Transition from Batch to Real-Time Analytics

Implications for Analytics Data Modeling

Data Analytics
Master's Degree Programme in Information and Communication Technology
Department of Computing, Faculty of Technology
Master of Science in Technology Thesis

Author:
Helmer Haapala

Supervisors:
Ph.D. Viktoriia Shubina
D.Sc. Tuomas Mäkilä

06.04.2026
Turku

Master's thesis

Subject: Information and Communication Technology
Author(s): Helmer Haapala
Title: Transition from Batch to Real-Time Analytics: Implications for Analytics Data Modeling
Supervisor(s): Ph.D. Viktoriia Shubina, D.Sc. Tuomas Mäkilä
Number of pages: 95 pages
Date: 06.04.2026

This thesis examines how analytics-layer data modeling changes when batch-oriented processing is reduced or removed in favor of continuous or near-real-time analytics. Batch-oriented architectures have historically supported analytical interpretation by providing temporally stable snapshots through scheduled extraction, transformation, and loading processes that reconcile data before exposure. As real-time and streaming-oriented architectures become more common, these conditions weaken, making it necessary to reconsider how analytical truth, closure, revision, and semantic consistency are maintained when the analytical state is continuously evolving.

The study addresses this problem through two complementary methods: a systematic literature review conducted in accordance with PRISMA 2020 guidelines and a comparative simulation study. The literature review identifies and analyzes prior research on analytics-layer modeling under real-time or continuous processing constraints, while the simulation study evaluates how alternative architectural patterns behave under a shared set of business requirements. To support this analysis, the thesis proposes a six-dimension framework consisting of analytical truth, temporal closure, history mutability, semantic stabilization, analytical abstraction, and semantic consistency.

The literature review results point to several distinct architectural patterns that preserve different aspects of interpretability while relaxing others, including closed snapshot warehouses, open evolving streams, window-bounded streams, log-consistent HTAP architectures, and virtual semantic snapshots. The simulation study further demonstrates that, although architectures differ in their native state, they can be adapted to satisfy similar business requirements, including requirements commonly associated with batch-native behavior. What differs is not primarily the user-facing analytical contract that can be achieved, but the architectural means by which it is achieved. This includes the maintenance burden, closure mechanism, and the need for supporting intermediary structures.

The thesis offers three related conceptual contributions. First, it proposes a six-dimension framework covering analytical truth, temporal closure, history mutability, semantic stabilization, analytical abstraction, and semantic consistency, which makes the temporal and semantic assumptions of different analytics architectures explicit and comparable across paradigms. Second, it applies this framework to argue that the transition to real-time analytics represents not only a change in processing speed but a change in the interpretability conditions of analytics-layer models: the central question shifts from whether familiar analytical requirements can still be met to how truth, closure, revision, and semantic consistency must be explicitly maintained when the analytical state can no longer be assumed closed. Third, the simulation study demonstrates that while architectures with different native semantics can be adapted to satisfy similar analytical contracts, doing so may relocate rather than eliminate the need for stabilization, a finding with direct implications for how real-time architecture decisions are evaluated in practice.

Key words: Analytics data modeling, real-time analytics, batch processing, semantic consistency, temporal closure, systematic literature review, simulation

Table of contents

List of Tables

List of Figures

1 Introduction

Business intelligence and analytics systems have traditionally been built on the assumption that analytical data can be collected, reconciled, and exposed in periodic batches. In practice, this assumption has most often been implemented through batch-oriented analytics architectures, in which data is extracted, transformed, and loaded (ETL) into structured data warehouses on scheduled cycles measured in hours or days. Within such settings, data models such as dimensional schemas and normalized analytical structures have been optimized for historical analysis, predictable refresh cycles, and query performance over relatively stable datasets (Inmon, 2002, p. 31; Kimball & Ross, 2013, pp. 2–6). Batch-oriented ETL does more than move data from source systems into analytical storage: it also stabilizes the analytical state by cleaning, conforming, and reconciling data before exposure, allowing late-arriving records and source-system corrections to be resolved retrospectively (Kimball & Ross, 2013, pp. 2–6, 19–21, 62–63).

The growing adoption of real-time and streaming analytics challenges these batch-oriented assumptions. When data is ingested and exposed continuously rather than through periodic reconciliation cycles, the analytics layer can no longer rely on batch-based reconstruction of historical state as its primary stabilizing mechanism. Instead, analytical models must contend with evolving facts, mutable dimensional context, late or out-of-order events, and the possibility that analytical results may change after they have already been consumed. In such settings, data freshness is no longer merely an operational concern but a modeling constraint that directly affects semantic interpretation (Qu & Wang, 2024, pp. 110–112). Rather than asking only whether traditional dimensional models can be accelerated or adapted, it becomes necessary to examine which temporal and semantic assumptions batch-oriented processing stabilizes, what challenges emerge when that stabilizing role is reduced or removed, and how alternative analytical architectures re-establish or replace those conditions. More specifically, it becomes unclear how analytical truth, temporal closure, historical revision, and semantic consistency can be maintained when the analytical state is continuously evolving.

While existing research has extensively examined real-time analytics from the perspective of data infrastructure, processing engines, and pipeline design (Akash Vijayrao Chaudhari & Pallavi Ashokrao Charate, 2025, pp. 809–811; Dileep Kumar Siripurapu, 2025, pp. 2484–2486; Qu & Wang, 2024, pp. 110–112), the implications of these architectural changes for analytics-layer data modeling remain underexplored. Much of the literature continues to

assume batch-based semantics, even when proposing near-real-time or streaming solutions. As a result, the question of how analytical meaning is defined, stabilized, and communicated in the absence of batch processing remains insufficiently addressed. The gap is therefore not only technological but conceptual: if batch-oriented architectures have historically aligned assumptions about truth, closure, correction, and consistency before analytical exposure, then continuous analytics requires those assumptions to be made explicit and reconsidered. Addressing this gap is important because the analytics layer is where data becomes analytically meaningful and is translated into a form that supports decision-making in enterprise environments.

1.1 Purpose and objectives

This study examines how analytics-layer data modeling changes when batch-oriented reconstruction is reduced or removed, with particular attention to how analytical truth, closure, revision, and semantic consistency are maintained under continuous or near-real-time data arrival. Rather than evaluating real-time analytics only as a latency problem, the study examines how reduced batch reconstruction affects the conditions under which analytics-layer models remain interpretable.

The research questions of this study are:

> **RQ1**: What temporal and semantic assumptions are embedded in analytics-layer models, and how does batch-based processing support these assumptions?
> **RQ2:** What modeling challenges emerge when batch-based reconstruction of analytical history is removed?
> **RQ3:** What modeling strategies or paradigms have been proposed to address these challenges in continuous analytics environments?

To address these questions, the thesis combines a systematic literature review with a comparative simulation study. The aim is to clarify how different architectural paradigms preserve interpretability under continuous or near-real-time conditions. The contribution of the study lies in making these trade-offs and interpretability conditions explicit across alternative architectural paradigms. To support this, the thesis proposes an author-derived six-dimension framework and uses it to interpret both the literature and the simulation scenarios.

1.2 Research scope

This study is situated at the intersection of traditional analytics data modeling and real-time analytics architectures, with a specific focus on the analytics layer. In this thesis, the analytics layer denotes the modeling and semantic structures through which operational data is transformed into interpretable analytical meaning, including analytical schemas, semantic definitions, and the abstraction conventions by which results are exposed to downstream consumers. The central concern is therefore not real-time processing in general, but the conditions under which analytics-layer models remain interpretable when batch-oriented stabilization is reduced or removed.

Accordingly, the thesis does not attempt to provide an end-to-end account of real-time data infrastructure. Topics such as ingestion mechanisms, change data capture, stream processing, and physical storage design are considered only insofar as they affect the temporal and semantic assumptions addressed by the research questions. The primary unit of analysis remains the analytics-layer model and its implied conditions of truth, closure, revision, and consistency, not specific pipeline implementation choices. Organizational and business-oriented implications, such as managerial decision-making or the business value of technologies, are likewise outside the scope of this thesis.

For the purposes of this research, real-time analytics refers to analytical operation under continuous or near-continuous data arrival, where latency targets constrain the extent to which analytical state can be treated as closed and retrospectively reconciled. This definition includes near-real-time and micro-batch settings when they function as substitutes for a fully reconstructed batch snapshot, and it treats streaming-oriented paradigms as relevant when they directly involve a layer intended for analytical consumption. By contrast, contributions focused primarily on algorithmic streaming, event processing without an analytical consumption layer, or domain-specific operational applications without transferable analytics-layer modeling implications fall outside the scope of direct analysis.

These delimitations keep the analysis focused on how analytics-layer models adapt when batch-based stabilization weakens. This framing keeps the analysis tightly aligned with the research questions and supports the broader aim of the thesis: to clarify which assumptions batch-oriented architectures have historically stabilized, what challenges emerge when those assumptions weaken, and how alternative architectures respond under real-time constraints.

1.3 Thesis structure

The thesis proceeds from conceptual grounding to empirical comparison. It first establishes the modeling assumptions, such as analytical truth and temporal closure, that batch-oriented analytics has historically relied on, then examines how those assumptions are treated in the literature on real-time analytics and finally tests their practical implications through simulation.

Chapter 2 establishes the theoretical background for the study by introducing core analytics data modeling paradigms. It reviews traditional multidimensional and relational analytics models, examines the role of semantic-layer-driven analytics, and identifies the temporal and semantic assumptions that batch-oriented architectures have historically stabilized. The chapter concludes by showing how continuous data arrival challenges these assumptions and by introducing the conceptual basis for the analytical framework used later in the thesis.

The methodology is covered in Chapter 3. Two complementary methods are used: a systematic literature review following PRISMA 2020 guidelines, and a comparative simulation study. The chapter explains why a qualitative synthesis was preferred over a statistical meta-analysis given the fragmented state of the literature and outlines how the simulation was designed to keep architectural comparisons grounded in shared conditions rather than architecture-specific assumptions.

Chapter 4 presents the results of the systematic literature review. It details the search strategy, inclusion and exclusion criteria, and study selection process, followed by a thematic analysis of the included studies. Using the analytical framework developed in the thesis, the chapter synthesizes how existing research positions different architecture patterns along the six dimensions and identifies the main interpretability challenges and architectural responses associated with reduced batch reconstruction.

Chapter 5 then takes those patterns into a controlled simulation setting. Five architecture families spanning snapshot-oriented, streaming-oriented, transactional, and semantic-layer-based approaches, together with a batch reference and a ground-truth implementation, are exposed to the same synthetic source history and evaluated against a shared set of business requirements. The results show both where architectures differ in their native behavior regarding truth, closure, and revision and how far they can be adapted when requirements demand it.

Chapter 6 discusses the findings by integrating the results of the systematic literature review and the simulation study. It interprets the results in relation to the research questions, highlights the main modeling trade-offs that emerge when batch-oriented alignment is weakened, and considers the limitations and broader implications of the study. The chapter also outlines directions for future research.

Chapter 7 concludes the thesis by summarizing the main contributions of the study and reflecting on their implications for analytics-layer modeling theory and practice in real-time analytical environments.

2 Analytics data modeling

Analytics data modeling refers to the conceptual and logical structuring of data for analytical consumption, where the primary objective is to support insight-generation and decision making conducted via analytics systems. (Kimball & Ross, 2013, pp. 7–16). Analytics systems operate on transformed representations of operational data, enabling users to analyze trends, compare entities, and evaluate performance over time. This architectural separation from transactional systems historically justified delayed data availability and batch-oriented processing, allowing analytics-layer models to assume that data is complete, reconciled, and temporally stable at the point of analysis (Chaudhuri & Dayal, 1997, pp. 65–67; Inmon, 2002, pp. 31–37). Within this architecture, the analytics layer occupies a central role as the location where business meaning is formalized. It is the layer where raw or semi-processed data is transformed into analytically meaningful constructs such as facts, dimensions, measures, and hierarchies. Whether implemented through multidimensional cubes or relational star schemas, the analytics layer abstracts the structural complexity of source systems and exposes a consistent analytical interface to end users. Modeling decisions at this layer directly shape how business phenomena are represented, aggregated, and interpreted, and therefore constrain the analytical questions that can be meaningfully answered (Kimball & Ross, 2013, pp. 7–16).

Traditionally, analytics-layer data models have been developed within the context of batch-oriented data warehousing, where analytical datasets are populated through scheduled ETL-processes rather than continuous updates (Chaudhuri & Dayal, 1997, pp. 65–67; Inmon, 2002, pp. 31–37). Within this setting, different modeling approaches represent alternative ways of structuring analytical datasets for efficient aggregation and interpretation. The following sections first introduce two foundational analytics paradigms and then examine how their batch-era assumptions are challenged in continuous and real-time analytics. This chapter primarily addresses RQ1 by establishing the temporal and semantic assumptions embedded in analytics-layer models and by clarifying the role batch-oriented processing has historically played in stabilizing those assumptions.

2.1 Multidimensional analytics

Multidimensional analytics is one of the foundational paradigms for analytical decision support and is commonly associated with Multidimensional Online Analytical Processing (MOLAP). The paradigm emerged from the recognition that analytical workloads differ

structurally from transactional workloads: instead of executing short, selective updates and lookups, analytical users require exploratory queries that scan large datasets, compute aggregates, and compare results across multiple perspectives (Chaudhuri & Dayal, 1997, pp. 65–67; Inmon, 2002, pp. 31–37). To support these requirements, OLAP research and early commercial systems adopted a multidimensional view of business data, where analytical meaning is represented through measures and dimensions rather than normalized entity-relationship structures (Chaudhuri & Dayal, 1997, pp. 67–69; Codd et al., 1993, pp. 4–5).

In the multidimensional model, measures represent numerical phenomena of interest, such as revenue, sales volume, or inventory levels, while dimensions provide contextual axes along which these measures are analysed, such as time, product, region, or organizational unit. Dimensions are frequently hierarchical, enabling consistent navigation between levels of detail (e.g. day, month, year), and allowing users to interpret results both at granular and aggregated levels (Chaudhuri & Dayal, 1997, pp. 68–69; Shoshani, 1997, pp. 185–187). This perspective aligns closely with how business analysts formulate questions and interpret results in terms of categories, time periods, and comparative slices of performance (Codd et al., 1993, pp. 6–7).

