit preconditions. Unlike materialized view selection, which optimizes offline view sets, we perform runtime caching with online key derivation from both SQL and NL.

**SQL and NL Query Caching.** Many systems use normalized SQL text or AST-based canonical forms to key caches [3]. Text-to-SQL research [9, 6, 22, 23] enables NL→SQL→cache-key pipelines. VOOL [24] provides a modular framework for vocalizing OLAP sessions, mapping between NL and OLAP operations; our work shares the goal of bridging NL and structured OLAP semantics but focuses specifically on caching with correctness guarantees rather than session vocalization. We key by OLAP semantic contract directly, extending coverage to both SQL variants and NL paraphrases.

**BI Semantic Layer Caches.** Enterprise BI servers (e.g., Oracle BI Server, Looker, Cube [25], dbt Semantic Layer [26]) implement logical caches with subset and roll-up reuse rules similar to our derivations. However, these caches are keyed by platform-specific logical models; cache reuse requires queries to be routed through the platform's query engine. Our contribution is a *portable middleware* that canonicalizes both heterogeneous SQL and NL into a unified semantic key space for caching, enabling cross-client and cross-modality reuse without requiring adoption of a single platform.

**Approximate Matching and Embedding Caches.** Embedding-based caches use semantic similarity but cannot distinguish "total revenue" from "average revenue" [10]; our zero-tolerance for false hits makes them unsuitable. General query equivalence is hard [27, 28]; our intent signature provides a practical fingerprint for a restricted but common fragment.

## 8. Conclusion and Future Work

We introduced a safety-first semantic caching middleware for dashboard-style OLAP, keyed by an *OLAP Intent Signature* encoding measures, grouping levels, filters, time windows, and post-aggregation operators. Intent-signature caching achieves 82% hit rate across three workloads (vs. 28.2% text, 55.6% AST), yielding 85–90% backend savings; derivations raise hierarchical hit rates from 37% to 80%. NL canonicalization accuracy drops on adversarial inputs, necessitating conservative safety policies. Unlike BI semantic layers, our approach unifies heterogeneous SQL and NL into a single, portable cacheable representation.

**Future Work.** Robust invalidation under data updates is the key gap for production deployment. Promising directions include: stronger schema grounding for NL canonicalization; richer cube-lattice derivations [29]; validation against production query traces; and extending input modalities to MDX (the multidimensional query language for OLAP cubes), which would require an MDX parser but shares the same underlying intent structure.

## Declaration on Generative AI

The LLMs evaluated in this work (GPT-4o-mini, Claude-3.5-haiku) are components of the proposed system and were not used to prepare the manuscript. The author used Claude (Anthropic) and Grammarly for writing style, grammar, spelling, and formatting, and OpenAI Deep Research for citation management. The author reviewed and edited all outputs and takes full responsibility for the content.

## References

[1] B. Mohammadi, L. Bindschaedler, The case for instance-optimized LLMs in OLAP databases, in: Proceedings of DOLAP 2025: 27th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data, co-located with EDBT/ICDT 2025, volume 3931 of *CEUR Workshop Proceedings*, CEUR-WS.org, Barcelona, Spain, 2025, pp. 59–63. URL: https://ceur-ws.org/Vol-3931/short3.pdf.

[2] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, H. Pirahesh, Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals, Data Mining and Knowledge Discovery 1 (1997) 29–53. URL: https://doi.org/10.1023/A:1009726021843. doi:10.1023/A:1009726021843.

[3] ClickHouse, Query cache, ClickHouse Docs, 2026. URL: https://clickhouse.com/docs/operations/query-cache, last modified 2026-01-15 (Git history). Accessed 2026-02-20.