A key contribution of OLAP as a paradigm is the definition of interactive analytical operations that are natural in a multidimensional space. Classical OLAP operations include slice-and-dice (filtering and projecting across dimensions), pivot (re-orienting dimensional views), and hierarchical roll-up and drill-down (aggregating to higher levels or navigating to finer detail). These operations are widely referenced as defining characteristics of OLAP interaction and serve as the conceptual interface between users and multidimensional analytical structures (Chaudhuri & Dayal, 1997, pp. 67–69; Codd et al., 1993, pp. 6–7; Vassiliadis & Sellis, 1999, pp. 64–65).

MOLAP systems implement this conceptual model directly by representing the analytical dataset as a multidimensional cube (or a closely related set of cube structures) rather than as a purely relational schema. MOLAP systems use specialized multidimensional storage structures, commonly described as array-based or cube-indexed representations designed to accelerate aggregation-heavy queries (Chaudhuri & Dayal, 1997, pp. 65–74). The cube abstraction is not merely a user-facing metaphor: it strongly influences the physical organization of data and the execution strategies used to answer analytical queries.

A defining technical characteristic of MOLAP is the extensive reliance on precomputation. Because analytical workloads frequently involve repetitive aggregation across common groupings, MOLAP systems often compute and store a large set of aggregates in advance as part of cube processing. This design shifts computational cost away from query time and toward processing time, enabling low-latency query responses and predictable interactive performance for common analytical operations (Chaudhuri & Dayal, 1997, pp. 65–66; Vassiliadis & Sellis, 1999, pp. 64–65). The theoretical basis for this strategy is also reflected in the database research literature on data cube computation and multidimensional aggregation, where precomputed group-bys and systematic aggregate generation are treated as central optimization problems for OLAP-style queries (Gray et al., 1996, pp. 29–31).

The performance benefits of precomputation are accompanied by structural and storage considerations. Multidimensional cubes can become sparse when the dimensionality grows or when many combinations of dimension values contain no observations. Classical OLAP literature highlights sparsity and storage utilization as one of the central technical constraints of MOLAP engines, motivating compression techniques and hybrid representations that store dense regions efficiently while indexing sparse regions through auxiliary structures (Chaudhuri & Dayal, 1997, pp. 65–67; Vassiliadis & Sellis, 1999, pp. 64–65). These design challenges reflect a fundamental trade-off: MOLAP achieves fast analytical retrieval by committing to a multidimensional representation, but that representation can become less efficient as real-world data becomes more heterogeneous or dimensional models expand.

From the perspective of analytics-layer modeling, MOLAP is often characterized by its integrated treatment of structure and execution. Dimensions, hierarchies, and measures are defined as first-class constructs, and the cube processing logic materializes these constructs into a consistent analytical structure used by query engines and front-end tools. This tight integration supports high-performance interactive analysis and allows OLAP engines to implement aggregation and navigation operations efficiently, often without relying on general-purpose relational optimization alone (Chaudhuri & Dayal, 1997, pp. 68–69; Shoshani, 1997, pp. 185–187). In this sense, MOLAP systems represent an analytics paradigm where the model is not only a logical abstraction, but also a guiding blueprint for storage and query evaluation.

From the perspective of analytics-layer assumptions, multidimensional analytics operationalizes analytical correctness as retrospective snapshot correctness: analytical results

are interpreted as correct relative to the processed cube state rather than relative to an evolving operational source. Cube processing functions as an explicit temporal closure mechanism, delimiting when the analytical state is considered complete and suitable for interpretation (Chaudhuri & Dayal, 1997, pp. 68–69; Shoshani, 1997, pp. 185–187). Because cube materialization constructs a consistent internal representation of dimensions and aggregates, semantic meaning is largely stabilized inside the cube during processing, and semantic consistency is typically strong within each processed snapshot. This design yields high repeatability for a given processed state, but it also implies a reconstructive relationship to history: changes in source data or dimensional definitions are commonly reflected through reprocessing and regeneration of the analytical snapshot.

The historical role of MOLAP is therefore best understood as an early analytics-layer design centered on representing business phenomena as multidimensional structures optimized for aggregation-heavy exploration. As data warehousing environments expanded in scale and organizations increasingly preferred to reuse relational infrastructure, alternative approaches emerged that sought to retain dimensional semantics while relocating execution to relational database engines (Kimball & Ross, 2013, pp. 7–16).

2.2 Relational analytics

Relational analytics, commonly referred to as Relational Online Analytical Processing (ROLAP), denotes an analytics paradigm in which multidimensional analytical workloads are supported through relational representations and relational query semantics. Rather than materializing analytical structures into proprietary multidimensional storage, ROLAP systems persist analytical data in relational form and evaluate analytical queries by joining and aggregating relational tables. Early OLAP literature characterizes this approach as an attempt to retain the analytical interaction model of OLAP while leveraging the maturity and general-purpose capabilities of relational data management technology (Chaudhuri & Dayal, 1997, pp. 68–69; Shoshani, 1997, pp. 185–187).

At the modeling level, ROLAP commonly realizes multidimensional meaning through dimensional relational schemas, where facts and dimensions are expressed as tables and not as cube-native structures. The dominant pattern is the star schema family, in which a central fact table records measurable observations at a defined grain and surrounding dimension tables provide descriptive context used for filtering, grouping, and navigation. OLAP modeling surveys emphasize that these schema forms provide a practical bridge between

multidimensional analytical concepts and relational implementation, enabling cube-like analysis to be expressed through relational joins and aggregations (Chaudhuri & Dayal, 1997, pp. 65–67; Vassiliadis & Sellis, 1999, pp. 64–65).

In enterprise analytics practice, relational analytics became closely associated with dimensional modeling as a design discipline concerned with semantic clarity, consistency, and reuse across business processes. Kimball and Ross (2013, pp. 7-16) formalize this perspective by presenting dimensional modeling as a method for structuring analytics-layer data around business processes, where analytical meaning is expressed through facts representing measurements of process performance and dimensions representing shared descriptive context (Kimball & Ross, 2013, pp. 7–16). This interpretation extends ROLAP beyond a technical execution approach by treating the relational schema itself as the primary analytics-layer abstraction through which analytical questions are structured and answered.

A central modeling principle in Kimball-style relational analytics is the explicit definition of fact table grain. Grain specifies what a single row in a fact table represents and thereby determines the valid aggregation behavior of measures across dimensional perspectives. Establishing grain is therefore a foundational step in ensuring analytical correctness, as inconsistent or ambiguous grain leads directly to misleading aggregates and unstable business interpretation (Kimball & Ross, 2013, pp. 11–16). Within this framework, fact tables are not a uniform construct but may take different semantic roles depending on the analytical intent. Kimball and Ross distinguish between transaction fact tables capturing discrete business events, periodic snapshot fact tables representing state at regular intervals, and accumulating snapshot fact tables describing process lifecycles through milestone progress. These alternatives represent different relational realizations of business phenomena and support different analytical questions about events, states, and process duration (Kimball & Ross, 2013, pp. 60–69).

Relational analytics also emphasizes conformed dimensions as a mechanism for semantic integration and comparability across the analytics layer. A conformed dimension provides a shared definition of a business concept such as customer, product, or calendar time, enabling multiple fact tables to be analysed together within a consistent interpretive framework. This principle supports the construction of an enterprise-wide analytics layer in which metrics remain comparable across reports and departments, and where analytical meaning is stabilized through common dimensional structures while avoiding independent redefining within each

data mart (Kimball & Ross, 2013, pp. 19–21). In addition, relational analytics commonly relies on surrogate keys and controlled history management as mechanisms for stabilizing analytical interpretation on dimensional level. Dimensional entities are typically represented through surrogate identifiers rather than operational business keys, enabling integration across heterogeneous source systems and preserving stable joins even when source identifiers change. This design also supports structured representation of evolving descriptive context through slowly changing dimension techniques, where attribute changes can be captured as versioned dimension records to preserve historically valid interpretation of facts (Kimball & Ross, 2013, pp. 62–63).

Relational analytics further depends on aggregation validity as a semantic constraint. Measures differ in their additivity across dimensions, meaning that not all metrics can be meaningfully summed across time or other hierarchies without additional rules. This property interacts directly with grain definition and dimensional history: incorrect treatment of non-additive measures, or mismatched grain, results in analytically plausible yet semantically incorrect outputs. As a result, ROLAP modeling does not only define relational structures, but also encodes assumptions about which aggregations are valid and under which dimensional contexts (Kimball & Ross, 2013, pp. 11–16). Table 1 summarizes the resulting architectural differences between the classical MOLAP and ROLAP paradigms, focusing on where aggregation is computed, how results are stored, and how analytical state is refreshed and stabilized.

Table 1. Architectural characteristics of MOLAP and ROLAP (derived from Chaudhuri & Dayal, 1997)

Attribute	**MOLAP**	**ROLAP**
Storage	Multidimensional arrays	Relational tables
Aggregation	Precomputed	At query execution
Update mechanism	Cube processing	ETL reload
Flexibility	Low	High
Scalability	Limited	High
Semantic stabilization	Processing snapshot	Schema + ETL snapshot

As shown in Table 1, MOLAP favours performance and consistency through pre-materialized snapshots, whereas ROLAP favours flexibility by computing aggregations at query execution under schema and ETL constraints. These aggregation constraints are reflected not only in how facts and dimensions are modelled, but also in how analytical results are materialized and

maintained. ROLAP preserves the snapshot-oriented correctness goal of MOLAP, in which analytical correctness is defined relative to a temporally closed and reconciled analytical state, but stabilizes it through different mechanisms.

Temporal closure is typically implicit and not engine-defined: the analytical state becomes complete at the boundaries of ETL refresh cycles or load windows, after which queries are assumed to evaluate against a sufficiently stable relational snapshot (Kimball & Ross, 2013, pp. 518–519). Consequently, history mutability is usually selectively reconstructive, as late-arriving facts, dimension corrections, and backfills can be applied by reloading partitions or reprocessing affected tables. Within this paradigm, semantic stabilization is primarily achieved through the combined effect of schema design and batch reconciliation, yielding deterministic outputs as long as the warehouse snapshot remains stable.

2.3 Semantic layer driven analytics

While Kimball-style dimensional modeling remains a dominant conceptual foundation for analytics-layer design, contemporary relational analytics environments increasingly differ from the classical warehouse-centered interpretations in where analytical meaning is defined and how it is enforced. In modern relational analytics, dimensional schemas continue to provide structural clarity for facts and dimensions, but analytical interpretation is increasingly mediated through semantic abstractions that formalize measures, relationships, and calculation rules above the storage layer. (Chen et al., 2012, pp. 1165–1167). Classic OLAP and data warehouse architectures largely treated the warehouse schema as the primary carrier of analytical meaning, with metric definitions and interpretive rules embedded in relational structures and reporting logic (Inmon, 2002, pp. 31–37; Kimball & Ross, 2013, pp. 2–6). In contrast, modern relational analytics increasingly externalizes these definitions into governed semantic layers, enabling consistent interpretation across tools while allowing the underlying storage layer to remain optimized for scalable persistence and query execution (Bomma, 2023, pp. 94–95).

This architectural separation is typically realized through multi-layer analytics architectures in which data ingestion, transformation, and analytical serving are treated as distinct responsibilities (Bomma, 2023, pp. 94–95). Persistent storage is often implemented using cloud data warehouses or lakehouse platforms, while transformation pipelines curate analytically consumable datasets that preserve dimensional structure (Armbrust et al., 2021,

pp. 1–2). Above these layers, a semantic layer defines canonical entities, relationships, and standardized measures that act as a contract between data producers and analytical consumers. The key change is that dimensional databases or warehouse schemas are no longer the location of analytical meaning. Instead, semantic stabilization occurs in a separate abstraction layer that governs how facts are interpreted, aggregated, and related at query time. This relocation improves reuse and metric consistency across analytical tools but also introduces an implicit assumption: that analytical queries operate over a sufficiently stable underlying data state such that semantic definitions yield repeatable and interpretable results. Figure 1 illustrates a modern, layer-based analytics environment as described by Bomma (2023).

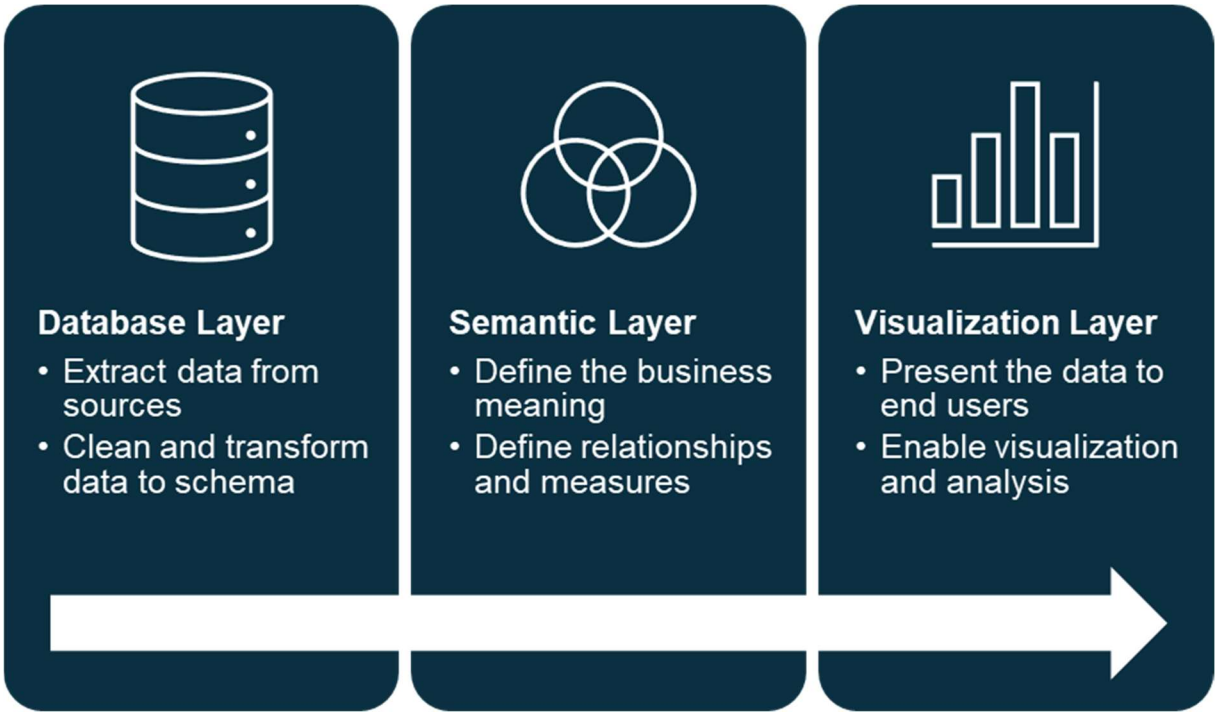

Figure 1. Modern analytics environment (based on Bomma, 2023)

As shown in Figure 1, the semantic layer becomes the primary interface through which analytical meaning is defined and enforced. Rather than embedding business rules directly in the warehouse schema, modern environments centralize measure definitions, relationship logic, and calculation behavior in a governed semantic model. The relational warehouse increasingly functions as a scalable persistence layer, while analytical computation is mediated through intermediate in-memory analytical engines. Instead of resolving analytical queries solely through the underlying database at query time, modern systems frequently execute aggregations, measure logic, and semantic interpretation within in-memory or columnar analytical layers that maintain their own optimized representations of analytical

state. This architectural evolution aligns with broader developments in analytical database systems, where column-oriented storage layouts and vectorized execution have been widely adopted to improve analytical performance by reducing I/O and accelerating scan-and-aggregate workloads (Abadi et al., 2009, pp. 1664–1665). Similarly, cloud-native warehouse architectures have emphasized separation of storage and compute, enabling elastic analytical serving that is decoupled from ingestion and persistence concerns (Armbrust et al., 2021, pp. 1–2; Dageville et al., 2016, pp. 215–218). The key implication is that relational analytics is no longer equivalent to direct runtime evaluation over relational base tables. Even when the underlying data remains stored in star schemas, query-time meaning is increasingly governed by a computational layer that caches, materializes, or incrementally maintains analytical representations in memory. This pattern is also visible in real-time and low-latency analytical systems, where interactive OLAP-style queries are achieved through specialized serving layers that maintain execution-optimized internal state rather than relying on ad hoc evaluation over raw relational tables (Im et al., 2018, pp. 169–172; Yang et al., 2017, pp. 1704–1706).

This shift can be interpreted as a relocation of the semantic stabilization: instead of treating the warehouse schema and ETL process as the primary site where analytical meaning is fixed, modern architectures increasingly treat governed semantic models as the canonical contract for metric definitions, relationships, and calculation behavior (Bomma, 2023, pp. 94–95; Chen et al., 2012, pp. 1165–1167). However, relocating semantics upward does not eliminate temporal assumptions, but rather changes how they are satisfied. Semantic models still assume the underlying analytical state to support repeatable interpretation, either because the storage layer is refreshed in stable increments, or because intermediate serving layers materialize and maintain execution-optimized analytical state (Abadi et al., 2009, pp. 1664–1665; Armbrust et al., 2021, pp. 1–2; Dageville et al., 2016, pp. 215–218). This dependency becomes central under real-time constraints, where the stability of the analytical state can no longer be assumed to coincide with batch boundaries.

2.4 Towards real-time analytics

The shift toward real-time analytics is most often motivated by a simple operational requirement: analytical information is expected to reflect business reality with minimal delay. As organizations increasingly make decisions in response to rapidly changing market conditions, dynamic pricing, online customer behavior, and continuously evolving operational

status, the value of analytics becomes tightly coupled with freshness. For example, a sales dashboard that updates only after a pricing change, demand spike, or service disruption has already passed may still be technically correct, but no longer timely enough to support the decision it was meant to inform. In this setting, latency is not only a technical performance characteristic, but an interpretive constraint that shapes what analytical results can be trusted to represent at the moment of consumption. Recent real-time data warehousing literature therefore frames freshness as a first-class objective, emphasizing that reducing end-to-end delay changes both the architecture of analytical systems and the conditions under which correct analytical meaning can be produced (Qu & Wang, 2024, pp. 110–112).

A key insight in the dimensional warehousing literature is that real-time is not a single delivery mode, but a spectrum of latency targets that imply fundamentally different architectural choices. Kimball and Ross (2013, p. 521) distinguish between daily, intra-day, and instantaneous delivery, arguing that each category requires a different extraction and serving design and introduces different expectations of correctness and completeness. Daily delivery corresponds to conventional batch ETL, where the warehouse is refreshed from reconciled and stable extracts. Intra-day delivery refreshes data multiple times per day, often through micro-batches, but does not guarantee that the analytical view represents the full and final truth of the operational system at every moment. Instantaneous delivery, by contrast, implies synchronous reflection of the operational system and is commonly approached through enterprise information integration (EII) patterns rather than batch-oriented warehousing pipelines (Kimball & Ross, 2013, p. 522).

This categorization highlights why accelerating a batch warehouse is not merely a scaling exercise. As latency requirements tighten, the conditions that normally stabilize analytical meaning begin to weaken. Kimball and Ross explicitly frame real-time delivery as a trade-off space in which reducing latency often requires reducing or postponing reconciliation, validation, and conformance steps that would otherwise occur before the data becomes analytically visible. In aggressive low-latency designs, it may become necessary to retain a parallel batch-oriented flow that subsequently overwrites or reconciles earlier real-time data to restore historical correctness (Kimball & Ross, 2013, pp. 522–523).

These constraints also cause temporal and semantic anomalies between facts and dimensions. In real-time environments, transactional events may arrive before their descriptive context has been updated or integrated. Kimball and Ross recommend allowing facts to be posted using

prior dimension versions when available, or with generic placeholder dimension members otherwise, explicitly acknowledging that there may exist a window in which dimensions do not accurately describe the newly arrived facts. This pattern is closely aligned with the broader dimensional modeling treatment of late arriving dimensions, but real-time delivery makes such situations routine rather than exceptional. (Kimball & Ross, 2013, p. 523)

At the serving layer, Kimball and Ross describe a pragmatic solution for bridging the gap between historical analysis and current-state requirements: real-time partitions as extensions of conventional fact tables. In this design, a hot partition stores activity since the last static warehouse update and is physically or administratively separated from the historical portion of the fact table. To support continuous ingestion, the hot partition is indexed lightly or not at all, while still supporting interactive queries by keeping the partition small enough to be pinned in memory. The objective is to preserve a seamless analytical time series up to the current instant without reprocessing the full warehouse repeatedly. The approach is described as applicable to transaction and periodic snapshot facts, where the hot segment can later be merged into the historical table as the next closed period. (Kimball & Ross, 2013, pp. 524–525)

These temporal patterns do not remove the dimensional paradigm but rather reveal which parts of its stability are normally supplied by batch closure. Conventional batch ETL provides an implicit temporal boundary that enforces snapshot semantics: facts and dimensions are integrated, cleaned, and reconciled before analytical interpretation is allowed. As the pipeline becomes incremental and continuously open, the warehouse may no longer be able to assume that all relevant events have arrived, that dimension context is complete, or that business-rule corrections have been applied. Consequently, analytical correctness becomes contingent on explicit design decisions about when data is considered sufficiently complete for consumption and how revisions are handled after the fact.

However, Kimball and Ross' discussion makes clear that these challenges are not purely infrastructural: they directly affect the interpretability of analytics-layer models. When the analytical state is no longer guaranteed to be closed, the semantic layer and dimensional abstractions must operate over potentially provisional facts and evolving dimensional context, weakening the batch-era assumption that analytical meaning is evaluated against a reconciled snapshot. Consequently, reducing refresh frequency to near real-time or continuous operation should be understood as a qualitative shift in the premises of analytics-layer modeling. Batch-

based analytics derives stability from reconstruction: it can reconcile, restate, and reprocess historical meaning until the snapshot is considered correct. Real-time analytics must instead define how correctness is established when the analytical state cannot be assumed closed, when events may arrive late, and when dimensions and facts may evolve concurrently. Yet this question remains insufficiently addressed in the literature, which has focused more extensively on real-time processing and infrastructure than on how analytics-layer models define and preserve correct analytical meaning under such conditions.

Because these interpretability conditions are treated inconsistently across the literature and are often left implicit in system designs, a systematic synthesis is required to make the underlying assumptions explicit and comparable across paradigms.

2.5 Temporal and semantic assumptions

To analyse these changes systematically, this thesis characterizes analytics-layer paradigms using six conceptually separable dimensions. These dimensions are proposed by this thesis as an analytical framework. They are derived through conceptual synthesis of the foundational analytics modeling literature reviewed in this chapter, rather than adopted as a named framework from any single source. The intellectual basis for each dimension is therefore grounded in established knowledge about how batch-oriented architectures have historically operationalized analytical meaning, as discussed in Sections 2.1 through 2.4. Accordingly, the dimensions should not be read as direct quotations or standardized terms from the cited authors. Instead, the citations indicate the specific constructs and recurring ideas within that foundational literature from which each dimension is constructed. These dimensions are introduced because the shift towards reduced batch reconstruction changes the conditions under which analytical results remain interpretable: what is treated as correct, when the results are considered complete enough to interpret, whether later revision is permitted, where semantic meaning is stabilized, how that meaning is presented through analytical abstractions, and what stability can be assumed by downstream consumers. Together, the six dimensions form a minimal set that covers these recurring interpretability commitments without collapsing distinct issues into a single category, enabling consistent comparison across heterogeneous designs.

Analytical truth defines what it means for an analytical result to be considered correct. It distinguishes between retrospective snapshot correctness, provisional or evolving correctness, log-consistent transactional correctness, and window-bounded correctness. Analytical truth in

this thesis refers to the criterion under which an analytical result is regarded as correct for interpretation, not an assertion that the system has captured an objective real-time ground truth. Especially under reduced batch reconstruction, correctness is frequently defined relative to a declared temporal scope and processing rule. This dimension is motivated by the classical OLAP framing of analytics as evaluation over a consistent analytical state (Chaudhuri & Dayal, 1997, pp. 65–67) and by dimensional modeling discussions that highlight the interpretive dependence of measures on stable grain and dimensional context (Kimball & Ross, 2013, pp. 11–16).

Temporal closure specifies when the analytical state is treated as complete enough for interpretation. Closure may be explicit (e.g. cube processing), implicit (e.g. load-window boundaries in relational warehousing), or stream-native (e.g. window-bounded or incrementally finalized states). This dimension builds directly on OLAP's snapshot-oriented processing perspective (Chaudhuri & Dayal, 1997, pp. 68–69) and on dimensional warehousing's explicit treatment of latency as a spectrum (daily, intra-day, instantaneous) with differing completeness expectations (Kimball & Ross, 2013, pp. 521–523).

History mutability captures whether analytical history may change after initial exposure. Paradigms differ in whether the historical state is fully recomputable, selectively correctable through backfills and restatements, strictly append-only, or revised through log-based transactional mechanisms. In this thesis, recomputation refers to the capability to reconstruct analytical outputs by rerunning logic over historical inputs. Restatement refers to the intentional replacement of previously exposed outputs due to corrected logic or late-arriving data and backfill denotes the operational act of inserting or reprocessing historical data to enact such restatements. This dimension reflects the practical reality that as latency targets tighten, reconciliation and correction steps may be postponed, requiring later overwriting or reconciliation to restore historical correctness (Kimball & Ross, 2013, pp. 522–523), and it aligns with real-time warehousing literature that frames freshness as changing the conditions under which correct analytical meaning can be produced (Qu & Wang, 2024, pp. 110–112).

Semantic stabilization identifies where analytical meaning is enforced. Depending on the paradigm, stabilization may be implemented through batch ETL and schema conventions, incremental maintenance mechanisms, query-time interpretation, engine-internal semantics, or semantics delegated to source systems and contracts (Chen et al., 2012, pp. 1165–1167). This dimension is central for explaining how systems preserve interpretability when batch-

style reconciliation is reduced or absent, particularly as semantic governance moves upward from the warehouse schema to explicit semantic layers.

Analytical abstraction describes the conceptual model through which users access analytical meaning. Examples include star and snowflake schemas, streaming-adapted dimensional structures, event or log-oriented tables, engine-defined analytical structures, and virtualized analytical views. This dimension follows OLAP literature distinguishing multidimensional interaction models from relational realizations (Chaudhuri & Dayal, 1997, pp. 65–69) and dimensional modeling's position of the star schema as the primary analytics-layer abstraction for business interpretation (Kimball & Ross, 2013, pp. 7–16).

Semantic consistency describes the stability and repeatability provided to analytical consumers. Semantic consistency refers to the degree to which an analytical result preserves a stable meaning under repeated evaluation, essentially whether the same question yields the same answer when re-executed under the same interpretive assumptions. Paradigms vary in whether they offer deterministic snapshot repeatability, eventual convergence, bounded inconsistency, or approximate or contract-based guarantees. Consistency guarantees emphasizes that this stability is a declared contract offered to downstream consumers about when and how results may change. This dimension is implied both by OLAP's emphasis on stable analytical interaction over consistent structures (Chaudhuri & Dayal, 1997, pp. 67–69) and by real-time warehousing discussions that explicitly acknowledge provisional exposure and later reconciliation as latency decreases (Kimball & Ross, 2013, pp. 522–523; Qu & Wang, 2024, pp. 110–112).

Together, these dimensions provide a common basis for comparing classical analytics paradigms with real-time architectures. They make the relevant temporal and semantic assumptions explicit and, in doing so, support the research questions. Chapter 4 applies the framework as the coding scheme for the literature review, and Chapter 5 uses it to interpret the simulation results. In this sense, the framework operationalizes the thesis's research questions by making it possible to examine what batch-based analytics assumes, what changes when those assumptions weaken, and how alternative architectures respond

3 Methodology

The research methods used in this study are twofold. First, a theoretical framework for analytics data modeling in a real-time environment is derived from a Systematic Literature Review (SLR). The literature review is conducted in accordance with the PRISMA 2020 - reporting guidelines (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) as proposed by Page et al. (2021). Following the literature review, an empirical simulation study is conducted to evaluate the practical implications of the identified temporal and semantic architecture patterns. The empirical study applies a controlled synthetic simulation to examine the practical implications of the theoretical findings under shared business-oriented conditions. The empirical simulation study is conducted as a controlled comparative simulation study, following the evaluative research framing of Wohlin and Aurum (2015), the simulation-method perspective of Müller and Pfahl (2008), and the purpose-bound verification and validation principles described by Sargent (2013).