[4] C.-S. Park, M.-H. Kim, Y.-J. Lee, Rewriting OLAP queries using materialized views and dimension hierarchies in data warehouses, in: Proceedings of the 17th International Conference on Data Engineering (ICDE 2001), Heidelberg, Germany, April 2–6, 2001, IEEE Computer Society, 2001, pp. 515–523. URL: https://doi.org/10.1109/ICDE.2001.914865. doi:10.1109/ICDE.2001.914865.

[5] D. Srivastava, S. Dar, H. V. Jagadish, A. Y. Levy, Answering queries with aggregation using views, in: Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB '96), Mumbai (Bombay), India, September 3–6, 1996, Morgan Kaufmann, 1996, pp. 318–329. URL: https://www.vldb.org/conf/1996/P318.PDF.

[6] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo, X. Zhou, C. Ma, G. Li, K. Chang, F. Huang, R. Cheng, Y. Li, Can LLM already serve as A database interface? A BIg bench for large-scale database grounded text-to-SQLs, in: Advances in Neural Information Processing Systems, volume 36, 2023. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/83fc8fab1710363050bbd1d4b8cc0021-Abstract-Datasets_and_Benchmarks.html.

[7] S. Dar, M. J. Franklin, B. Þ. Jónsson, D. Srivastava, M. Tan, Semantic data caching and replacement, in: Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB '96), Mumbai (Bombay), India, September 3–6, 1996, Morgan Kaufmann, 1996, pp. 330–341. URL: https://www.vldb.org/conf/1996/P330.PDF.

[8] P. Scheuermann, J. Shim, R. Vingralek, WATCHMAN: A data warehouse intelligent cache manager, in: Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB '96), Mumbai (Bombay), India, September 3–6, 1996, Morgan Kaufmann, 1996, pp. 51–62. URL: https://www.vldb.org/conf/1996/P051.PDF.

[9] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, D. Radev, Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP 2018), Brussels, Belgium, October 31–November 4, 2018, 2018, pp. 3911–3921. URL: https://aclanthology.org/D18-1425/. doi:10.18653/v1/D18-1425.

[10] F. Bang, GPTCache: An open-source semantic cache for LLM applications enabling faster answers and cost savings, in: L. Tan, D. Milajevs, G. Chauhan, J. Gwinnup, E. Rippeth (Eds.), Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023), Association for Computational Linguistics, Singapore, 2023, pp. 212–218. URL: https://aclanthology.org/2023.nlposs-1.24/. doi:10.18653/v1/2023.nlposs-1.24.

[11] J. Hyde, Cache control, Pentaho Mondrian Documentation, 2011. URL: https://mondrian.pentaho.com/documentation/cache_control.php, author: Julian Hyde; last modified by Luc Boudreau, August 2011. Accessed 2026-02-20.

[12] M. Golfarelli, S. Rizzi, Data Warehouse Design: Modern Principles and Methodologies, McGraw Hill Professional, 2009.

[13] H.-J. Lenz, A. Shoshani, Summarizability in OLAP and statistical data bases, in: Proceedings of the 9th International Conference on Scientific and Statistical Database Management (SSDBM '97), Olympia, Washington, USA, August 11–13, 1997, IEEE Computer Society, 1997, pp. 132–143. URL: https://doi.org/10.1109/SSDM.1997.621175. doi:10.1109/SSDM.1997.621175.

[14] T. B. Pedersen, C. S. Jensen, C. E. Dyreson, Extending practical pre-aggregation in on-line analytical processing, in: Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99), Edinburgh, Scotland, UK, September 7–10, 1999, 1999, pp. 663–674. URL: https://www.vldb.org/conf/1999/P62.pdf.

[15] Transaction Processing Performance Council, TPC Benchmark™ DS Standard Specification, Transaction Processing Performance Council, 2021. URL: https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v3.2.0.pdf, version 3.2.0.

[16] P. O'Neil, B. O'Neil, X. Chen, Star Schema Benchmark, Technical Report, University of Massachusetts Boston, 2009. URL: https://www.cs.umb.edu/~poneil/StarSchemaB.PDF, revision 3.