3.1 Systematic literature review

A systematic literature review (SLR) is a formal, transparent, and replicable method for identifying, evaluating, and synthesizing existing research relevant to a clearly defined set of questions or phenomena. Unlike traditional narrative reviews, which are often selective and interpretative, SLRs rely on a structured and protocol-driven approach that minimizes researcher bias and enhances the reliability of the review process. As defined by Kitchenham & Charters (2007), an SLR applies explicit procedures for searching, selecting, and synthesizing empirical evidence. These procedures enable mapping the existing body of knowledge, identifying gaps, and supporting evidence-based decision-making.

The SLR was chosen to provide a transparent and replicable basis for the thesis's conceptual framework and subsequent analysis. By adhering to established guidelines and applying a systematic search and selection protocol, the review aims to capture the full scope of relevant academic contributions while avoiding the pitfalls of selective evidence use. This structured approach enables the study not only to consolidate existing knowledge but also to critically assess the methodological precision and thematic focus of prior research. In doing so, the SLR provides a robust empirical foundation for the subsequent phases of the study and ensures that the conclusions drawn are firmly grounded in the best available evidence.

Conducting the SLR according to the PRISMA 2020 framework provides a well-established, reliable and transparent process to conduct the research and report on the process. PRISMA guidelines comprise a staged research process consisting of identification, screening, eligibility assessment, and final inclusion of studies. During the identification phase, comprehensive searches are performed across relevant academic databases to capture the broadest possible set of publications. Retrieved records are subsequently screened for relevance based on predefined inclusion and exclusion criteria, first at the level of titles and abstracts and later through full-text assessment. The framework emphasizes the importance of defining these criteria in advance to minimize subjective bias and enhance repeatability. Studies that meet all eligibility criteria are incorporated into the final synthesis. The entire selection process is documented using the PRISMA flow diagram, which provides a transparent account of the number of records identified, screened, excluded, and ultimately included. (Page et al., 2021, pp. 3–6).

Because the research questions concern conceptual modeling assumptions and semantic design choices rather than quantifiable effects, and because the studies retrieved through the systematic search are anticipated to be heterogeneous in methodology, architecture type, and reporting convention, a qualitative synthesis is adopted in preference to statistical meta-analysis. To conduct this synthesis systematically, thematic analysis following Braun and Clarke (2012, pp. 57–58) is applied as the method for identifying and organizing recurring patterns across the modeling assumptions, architectural choices, and consistency claims reported in the included studies. Rather than allowing categories to emerge inductively, a deductive coding approach is used in which the six-dimension framework introduced in Section 2.5 serves as the predefined coding scheme. This choice is justified by the derivation of the framework: because the six dimensions were constructed from the foundational literature of Chapter 2 independently of the studies retrieved through the systematic search, the framework functions as an external analytical instrument that can be applied to those studies without circularity. (Braun & Clarke, 2012, pp. 66–68). The operational details of the coding procedure, including the unit of analysis, the normalization of codes into assumption patterns, and the traceability mechanisms used, are presented in Chapter 4. Accordingly, the literature review functions as the primary basis for answering RQ1, while also providing the conceptual evidence base for RQ2 and RQ3.

3.2 Empirical simulation study

The empirical part of this thesis is positioned as a comparative simulation study. Following Wohlin and Aurum (2015), it is useful to distinguish between the broader purpose of empirical inquiry and the specific method used to generate evidence. In their framework, evaluation research denotes research conducted to assess the effects, implications, or suitability of a method, tool, framework, or technology, whereas simulation is a method in which a model of a real-world entity is used to explore alternative implementations and what-if questions under explicitly defined conditions (Wohlin & Aurum, 2015, pp. 1435–1442). This distinction is important because simulation is not itself a research purpose, but a methodological means for producing comparative evidence in support of evaluative inquiry. In this thesis, simulation is used to compare how alternative architecture patterns preserve analytically visible state under the same evolving source conditions and the same analytical requirements.

This methodological choice follows from the nature of the research questions. The empirical objective is not to document one naturally occurring organizational case, but to compare several architecture realizations under shared conditions so that observed differences can be attributed primarily to the design assumptions under study. In naturally occurring settings, such comparison would be difficult to achieve because source systems, infrastructures, operational constraints, and reporting practices vary across cases, making it harder to determine whether observed differences arise from the architecture or from its surrounding environment. Simulation is therefore methodologically appropriate here because it enables multiple alternatives to be exposed to the same source history and the same analytical contracts in a controlled and repeatable form (Wohlin & Aurum, 2015, pp. 1430–1437, 1441–1442). In software engineering research, simulation is particularly useful when the objective is to study the consequences of alternative design choices under conditions where relevant assumptions, inputs, and observation logic can be controlled (Müller & Pfahl, 2008, pp. 117–121).

A central methodological distinction in simulation research is the distinction between the conceptual model and the computerized model. Following Sargent (2013, pp. 12–15), the conceptual model is the mathematical, logical, or otherwise formalized representation of the problem entity developed for a particular purpose, whereas the computerized model is the executable implementation of that conceptual model. The conceptual model therefore defines

what aspects of the problem are represented, what is intentionally abstracted away, which outputs are of interest, and what assumptions delimit the domain of applicability of the study. In this thesis, the conceptual model is centered on the temporal and semantic behavior of alternative analytics architectures under a shared evolving source history. The model is intentionally reductionist: it abstracts away from the full organizational and infrastructural complexity of enterprise analytics environments to preserve the features necessary to answer the research questions. This principle of purposeful abstraction is important because a useful simulation model is not one that reproduces the world in maximal detail, but one that is as simple as possible while still preserving the mechanisms necessary to answer the motivating research question (Sargent, 2013, pp. 12–14).

The simulation used in this thesis should therefore be understood as purpose-bound rather than universally representative. Its adequacy is evaluated relative to the analytical objective for which it is designed: controlled comparison of architecture behavior under shared source conditions and shared business-oriented requirements. The study does not attempt to reproduce the full performance envelope of production-scale enterprise systems, nor does it claim predictive validity with respect to infrastructure cost, user behavior, or organizational outcomes. The rigor of such a study depends not only on whether the implementation runs correctly, but also on whether the model is credible for its intended purpose. For this reason, simulation methodology places particular emphasis on verification and validation. Verification concerns whether the computerized model has been implemented correctly with respect to the intended conceptual design, whereas validation concerns whether the model is sufficiently accurate for its intended purpose (Sargent, 2013, pp. 12–14). Validation is therefore not absolute but contextual: a model may be adequate for one analytical purpose and inadequate for another.

Accordingly, simulation is used in this thesis as a methodological instrument for comparative evaluation under controlled conditions. Its role is to provide a structured means of examining the consequences of alternative design assumptions while holding the analytical setting sufficiently stable for meaningful comparison. More concretely, the simulation is used to compare architecture implementations representing the assumption patterns identified in the literature review against a common evolving source history and a common set of business-oriented analytical requirements. The concrete operationalization of this design, including the simulation environment, the architecture implementations, the scenarios, and the applied verification and validation procedures, is presented in Chapter 5. The simulation therefore

contributes most directly to RQ2 and RQ3 by examining how modeling challenges emerge empirically when batch-oriented reconstruction is weakened and how different architecture patterns can be adapted to preserve acceptable analytical contracts under those conditions.

4 Literature review

This chapter presents a systematic literature review (SLR) conducted to examine how real-time and continuous data processing constraints affect analytics-layer data modeling. The review addresses all three research questions by identifying the modeling assumptions embedded in traditional batch-oriented analytics (RQ1), analysing how these assumptions are challenged in real-time environments (RQ2), and synthesizing the modeling strategies proposed in the literature to address these challenges (RQ3).

Because the research questions concern conceptual modeling assumptions and semantic design choices rather than quantifiable effects, a structured qualitative synthesis was adopted. The review follows the PRISMA 2020 guidelines to ensure transparency and reproducibility in study selection and reporting, while employing systematic coding and thematic analysis to derive analytically meaningful patterns from the included studies. The outcome of this chapter is therefore not merely a descriptive overview of prior work, but an evidence-based conceptualization of how analytics-layer modeling evolves when batch-based reconstruction of analytical history is reduced or removed.

4.1 Search strategy

The objective of the literature search was to identify peer-reviewed publications that address analytics-layer data modeling in the presence of real-time, near-real-time, or continuous data processing constraints. An iterative search strategy was adopted, where pilot searches were used to refine terminology and ensure that the final query set captured the concept of interest without being overly dependent on a single vocabulary. A challenge identified during preliminary scanning was that the terminology surrounding real-time analytics is not fully standardized across academic domains. Concepts relevant to real-time analytics are frequently described using adjacent terms such as streaming analytics, real-time business intelligence, operational analytics, continuous analytics, and low-latency analytics. In addition, modeling-related contributions may appear under terms such as semantic layer, analytical schema design, or data warehouse modeling without using the phrase "data modeling” directly. For this reason, the search strategy followed a concept-driven rather than term-driven approach, using multiple synonymous and adjacent expressions to operationalize each core concept of the research question across information systems, data management, and applied analytics literature. This reduced the risk of excluding relevant studies due to disciplinary or temporal

differences in terminology and supported a more comprehensive retrieval of conceptually relevant publications.

The search strategy was structured around three conceptual blocks derived from the research objective. The resulting conceptual structure of the search strategy is summarized in Table 2.

Table 2. Identified concepts during search strategy assessment

Concept	**Focus**
Real-Time Analytics	Real-time, streaming, incremental
Data Modeling	Dimensional, semantic, OLAP models
Analytics Applications	Business Analytics, Business Intelligence, Analytics Layer

The research topic can be placed at the intersection of two core clusters: Real-Time Analytics and Analytics Modeling. In addition, the topic can be framed within the qualifying context of Analytics Applications. The Analytics Applications -concept block was used as a contextual qualifier, ensuring that retrieved studies addressed analytics modeling in a decision-support or analytical consumption context, rather than purely transactional, operational, or algorithmic settings. Based on these concepts, the search terms were selected to include relevant terminology that best corresponds with the research topic. This approach intentionally allowed the retrieval of publications where analytics modeling was embedded within architectural, system-level, or application-driven studies.

Within each concept, the chosen search terms were combined using the OR operator, while the concepts themselves were combined using the AND operator. To balance breadth and manageability, wildcard characters and truncation operators were utilized to capture spelling variations, pluralization, and related term forms (e.g. model, modeling, modelling).

The exact queries used for each database are provided in Appendix B. The databases used include Scopus, Web of Science, IEEE Xplore, ACM Digital Library and AIS eLibrary. These were selected due to their extensive coverage, accessibility and academic credibility. Searches were applied to titles, abstracts, and keywords where supported to increase precision and relevance. Where such a scope was not supported, the closest alternative was selected. A notable exception to this rule was ACM Digital Library, where the search was applied to any available field.

To ensure academic quality and comparability, the search was restricted to peer-reviewed journal articles and conference proceedings, and to publications written in English. No strict publication-year cutoff was applied at the search stage, as foundational work on analytics modeling and data warehousing is relevant for framing how real-time requirements may challenge established principles. However, in screening, priority was given to studies in which real-time analytics acted as the principal theme rather than a peripheral one.

The final searches were executed on 6.1.2026. Retrieved records were exported to Zotero reference management system for duplicate removal and preliminary filtering based on formal eligibility constraints. Prior to title and abstract screening, a set of records was removed for formal eligibility reasons unrelated to topical relevance. These removals were performed to ensure that all remaining records could be consistently and meaningfully evaluated in subsequent screening stages. First, records with insufficient bibliographic metadata were excluded. These included items lacking a title, abstract, publication venue, or publication year, as such records could not be reliably assessed against the inclusion criteria. Second, publications for which the full text was not available in English were excluded. Third, non-research publications were removed. This category included editorials, prefaces, workshop summaries, conference overview reports, and similar items that did not present an original research contribution. Workshop papers that reported empirical, architectural, or methodological research were retained. Records removed under these criteria are reported in the PRISMA flow diagram as records removed for other reasons. The resulting screening and selection process is summarized in the PRISMA flow diagram in Figure 2.

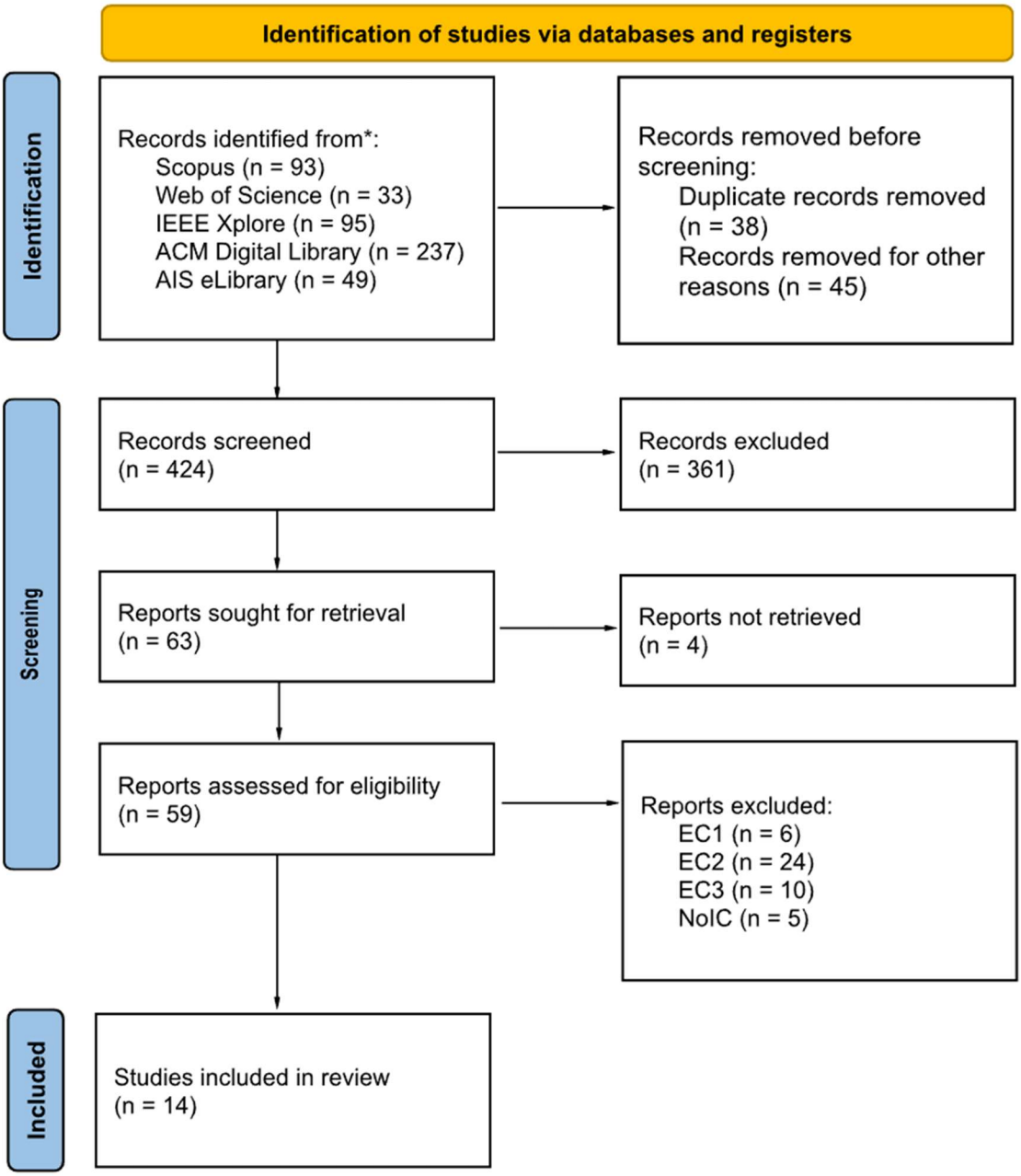

Figure 2. PRISMA flow diagram

4.2 Inclusion and exclusion criteria

Following the initial filtering of records based on publication type, language, and metadata completeness, the remaining studies were screened using a set of inclusion and exclusion criteria aligned with the research objective. The criteria were designed to operationalize the three conceptual blocks identified in the search strategy and presented earlier in Table 2. Specifically, each study was assessed based on whether it was situated within a business analytics, business intelligence, or enterprise analytics context, contributed to analytics-layer data modeling or schema design, and addressed real-time, near-real-time, streaming, or low-latency analytical processing. The inclusion and exclusion criteria are presented in Table 3. A study was included only if it satisfied all inclusion criteria and did not meet any exclusion criteria.

Table 3. Inclusion and Exclusion criteria used in screening

Inclusion Criteria (IC)		**Exclusion Criteria (EC)**	
IC1	The study is situated in a business analytics, business intelligence, or enterprise analytics context	EC1	The study is domain-specific without transferable business analytics modeling contributions
IC2	The study addresses analytics-layer data modeling, schema design, or semantic modeling for analytical systems	EC2	The study focuses on data infrastructure without relevance to analytics-layer data modeling
IC3	The study considers real-time, near-real-time, streaming, or low-latency analytical processing	EC3	The study addresses real-time systems or streaming architectures without an analytics or BI-oriented use case

Domain-specific studies were excluded only when their modeling contributions were tightly coupled to a specific application domain and could not be generalized beyond that context. Similarly, studies focusing on operational systems, machine learning models, or streaming architectures without a clear analytical or business intelligence use case were excluded. This screening approach ensured that the final study cohort focused on transferable insights into how analytics-layer data modeling is designed or adapted in real-time analytical environments.

Screening was conducted in three stages in accordance with PRISMA 2020. First, titles and abstracts were screened to remove records that were clearly irrelevant to analytics-layer data modeling in real-time analytical contexts. Second, full texts were retrieved for all remaining records where available. Finally, full-text eligibility assessment was performed using the

inclusion and exclusion criteria defined in Table 3, with exclusion reasons recorded at this stage. During title and abstract screening, the screening was performed conservatively by the author and studies were retained for full-text review when relevance was uncertain.

Full texts were retrieved using DOI resolution and institutional access, supplemented by searches for author-deposited versions via Google Scholar and institutional repositories. Despite these efforts, four studies could not be retrieved in full due to publisher paywall restrictions and the absence of accessible author versions. These studies were therefore excluded at the eligibility stage and are reported as “not retrieved” in the PRISMA flow diagram.

Full-text screening focused on assessing whether studies satisfied all three inclusion criteria simultaneously. Studies were excluded where contributions were limited to domain-specific applications without transferable analytics-layer data modeling implications, or where real-time or streaming architectures were presented without a business intelligence or analytical consumption context. Architectural and framework-oriented studies were included when they provided explicit or implicit contributions to analytical schema design, semantic layers, or multidimensional modeling in real-time analytical settings.

As shown in Figure 2, the screening and eligibility assessment resulted in a final cohort of fourteen studies, which were subsequently subjected to systematic coding and thematic analysis. Section 4.3 presents the included studies and introduces the analytical framework used to examine how analytics-layer modeling assumptions are articulated and how these assumptions are affected when data processing shifts from batch-oriented to real-time and continuous modes.

4.3 Included studies and thematic analysis

The included studies span the period from 2013 to 2025 and represent a broad set of real-time analytics perspectives, including batch-oriented warehousing extensions, streaming analytics, HTAP-oriented designs, and cloud-native real-time warehouse architectures. Some studies explicitly discuss analytics-layer schema and semantic modeling, whereas others present architectural or engine-level contributions whose modeling implications must be inferred from how analytical state, correctness, and consumption are described. This section presents the included study cohort and the analytical procedure used to compare their modeling

stances. The thematic results derived from this coding are reported in Section 4.4, and Section 4.5 synthesizes these results to answer the research questions.

Given the heterogeneity of the included studies, a qualitative synthesis was more appropriate than meta-analysis. Instead, the review adopted a qualitative synthesis based on systematic coding and thematic analysis. Following Braun & Clarke (2012), thematic analysis was used as a systematic method for identifying and organising recurring patterns of meaning across the included studies. In this context, the data used consisted of the analytical descriptions, data models, and architectural choices reported in each paper.

A deductive coding approach was applied using the author-derived six-dimension framework introduced in Section 2.5 as the coding scheme. This approach is appropriate because the framework was developed independently of the included studies, grounded instead in the foundational analytics modeling and data warehousing literature reviewed in Chapter 2. The framework therefore functions as an external analytical instrument brought to the literature rather than one derived from it, which is the standard condition under which deductive coding is methodologically justified. Coding was guided by the research questions concerning (RQ1) the temporal and semantic assumptions embedded in analytics-layer models, (RQ2) the challenges that arise when batch-based reconstruction is reduced or removed, and (RQ3) the strategies proposed to preserve interpretability and correctness under continuous processing. The unit of analysis was each study's primary analytical stance, as evidenced by its stated or implied analytical truth, closure mechanism, history mutability, semantic stabilization, analytical abstraction, and consistency. The coded evidence was drawn from the analytical model descriptions, architectural representations, and explicit consistency claims present in the studies, rather than from named technologies as such.

To enable cross-study comparison despite substantial variance in terminology, the extracted codes were normalized into a small set of recurring assumption patterns. Patterns were formed by grouping studies that shared the same dominant positions across the six coding dimensions and then verifying that the remaining dimensions formed a coherent pattern rather than an accidental match. These patterns are summarized in Table 4, which serves as a compact typology of how temporal closure and semantic consistency are operationalized in real-time analytics designs.

Table 4. Assumption patterns observed in the included studies

Pattern	Analytical truth	Temporal closure	History mutability	Semantic stabilization	Analytical abstraction	Semantic consistency
A. Closed snapshot warehouse	Snapshot-correct	Batch / micro-batch	Re-computable	ETL / preprocessing	Star schema	Deterministic and repeatable
B. Open evolving stream	Provisional	None / continuous	Stateful updates	Streaming engine / source	Event / log tables	Bounded inconsistency
C. Window-bounded stream	Window-correct	Event-time windows	Append-only	Engine window semantics	Streaming OLAP / star	Bounded inconsistency
D. Log-consistent HTAP	Transaction correct	Commit snapshot	Log-revised	Transaction logs	Relational + analytics	Approximate / contract-based
E. Virtual semantic snapshot	Snapshot-correct	Logical snapshot	Re-computable	Semantic layer	Semantic model + star	Deterministic and repeatable
F. Comparative / survey	Multiple	Multiple	Multiple	Multiple	Multiple	Multiple

As shown in Table 4, the typology makes explicit how different approaches operationalize analytical correctness and stability: from batch-closed snapshot semantics (Pattern A), through continuously evolving stream semantics (Pattern B), to window-bounded event-time correctness (Pattern C). It also captures approaches that preserve snapshot-style interpretability through alternative closure mechanisms, such as log-consistent transactional snapshots (Pattern D) and semantic layer -driven logical snapshots (Pattern E). Pattern F is used for studies whose contribution is primarily comparative or survey-based and therefore spans multiple assumption positions. Pattern F papers primarily delimit the design space and vocabulary, and they are used as interpretive rather than as evidence of a specific assumption pattern. As with most qualitative syntheses, pattern assignment reflects interpretive judgment. However, traceability is maintained through study-to-pattern mapping in Table 5, and the full coding matrix in Appendix C.

Table 5. Included studies grouped by assumption pattern

Pattern	Study ID	n
A. Closed snapshot warehouse	1, 2, 10	3
B. Open evolving stream	3, 5, 14	3
C. Window-bounded stream	6, 7, 12	3
D. Log-consistent HTAP	13, 4	2
E. Virtual semantic snapshot	11	1
F. Comparative / survey	8, 9	2

The assumption patterns summarized above provide a compact view of how the included studies operationalize interpretability under reduced batch reconstruction. Building on this typology, the following synthesis examines how analytical correctness is defined and when analytical state is treated as complete enough for interpretation, before turning to how systems accommodate revision once late data and corrections are admitted. It then considers where semantic meaning is stabilized and through which analytical abstractions stable metrics are exposed and concludes by assessing the consistency guarantees that govern analytical consumption in continuously updated settings.

4.4 Results of the literature review

Rather than summarizing the included studies one by one, the analysis of the results is organized around the six coding dimensions and presented in paired form to reflect the three core questions of interpretation under reduced batch reconstruction:

1. How correctness and closure determine when results are interpretable
2. How revision is admitted and how historical outputs can change
3. Which analytical abstractions and consistency guarantees can be offered to downstream consumers

Throughout this section, interpretability is used to denote the conditions under which analytical outputs can be meaningfully consumed without ambiguity. An output is considered interpretable when the paradigm provides an answer to three questions: what counts as correct, when the state is complete enough to read as valid, and how exposed history may change. This definition is intentionally narrower than general usability, focusing specifically on the semantic and temporal premises required for analytical meaning.

4.4.1 Analytical truth and Temporal closure

Across the included studies, analytical truth is most frequently grounded in time-bounded completeness rather than in a single universal notion of real-time correctness. In the closed snapshot warehouse pattern (Pattern A), truth is defined as retrospective snapshot correctness: results are treated as correct relative to a processed load window, and closure is established through the completion of ETL jobs. Ferreira et al. (2013) explicitly frame near real-time as a matter of shortening the latency between operational updates and analytical availability while still operating through repeated ETL execution cycles, implying that interpretability is anchored in the successful completion of each load run (Ferreira et al., 2013, pp. 1–2). Liu and Iftikhar (2015) similarly approach timeliness through ETL throughput optimization, where the load cycle is reduced by implementing parallelization and partitioning. In their approach, closure still remains the boundary of a completed transformation step rather than an event-time notion of completeness. (Liu & Iftikhar, 2015, pp. 1–2).

Patterns B and C shift closure away from ETL completion and towards continuous or window-bounded closure. The lambda-style approach in RADStack positions stream correctness as provisional and explicitly contrasts it with the accurate views produced by batch computation. Closure for the fast path is therefore operational, while closure for truth is restored by the batch path when late events and corrections are incorporated (Yang et al., 2017, p. 2). This framing makes the analytical truth explicitly two-tiered: immediate results are usable but may contain inaccuracies, while final correctness is achieved when batch recomputation closes the window of uncertainty. In systems built around real-time OLAP storage engines, closure is often expressed as window-level sealing rather than as a full recomputation boundary. Pinot, for example, targets immutable append-only data and second-level freshness, and it merges a real-time stream with an offline global view that enables more optimal segments. This implies that interpretability stabilizes when a time slice is materialized into a segment that can be queried consistently (Im et al., 2018, p. 5). STRCUBE makes the windowed truth assumption even more explicit: it proposes star-schema OLAP over streaming facts via sliding windows, where results are meaningful with respect to the current window boundaries and their update cadence (Ramanathan & Venkatesh, 2024, pp. 2–4).

HTAP / transactional-analytics pattern (Pattern D) introduces a third analytical truth. Here the goal is correctness relative to transactional state in addition to freshness. Hermes frames real-time transactional analytics as obtaining analytical answers directly from the evolving

transactional environment and discusses the tension between freshness and isolation/overhead, which positions truth as log-consistent rather than as the latest event observed. (Milkai et al., 2025, p. 2). In other words, closure is governed by a transactional cut or a visibility snapshot, not by a warehouse load boundary nor by an event-time watermark.

The remaining patterns (E and F) emphasize that closure can also be externally delegated or architecturally plural. Logical integration work highlights near real-time access through federated/logical mechanisms, shifting the closure assumption toward whatever guarantees upstream sources can provide at query time (Silva et al., 2023, pp. 1–2). Comparative and typological papers similarly treat real-time as an umbrella for multiple closure regimes (batch windows, micro-batches, streaming windows, and hybrid architectures), implying that analytical truth must be evaluated relative to the chosen architecture rather than treated as a single binary property (Ben Aissa et al., 2020, pp. 2–4).