[17] New York City Taxi and Limousine Commission, TLC trip record data, NYC Taxi & Limousine Commission website (NYC.gov), 2026. URL: https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page, accessed 2026-02-20.

[18] A. Gupta, I. S. Mumick, Maintenance of materialized views: Problems, techniques, and applications, IEEE Data Engineering Bulletin 18 (1995) 3–18. URL: https://dblp.org/rec/journals/debu/GuptaM95.html.

[19] S. Rizzi, E. Gallinucci, CubeLoad: A parametric generator of realistic OLAP workloads, in: Advanced Information Systems Engineering (CAiSE 2014), volume 8484 of *Lecture Notes in Computer Science*, Springer, Cham, 2014, pp. 610–624. URL: https://link.springer.com/chapter/10.1007/978-3-319-07881-6_41. doi:10.1007/978-3-319-07881-6_41.

[20] Y. Kotidis, N. Roussopoulos, Dynamat: A dynamic view management system for data warehouses, in: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD '99), Philadelphia, PA, USA, June 1–3, 1999, Association for Computing Machinery, 1999, pp. 371–382. URL: https://doi.org/10.1145/304182.304215. doi:10.1145/304182.304215.

[21] J. Goldstein, P.-Å. Larson, Optimizing queries using materialized views: A practical, scalable solution, in: Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD 2001), Santa Barbara, CA, USA, May 21–24, 2001, Association for Computing Machinery, 2001, pp. 331–342. URL: https://doi.org/10.1145/375663.375706. doi:10.1145/375663.375706.

[22] M. Pourreza, D. Rafiei, DIN-SQL: Decomposed in-context learning of text-to-SQL

with self-correction, in: Advances in Neural Information Processing Systems, volume 36, 2023. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/72223cc66f63ca1aa59edaec1b3670e6-Abstract-Conference.html.

[23] S. Chaturvedi, A. Chadha, L. Bindschaedler, SQL-of-thought: Multi-agentic text-to-SQL with guided error correction, in: Deep Learning for Code in the Agentic Era (NeurIPS 2025 Workshop: DL4C), 2025. URL: https://arxiv.org/abs/2509.00581.

[24] M. Francia, E. Gallinucci, M. Golfarelli, S. Rizzi, VOOL: A modular insight-based framework for vocalizing OLAP sessions, Information Systems 129 (2025) 102496. URL: https://www.sciencedirect.com/science/article/pii/S0306437924001546. doi:10.1016/j.is.2024.102496.

[25] Cube, Introduction, Cube Documentation, 2026. URL: https://cube.dev/docs/product/introduction, last modified 2026-02-19 (Git history). Accessed 2026-02-20.

[26] dbt Labs, Inc., dbt semantic layer, dbt Developer Hub, 2026. URL: https://docs.getdbt.com/docs/use-dbt-semantic-layer/dbt-sl, last updated 2026-02-19. Accessed 2026-02-20.

[27] A. K. Chandra, P. M. Merlin, Optimal implementation of conjunctive queries in relational data bases, in: Proceedings of the 9th Annual ACM Symposium on Theory of Computing (STOC '77), Boulder, Colorado, USA, May 2–4, 1977, Association for Computing Machinery, 1977, pp. 77–90. URL: https://doi.org/10.1145/800105.803397. doi:10.1145/800105.803397.

[28] A. C. Klug, Equivalence of relational algebra and relational calculus query languages having aggregate functions, Journal of the ACM 29 (1982) 699–717. URL: https://doi.org/10.1145/322326.322332. doi:10.1145/322326.322332.

[29] V. Harinarayan, A. Rajaraman, J. D. Ullman, Implementing data cubes efficiently, in: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD '96), Montreal, Quebec, Canada, June 4–6, 1996, Association for Computing Machinery, 1996, pp. 205–216. URL: https://doi.org/10.1145/233269.233333. doi:10.1145/233269.233333.