4.4.2 History mutability and Semantic stabilization

The studies diverge most sharply on whether analytical history is allowed to change after initial exposure, and where such change is operationally realized. In closed snapshot warehouses (Pattern A), history mutability is typically selectively reconstructive: partitions, incremental loads, and accelerated ETL techniques are used to update recent data without fully rebuilding the warehouse on every run, but the architecture still assumes that corrected or late-arriving data can be reloaded and recomputed within the relational store. Ferreira et al. (2013) highlight partitioning and operational strategies as key enablers for frequent loading while maintaining queryability, reflecting a model where mutable history is managed through controlled reload/rebuild tactics rather than by immutable logs (Ferreira et al., 2013, pp. 3–7). Liu and Iftikhar (2015) similarly treat recomputation cost as an ETL engineering problem: history can be rewritten, but the practicality depends on how efficiently the ETL can be parallelized and partitioned (Liu & Iftikhar, 2015, pp. 2–4).

In the open evolving stream pattern (Pattern B), mutability is explicitly expected but is staged or continuously maintained rather than eliminated. RADStack notes that streams can contain duplicates and late events that require correction, and that the fast path may therefore diverge from batch truth until the batch layer recomputes a corrected result and the serving layer merges the two (Yang et al., 2017, p. 2). In this regime, semantic stabilization is not inside any single processing engine but rather emerges only after reconciliation across layers. The comparative architecture study reinforces this point by treating lambda’s dual-path design as a

pragmatic response to correctness limitations in streaming-only architectures, where fault tolerance and reprocessing semantics materially affect whether results can be stabilized (Karpathiotakis et al., 2017, pp. 1–2). Zhang et al. (2025) extend this open and evolving stance in a warehouse-native direction by targeting relational, incremental-maintenance semantics for continuously maintained views, explicitly avoiding stream-engine windowing while still enabling continuous updates (Zhang et al., 2025, p. 8). They further support online DDL (Data Definition Language) patterns that maintain accessibility and consistency while view definitions evolve, indicating that semantic stabilization can also be operationalized inside the data warehouse engine via continuous view maintenance and managed schema evolution rather than via batch rebuilds (Zhang et al., 2025, p. 12).

Windowed streaming OLAP engines (Pattern C) often adopt append-only, replace-by-segment mutability: raw events are immutable, but previously materialized artifacts can be replaced through compaction or correction. Pinot illustrates this by treating segments as immutable while allowing segments to be replaced with newer versions for updates and corrections (Im et al., 2018, pp. 2–5). Druid (as described in RADStack) likewise emphasizes immutable segments and notes that real-time ingestion is optimized for append-heavy workloads and does not support data updates, implying that corrections must occur via additional segments or replacement rather than in-place mutation (Yang et al., 2017, p. 8). STRCUBE's window model similarly stabilizes meaning at the level of window state, where the maintained fact table corresponds to the current window boundaries and updates occur as window slides (Ramanathan & Venkatesh, 2024, pp. 3–4).

The HTAP and transactional analytics pattern (Pattern D) shifts history mutability from materialized facts to transactional visibility control. Hermes explicitly frames design choices around the trade-off between freshness and isolation by implying that history is not rewritten by backfills but rather becomes visible through controlled transactional snapshots and capture mechanisms (Milkai et al., 2025, p. 12). In this regime, semantic stabilization is anchored in the guarantees of the transactional layer: what is true is what is visible under the selected consistency/isolation configuration.

The virtual semantic snapshot pattern (Pattern E) represents a distinct stance: it preserves snapshot-style interpretability by relocating stabilization into a logical integration and semantic abstraction layer rather than into a physically closed warehouse refresh. Silva et al. (2023) describe a layered architecture in which a global schema provides a unified logical

view over distributed sources and users may define star schemas over that global abstraction, allowing analytical queries to be issued on fresh data through high-level business-oriented entities without requiring knowledge of data location, fragmentation, or underlying technical details (Silva et al., 2023, pp. 1–6). In this regime, history remains logically recomputable through the semantic and mapping layers, while semantic stabilization is achieved by the global and star-schema abstractions that present a coherent analytical view over the underlying sources. Thus, the analytical effect is closest not to an open evolving stream, but to a virtual semantic snapshot in which the user-facing analytical state is stabilized logically rather than through a batch-closed physical snapshot (Silva et al., 2023, pp. 2–6).

4.4.3 Analytical abstraction and Semantic consistency

The reviewed studies also differ in what they treat as the primary analytical abstraction (star schema, denormalized event table, segments, maintained views) and therefore what kinds of consistency guarantees are realistic for downstream BI consumption.

Unsurprisingly, closed snapshot warehouses (Pattern A) remain closest to the Kimball-style abstraction: the analytical model is fundamentally relational and dimensionally organized, and semantic consistency is achieved by querying a closed snapshot. In these studies, near real-time is primarily a latency reduction objective, which does not fundamentally alter the abstraction of facts and dimensions (Ferreira et al., 2013, pp. 1–2).

In contrast, many window-bounded streaming architectures (Pattern C) achieve low latency by changing the abstraction itself. RADStack's serving layer is described as accepting fully denormalized data and moving away from the relational model, with immutability and segment-based storage used to deliver low-latency exploratory OLAP queries (Yang et al., 2017, p. 2). Pinot likewise promotes a segment-based OLAP model with immutable data segments and a query language that resembles SQL but omits joins. The abstraction is therefore not a star schema queried relationally, but a set of pre-indexed segments optimized for slice-and-dice over a fixed schema (Im et al., 2018, p. 5). In these systems, semantic consistency is typically deterministic within a segment/snapshot, but system-level correctness is bounded by ingestion, late events, and segment replacement strategies. STRCUBE is notable because it explicitly attempts to preserve the star-schema abstraction under streaming conditions by making the fact table itself a streaming entity while retaining dimensional joins and OLAP operations (Ramanathan & Venkatesh, 2024, p. 2). However, its reliance on windowing implies that consistency is inherently window-relative: results are correct with

respect to the maintained window state, not necessarily with respect to an eventually complete global history (Ramanathan & Venkatesh, 2024, pp. 3–4).

HTAP / transactional analytics work (Pattern D) redefines the abstraction again: analytical queries are evaluated against a system that must serve both transactional and analytical workloads. Hermes foregrounds that the strongest consistency guarantee competes with isolation and overhead, implying that guarantees for analytical consumption are choices along a spectrum rather than a fixed property (Milkai et al., 2025, p. 12). In such settings, semantic consistency is best characterized as visibility-consistent (deterministic given a chosen snapshot/isolation rule), rather than as the classical deterministic within a processed snapshot of a warehouse load.

Within the open evolving stream pattern (Pattern B), Streaming View offers a warehouse-native route to near real-time analytics by continuously maintaining relational views with incremental updates, including wide-table OLAP scenarios where a fact table joins multiple dimensions (Zhang et al., 2025, p. 10). Because the maintained artifact remains a SQL-defined view state within the warehouse, this approach preserves a more relational analytical abstraction than event- or log-table designs, while handling mutability through incremental maintenance and controlled recomputation strategies. Its semantic consistency therefore remains eventual or maintenance-dependent rather than equivalent to deterministic repeatability over a closed snapshot (Zhang et al., 2025, pp. 8–10).

The virtual semantic snapshot pattern (Pattern E) preserves BI-like interpretability more directly. Silva et al. (2023) describe a layered logical integration architecture in which a global schema provides a unified view over distributed sources and users may define star schemas over that global abstraction, allowing analytical queries to be issued on fresh data through high-level business-oriented entities without requiring knowledge of data location, fragmentation, or underlying technical details (Silva et al., 2023, pp. 1–2, 4–6). Because the primary analytical abstraction remains a semantic and star-schema layer defined above the integrated sources, semantic consistency remains closer to deterministic and repeatable analytical consumption than in the openly evolving designs, even though the stabilizing mechanism is logical rather than a physically closed warehouse snapshot (Silva et al., 2023, pp. 2, 4–6).

4.5 Synthesis

The review identifies several distinct design families rather than a single real-time replacement for batch-oriented analytics. These families differ in how they define correctness, establish closure, and support stable analytical consumption. The central implication is that once batch reconstruction is reduced, those design choices must be made explicitly rather than inherited from the architecture by default.

Traditional analytics-layer models have usually depended on batch processing to stabilize interpretation. In practice, ETL completion has often served as the point at which data is treated as coherent enough for analysis. The closed-snapshot warehouse pattern makes this especially visible: results are interpreted relative to a processed snapshot, and closure is tied to the completion of a batch or micro-batch cycle. As long as this closure holds, history remains recomputable and semantics can be stabilized through preprocessing and schema conventions, typically expressed through star-schema abstractions and yielding deterministic, repeatable outputs for any given processed state. The near-real-time warehousing contributions in the cohort largely preserve this assumption pattern and treat reduced latency as an engineering problem of faster refresh and more efficient partitioned maintenance, rather than as a redefinition of what counts as an interpretable analytical state. In this sense, batch-based processing supports analytics-layer assumptions not only by moving data, but by defining when interpretation is warranted and by providing a practical pathway to restate history until the snapshot is considered correct.

Once closure is no longer guaranteed by batch-job completion, the challenges arise where batch semantics previously provided free interpretability. When closure is continuous or window-bounded, analytical results become exposed under conditions where not all relevant events have arrived, dimensional context may lag behind facts, and late or out-of-order events can change previously observed outputs. This shifts the central risk from query performance to semantic interpretation: consumers may receive numbers that are meaningful only under a provisional analytical truth, or under an explicitly bounded window of correctness, even when the surface-level analytical interface resembles familiar BI interaction. Correspondingly, history mutability becomes a first-class modeling problem rather than an ETL afterthought. Some approaches admit mutability through staged reconciliation (where an initially exposed view is later overwritten or corrected), while others avoid in-place mutation by adopting immutable event histories and replacing derived artifacts to incorporate corrections. Across

these positions, the review shows that correctness in real-time settings is frequently not an absolute property but a relationship between a closure mechanism and a revision policy that determines how and when earlier outputs may be revised. This reframes the challenge: without batch reconstruction, the analytics layer must make explicit what is being promised and how those promises interact.

A further consequence of reduced batch reconstruction is the pressure it places on where semantics are stabilized and what the primary abstraction becomes. Several streaming-first designs achieve low latency by moving away from relational join-centric star schemas toward denormalized event/log tables or segment-oriented OLAP stores, implicitly relocating semantic stabilization from ETL/schema discipline to engine-internal maintenance rules. Where star-like abstractions are retained under streaming, they tend to be reinterpreted through window semantics: the fact table becomes a continuously updated, window-relative state rather than a stable historical table, and the meaning of aggregates becomes tied to the window definition instead of a closed period of loaded history. In HTAP-oriented work, the challenge is sharpened further: analytical interpretation must align with transactional visibility rules, making isolation and overhead part of the semantic trade-off space. Across these designs, semantic consistency shifts from deterministic repeatability over a processed snapshot toward bounded inconsistency, eventual convergence, or consistency guarantees defined by the chosen architecture. This indicates that the key modeling difficulty is not simply handling higher update rates but maintaining a coherent interpretive contract when the analytical state is open, mutable, or visibility-scoped.

Rather than converging on a single real-time dimensional model, the reviewed literature describes several strategies that preserve interpretability by choosing different substitutes for batch closure. One strategy accelerates the classical warehouse approach while preserving snapshot semantics: partitioning, parallelization, and micro-batch refresh retain recomputability and deterministic consumption at the cost of still relying on periodic closure. This approach is primarily useful when reducing temporal latency but not completely eliminating it. Taking one step closer to instantaneous delivery, a hybrid approach adopts dual views of truth exposing low-latency results as provisional while restoring correctness through later recomputation and reconciliation. In these environments, interpretability is achieved by explicitly separating fast and accurate paths, and by treating revision as expected rather than exceptional. A third strategy makes correctness explicitly window-bounded, using event-time windows and sealing mechanisms so that stability emerges at the level of finalized windows

or segments rather than at the level of a fully reconciled global history. In a similar, streaming-native manner, transactional correctness can be sought by anchoring analytics to commit snapshots or log-consistent visibility, thereby redefining closure in transactional terms instead of ETL or watermark terms. The review also includes warehouse-native approaches that attempt to preserve BI-like consumption semantics under continuous change by continuously maintaining relational views and managing schema evolution online. Rather than shifting semantics into a separate streaming engine, this route stabilizes meaning inside the warehouse through incremental maintenance and controlled recomputation, conceptually approximating a logical snapshot.

The main contribution of this review is therefore not the identification of a single preferred architecture, but a clearer view of the design space. The six-dimension framework proposed in this thesis makes those alternatives comparable by rendering their assumptions about truth, closure, mutability, stabilization, abstraction, and consistency explicit. The literature suggests that interpretability can still be preserved under real-time conditions, but only when those choices remain internally coherent. Taken together, the literature review shows that batch-oriented analytics has historically stabilized a particular combination of truth, closure, revision, and consistency assumptions (RQ1), that weakening this stabilization turns those assumptions into explicit modeling challenges (RQ2), and that several distinct architecture patterns have been proposed to reconstruct interpretability under continuous conditions (RQ3). This conclusion provides the conceptual basis for the empirical chapter that follows, where these assumption patterns are examined in a controlled comparative setting.

5 Empirical simulation study

This chapter applies the simulation method introduced in Section 3.2 to the empirical problem of the thesis through an executable simulation environment and a set of experimental conditions derived from the literature review. Where Chapter 4 identified the relevant assumption patterns conceptually, this chapter examines them empirically in relation to RQ2 and RQ3 by testing how the challenges associated with reduced batch reconstruction become visible in architecture behaviour and how different patterns can be configured to satisfy shared analytical requirements. The empirical study focuses on the temporal and semantic consequences of architecture choice under a controlled minimal analytical model, rather than on the full richness of enterprise analytics data modeling.

The technical implementation of the simulation, including source code, parameter definitions, generated artefacts, and reproducibility instructions, is documented in the accompanying GitHub repository (AnalyticsSimulation v1.2.0; https://github.com/HelmerHaapala/temporal-analytics). Accordingly, this chapter focuses on the study design, the evaluation conditions, and the interpretation of results, while more detailed implementation-level explanations are provided in the repository documentation.

5.1 Study design

Methodologically, the simulation was designed as a controlled comparative study in which multiple architecture implementations were exposed to the same evolving source history and evaluated through the same observation process. The unit of analysis is therefore not an organization or a single technical deployment, but an architecture implementation understood as a concrete realization of a literature-derived analytical pattern. This design follows the logic of comparative simulation in software engineering, where alternative design options are assessed under shared conditions to make behavioural differences attributable to the design choices under study rather than to uncontrolled environmental variation (Müller & Pfahl, 2008, pp. 117–121).

The study design includes both baseline and scenario-based evaluation. The baseline condition captures the native behaviour of each architecture without scenario-specific tuning and is used to observe the default temporal and semantic consequences of each implementation. The scenario-based evaluation then introduces explicit business-oriented requirements that function as analytical contracts. These scenarios allow the same underlying

architectures to be assessed not only in terms of their native behaviour, but also in terms of how they can be adapted to satisfy specific interpretability requirements. The resulting analysis therefore combines two perspectives: comparative behaviour without intervention and comparative feasibility under explicit analytical demands.

5.2 Simulation design

The comparative study was operationalized through four design components: a shared simulation environment, a common evolving source history, a set of alternative architecture implementations, and a unified observation logic through which their analytically visible states could be compared. Together, these components establish the controlled setting required for comparative simulation. By keeping the source history, analytical model, and observation process stable across implementations, the study attributes observed differences primarily to the temporal and semantic assumptions embedded in the architecture patterns. Because the source history is pre-generated and fixed for each run, repeated executions of the same configuration yield the same analytical outputs.

The following sections elaborate these components in more detail. Section 5.3 describes the simulation environment in which the comparison was conducted. Section 5.4 presents the architecture implementations included in the study. Section 5.5 defines the scenarios and target conditions used to express different analytical requirements. Together, these sections provide the concrete simulation design through which the methodological approach introduced in Section 3.2 was applied in the present thesis.

5.3 Simulation environment

Reflecting the modern analytics environment introduced earlier in Chapter 2, the simulation study was designed as a simplified three-layer analytics environment consisting of a database layer, a semantic layer, and a visualization layer. The structure of this environment is illustrated in Figure 3.

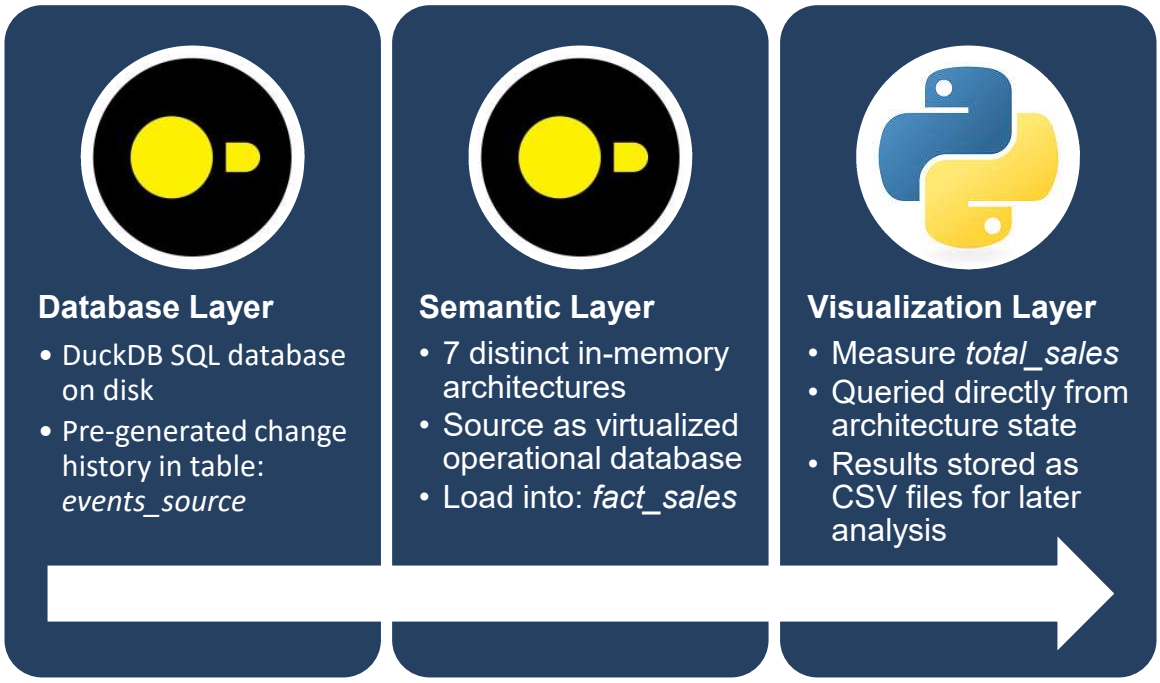

Figure 3. Simulation study environment

The database layer was implemented as an on-disk DuckDB database, while the semantic-layer architectures were implemented as separate in-memory DuckDB instances. The visualization layer was implemented minimally for the purposes of the study: rather than supporting end-user interaction, its role was to query the analytically visible state of each architecture and store the resulting outputs for later analysis.

In the database layer, the full change history was pre-generated and materialized in the table *events_source*, which served as the common source for all evaluated architectures. The source data was generated once for the full duration of the simulation run, effectively making the table a changelog of events observed in an operational database. This design allowed the architectures to be decoupled from one another while still progressing independently through the same shared source history. The database layer was intended to emulate a mutable operational database rather than a pre-reconciled analytical dataset or a native event stream. For that reason, the source had to be deduplicated before it could be queried as a current-state table. This choice reflects the practical setting that motivates the study: in many organizations, analytics is sourced from mutable operational systems rather than from purpose-built event-stream platforms.

In the semantic layer, each architecture loaded data from the same source database into its own analytical state. Shared source interpretation and deduplication logic were used wherever the research design required functional equivalence, so that differences in analytically visible

behaviour would reflect architecture-specific handling of closure, revision, and exposure rather than differences in source derivation. In the streaming-oriented cases, an additional intermediate transformation was required to reconstruct a changelog-like input from the operational source so that stream-compatible processing semantics could be emulated under the shared source condition.

In the visualization layer, the measure *total_sales* was queried directly from the current state of each architecture. The resulting outputs were then stored as CSV files for later analysis. This made it possible to compare the architectures using a shared observation process while preserving traceability between architectural state, analytical output, and downstream evaluation. In addition, the visualization layer was responsible for collecting the simulation metrics such as accuracy, freshness and rows loaded. These metrics are specific to the purposes of the simulation and should not be interpreted as features of a production-scale visualization layer.

5.4 Architecture implementations

The literature-derived assumption patterns presented in Section 4.3 were translated into five primary architecture implementations: the closed snapshot warehouse pattern (Architecture A), the open evolving stream pattern (Architecture B), the window-bounded stream pattern (Architecture C), the log-consistent HTAP pattern (Architecture D), and the virtual semantic snapshot pattern (Architecture E). In addition, a batch reference architecture and a ground-truth architecture were established. The batch reference was included as a conventional comparison point, while the ground-truth implementation served as an analytical reference rather than as a practical target. It applies each arriving change immediately and therefore provides the best available comparison point against which the other architectures can be evaluated.

The implementations were designed as controlled variants of the same analytical function. The purpose was not to build the strongest possible production implementation of each pattern, but to represent the temporal and semantic logic of each architecture under comparable conditions. For that reason, every architecture receives the same source history, supports the same analytical measure, and is evaluated through the same observation process. This keeps the comparison focused on what each architecture makes visible, when it makes it visible, and how it handles correction, closure, and historical revision over time.

To preserve comparability across architectures, every implementation uses the same deliberately minimal analytical model: one fact table with the same columns and the same core analytical measure in every architecture. No architecture-specific star schema, denormalized reporting layer, or richer semantic model was introduced. This simplicity was intentional. A more elaborate model would have introduced additional design degrees of freedom and made it less clear whether observed differences arose from the architecture pattern itself or from surrounding data-model choices. The one-fact-table model therefore functions as a control condition that keeps the analytical structure constant while allowing differences in visible behaviour and cumulative loading to be attributed more defensibly to how each architecture handles change and closure.

The shared analytical measure is *total_sales*, implemented in every architecture as the sum of $amount * quantity$ over the currently visible non-deleted rows. The measure was chosen because it is additive, immediately interpretable, and directly affected by the late arrivals, updates, and deletions present in the source history. A more complex derived measure would have introduced additional semantic assumptions that were not part of the architectural comparison itself. Each implementation was further designed to expose its outputs through the same measurement interface. The comparison therefore does not rely on architecture-specific reporting logic or downstream models, but on the same observable question in every case: what total becomes visible at each observation point, and how many cumulative rows must be loaded to sustain that visible result.

5.5 Scenario design and targets

B0 serves as the baseline reference scenario, in which all architectures are evaluated under one fixed set of default settings that are not optimized for any individual business requirement. Those baseline settings are anchored to the generated source time span rather than tuned per scenario. Its purpose is to provide a shared comparison frame and to make visible the trade-offs that arise from the architecture patterns themselves under comparable assumptions. S1-S5 are the business scenarios, in which the parameters of each architecture are pre-selected specifically to fulfil the active business targets where a passing configuration exists. The business scenarios were designed to reflect recognizably different and realistic analytical situations. Some are closer to operational monitoring, where immediacy matters most, while others resemble conventional reporting settings, where stability and closure are the primary relevant factors. In practice, this means that some scenarios place more weight on

immediate visibility and provisional truth, while others place more weight on delayed closure, limited history mutability, and stable semantic interpretation. B0 therefore shows native architectural behaviour under fixed assumptions, whereas S1-S5 show how far each pattern can be adapted to meet a given business requirement.

The scenarios are expressed through a small set of acceptance conditions that define what counts as an acceptable analytical contract in each case. The threshold metrics used to express these requirements are summarized in Table 6. Not every metric is active in every scenario, but together they make the business requirements explicit before the architecture patterns are compared.

Table 6. Acceptance condition metrics

Metric	**Definition**
Freshness	Maximum acceptable lag between source change and the visible analytical state at the observation points used in the scenario
Live Point-in-time accuracy	Closeness of the visible result to ground truth at the relevant observation point or settled reporting window
Closed daily-window accuracy	Closeness of the latest closed 24-hour event window to ground truth after an explicit settling delay
Closed weekly-window accuracy	Closeness of the latest closed 7-day event window to ground truth at the weekly review point defined by the scenario
Same-horizon restatement ratio	Frequency of post-exposure value changes without any advance in the visible reporting horizon

S1 represents a live sales dashboard used for intraday operational monitoring. The scenario reflects a business setting in which commercial or operational teams need an almost immediate view of sales development during the day, for example when monitoring campaign response or sudden shifts in sales activity. In terms of the six-dimensional framework, S1 places the strongest emphasis on freshness and a weaker requirement on semantic consistency. It therefore tolerates a looser analytical truth than the later scenarios. The scenario requires freshness of at most 10 minutes and live point-in-time accuracy of at least 60%.

S2 represents an hourly sales dashboard. It still belongs to the low-latency end of the scenario set, but it assumes that users expect a meaningfully more dependable live view than in S1. It requires freshness of at most 1 hour and live point-in-time accuracy of at least 80%. The one-hour cadence still reflects an operational use case, but one in which the dashboard is expected to remain meaningfully closer to effective truth.

S3 represents a daily sales dashboard used for recurring operational review. The scenario corresponds to a reporting context in which managers need a dependable view of a recently closed reporting window rather than a near-live view of the current one. In framework terms, S3 gives more weight to temporal closure and less weight to immediate visibility. It also begins to narrow the acceptable room for later revision. The scenario requires freshness of at most 24 hours, and it requires the latest closed 24-hour event window to reach at least 99%

accuracy after 8 hours of settling time. The settling delay reflects the idea that a daily result may remain provisional for a short period after the formal boundary of the day has passed.

S4 represents a weekly management review. The scenario captures a business context in which a closed weekly view must become available within a bounded review cycle and should then remain effectively stable for interpretation and decision support. It requires freshness of at most 7 days, closed weekly-window accuracy of at least 99.9%, and a same-horizon restatement ratio of at most 0%. The 7-day freshness target was chosen to represent a realistic weekly review cadence, while the 99.9% weekly-accuracy target reflects the expectations placed on this type of management review report. The zero-restatement condition makes explicit that once the reviewed week is exposed as the accepted result, further revision within that same reporting horizon is no longer acceptable.

S5 combines the requirements of the previous scenarios into a single shared-model case. The purpose of this scenario is to test how the behaviour of the architectures change when one model is applied to fill multiple purposes at the same time. The other scenarios allow scenario-specific parameter adoption, a freedom that is removed in this scenario. It requires freshness of at most 10 minutes, live point-in-time accuracy of at least 80%, closed daily-window accuracy of at least 99% after 8 hours, closed weekly-window accuracy of at least 99.9%, and a same-horizon restatement ratio of at most 0%.

The scenarios move from the loosest live-consumption case (S1) to a stricter hourly dashboard (S2), then to daily closed-window reporting (S3), weekly highly stable review (S4), and finally to a scenario where all the previous targets are set as requirements at once (S5). The purpose was not to privilege one architecture family or one reporting cadence, but to test how the different architecture patterns behave when these requirements change.

5.6 Evaluation criteria

The scenario metrics should be understood as the conditions under which an architecture is considered acceptable for a given business use case. In the business scenarios S1-S5, the simulation sets direct freshness parameters to the scenario target where that mapping is explicit and searches only the remaining degrees of freedom where needed. For the architectures that still require search, the remaining candidate set is evaluated in slowest-first order. The slowest-first rule was used to favor the least aggressive configuration that still satisfied the analytical contract, thereby avoiding unnecessary over-tuning toward freshness.

As a result, freshness, live accuracy, closed-window accuracy, and same-horizon restatement ratio function primarily as acceptance conditions that define what counts as an acceptable analytical contract in each scenario. Once all architectures satisfy the acceptance conditions of a scenario, those metrics no longer provide the main basis of differentiation, because the study has already aligned the implementations to the target conditions by design.

Two complementary comparison layers are therefore used in the empirical analysis. The first is the untuned baseline scenario B0, which shows how each architecture behaves before any scenario-specific adaptation is introduced. In that baseline setting, freshness and accuracy remain informative as behavioural indicators because they reflect the native closure and revision logic of the architecture. The second layer concerns the tuned business scenarios, where the main remaining point of comparison is the cumulative number of source rows processed to sustain the visible analytical state. Because every architecture is evaluated against the same source history, cumulative rows loaded provides a system-independent measure of how much loading, reloading, or rematerialization is required to maintain the result. The processed row count is a proxy for loading burden, not a complete measure of total system efficiency. It does not directly capture query latency, storage cost, or operational complexity. A notable exception in the interpretation of row count is Architecture E, which materializes the analytical state only at query time. For this architecture, the number of rows loaded therefore reflects query-time computational cost rather than the cost of maintaining analytical state in advance. This treatment is still justified, because in Architecture E the cost of maintaining the analytical state is inseparable from the cost of presenting it. The same underlying principle also applies to the other architectures: their ability to present the analytical state depends on the maintenance work required to materialize that state beforehand.

This framing also clarifies how the empirical section relates back to the six-dimension framework introduced earlier in the thesis. The scenario metrics operationalize minimum acceptable requirements, especially along temporal closure, analytical truth, and history mutability. The comparative findings, however, emerge mainly from how different architecture patterns satisfy those requirements and how much cumulative loading that requires. Semantic stabilization and analytical abstraction are therefore interpreted less through any single threshold metric than through the overall behaviour and update logic of the architectures, together with the volume of loaded rows needed to sustain those behaviours.

5.7 Verification, validation, and scope of validity

Following Sargent (2013), the rigor of a simulation study depends not only on the results it produces, but also on the credibility of the model with respect to its intended purpose. In this study, verification and validation are therefore treated as purpose-bound methodological concerns rather than as claims of universal realism. The objective is not to demonstrate that the simulation reproduces the full complexity of enterprise analytics environments, but to establish that it is sufficiently credible for comparative evaluation of architecture behaviour under shared source conditions and shared analytical requirements.

Conceptual model validity was addressed by deriving the compared architecture families from the assumption patterns identified in the literature review and by limiting the simulation to those features necessary to answer the research questions. The conceptual model abstracts from broader organizational, infrastructural, and user-facing complexity and instead focuses on the temporal and semantic mechanisms through which architectures construct and maintain analytically visible state. This reduction is intentional: the study is designed to compare how different architectures handle change, closure, revision, and exposure under controlled conditions, not to reproduce complete production environments.

Computerized model verification was addressed by implementing all architectures within the same technical environment, by exposing them to the same pre-generated source history, and by evaluating them through the same observation logic. Shared elements required for comparability, including the source-history interpretation, current-state derivation, deduplication logic, and measure definition, were kept consistent across implementations wherever the research design required functional equivalence. The common analytical measure, *total_sales*, was defined identically for all architectures, and the visible analytical state of each implementation was queried through the same observation process. In this way, observed differences are attributable primarily to the architecture-specific handling of change, closure, and revision rather than to inconsistent implementation conditions. Verification also included checking that small hand-traceable source histories produced the expected visible state in each architecture before full scenario runs were executed.

Operational validity was addressed through explicit business-oriented acceptance conditions together with a ground-truth reference implementation. The ground-truth implementation applies each arriving change immediately and therefore provides the best available analytical reference within the simulation. The business scenarios define acceptable thresholds for

freshness, accuracy, closure, and same-horizon restatement, making the evaluative criteria explicit before the architecture patterns are compared. The simulation outputs are therefore validated relative to their intended purpose: not prediction of real-world system performance, but comparative assessment of how alternative architectures satisfy the same analytical contracts under the same evolving source history.

Data validity was addressed by fixing the source history in advance and using it consistently across all implementations. The source is synthetic and should therefore not be interpreted as a statistically representative model of any one operational environment. Its role is instead to provide a controlled change history containing the forms of variation required by the research questions, including inserts, updates, deletions, and temporally staggered arrivals. This makes the data suitable for comparative evaluation of architectural behaviour, while also delimiting the claims that can be made from the results.

The study therefore aims for comparative internal validity rather than predictive validity: given the same source history, analytical model, and evaluative conditions, what differences become visible in how architectures preserve interpretability when batch-oriented reconstruction weakens?

5.8 Results

The results are interpreted in two layers. The first layer concerns feasibility: whether an architecture can be parameterized to satisfy the active analytical contract. The second layer concerns behavior and maintenance burden: how the architecture produces the accepted output and how much cumulative loading that requires. Figure 4 should therefore be read as a feasibility screen and not as a ranking result. Under the current tuning, all architectures pass all five business scenarios. For the streaming-native patterns, comparable evaluation under a shared mutable source required an additional intermediary layer, indicating that stabilization may be relocated rather than eliminated. The batch reference, A, and E read directly from the operational source, whereas B, C, and D required this additional change-log style layer to emulate a stream-compatible input.

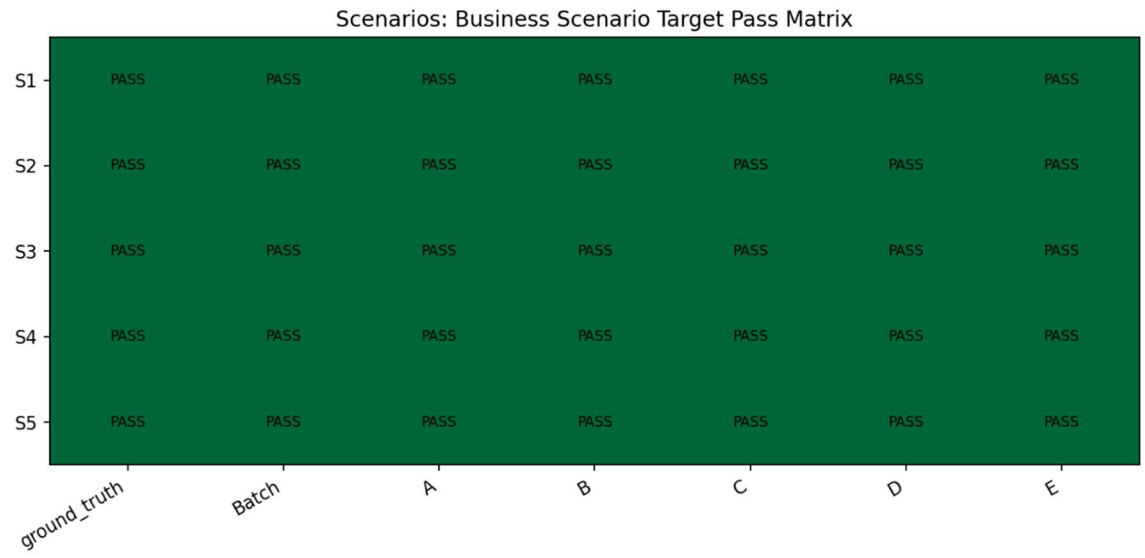

Figure 4. Business scenario target pass matrix

Because all architectures pass in S1-S5, the substantive result of the tuned runs is not who passes, but how differently the architectures behave while passing. The more informative evidence is therefore found in the sales trajectory comparison (Figures 5 and 14), in the truth-deviation views (Figures 6, 8, 10 and 12), and in the cumulative loading comparisons (Figures 7, 9, 11, 13, and 15). The scenario results in the following subsections are presented in two stages. First, the untuned baseline is used to examine the native behaviour of each architecture under the shared source history. Second, the business scenarios are used as explicit experimental conditions through which the architectures are evaluated against defined analytical requirements. This structure reflects the methodological distinction between observing the default semantics of each architecture and assessing its capacity to satisfy externally specified analytical contracts.

5.8.1 Baseline

The baseline reference condition should be read as a behavioural comparison under one shared, untuned parameter set rather than as a target-based evaluation. Its purpose is to show what kind of analytical contract each architecture offers before any scenario-specific adaptation is applied. The architectural parameters used for B0 were chosen by the author to best represent the behavioural characteristics of the architectures. The parameters used are accessible in the published repository in directory $/parameters/baseline\_params.json$. Figure 5 presents the visible sales trajectory of each architecture over arrival time, making it possible to compare how their native closure and revision logic shape the analytical state.

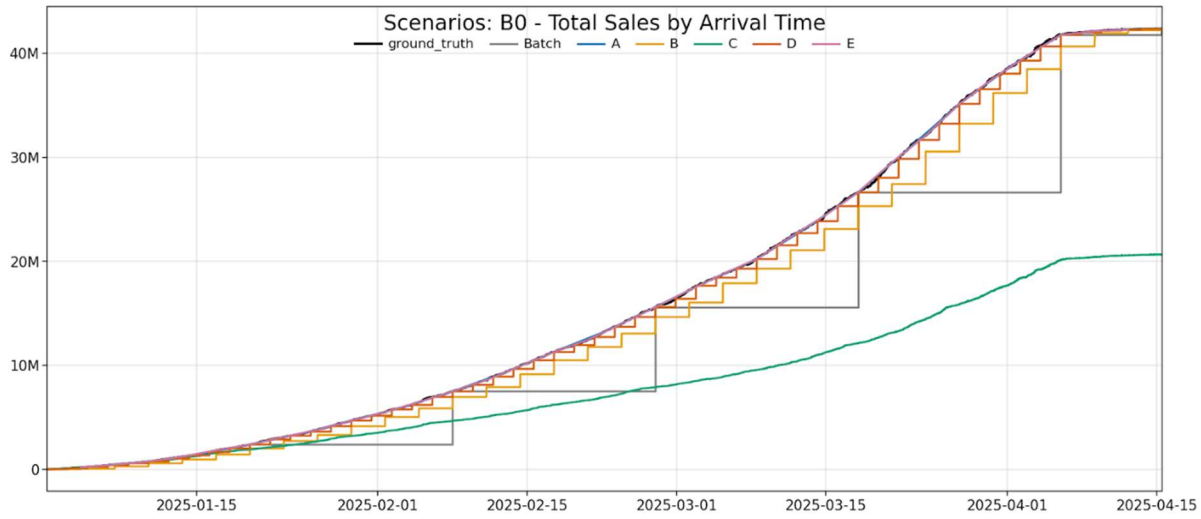

Figure 5. Baseline scenario total sales

Observing the visible sales trajectories reveals three distinct behavioural families. The batch reference exposes a coarse staircase with long closure intervals, reflecting classical snapshot semantics. At each refresh it returns to the reconciled cumulative state, but between those refreshes it visibly falls behind the ground-truth line. Architectures A, D, and E remain much closer to the overall ground-truth trajectory, but they do so through different closure mechanisms: A through incremental recomputation, D through transactional visibility, and E through semantic snapshot refresh. Architectures B and especially C expose more openly provisional states. B follows the same broad sales trajectory but presents it through a visibly stepped reconciliation pattern, while C diverges most strongly from the cumulative ground-truth curve because it maintains a window-bounded visible state instead of a fully accumulated historical total. In practice, this means that once rows fall outside the active window, they no longer contribute to the visible cumulative sales figure, even though they still belong to the longer historical truth. Even though architectures B and D appear similar to the batch architecture in terms of having a stepped refresh cadence, the semantics differ. These architectures do not promise absolute truth at refresh, unlike the batch reference. This is why architecture B can be always seen lagging behind ground truth: the input data is deliberately delayed at ingestion.

The baseline confirms that the architectures expose different analytical contracts before tuning: snapshot closure in the batch-oriented cases, provisional reconciliation in B, and window-relative visibility in C. The baseline operationalizes the findings of the SLR by showing that different temporal closure mechanisms already produce different analytical behaviours even before business targets are imposed.

5.8.2 Scenario 1 – Live Sales Dashboard

As the first business-oriented scenario, S1 represents the most demanding temporal requirement in this simulation through its 10-minute freshness target. Because all architectures satisfy the acceptance condition, the comparison is most informative not at the level of pass/fail, but at the level of update logic and maintenance burden. Figures 6 and 7 should therefore be read together. Figure 6 shows the visible deviation from ground truth over arrival time, and Figure 7 reveals the computational cost of meeting the targets.

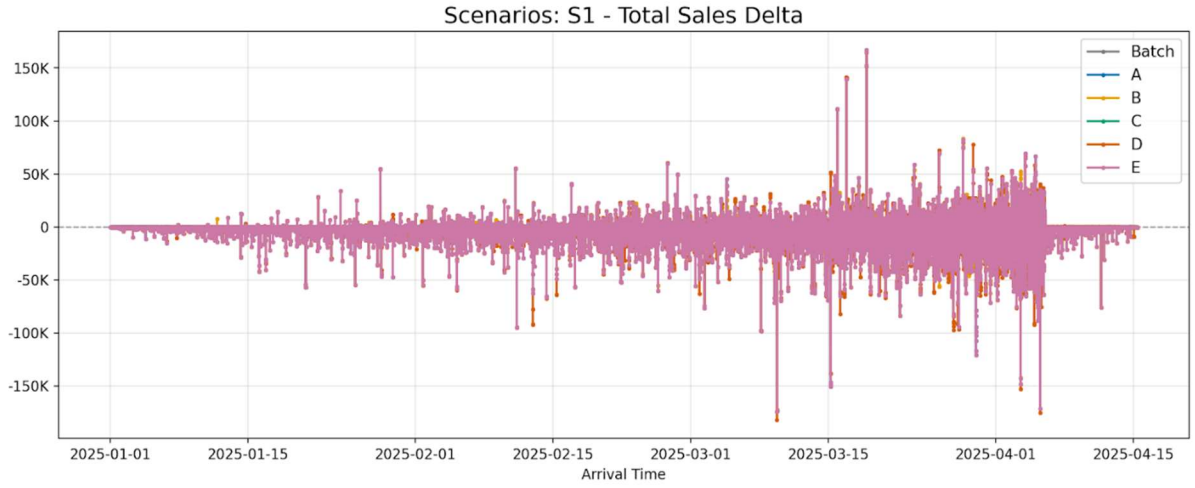

Figure 6. S1 Total sales delta

The simulation shows that architectures with different native analytical contracts can still be configured to satisfy similar contracts, but the cost of doing so remains architecture-specific. Some architectures stay near ground truth through punctuated closure and recurrent refresh, while others do so by keeping an already open and continuously updated state visible. The short-lived spikes visible in the batch-oriented cases and in the architectures that reconcile at discrete moments reflect updates, deletions, and late corrections arriving between refreshes. Because the deviations repeatedly return toward zero instead of drifting steadily in one direction, the figure shows recurrent reconciliation rather than uncontrolled divergence.

This is why Figure 6 should be read as a user-facing outcome view instead of proof that the architectures have become the same. For a dashboard consumer looking only at the visible totals, the architectures may seem broadly interchangeable under S1. Underneath that similar surface, however, the architectures still rely on different closure mechanisms and update logics to keep the visible result acceptable.

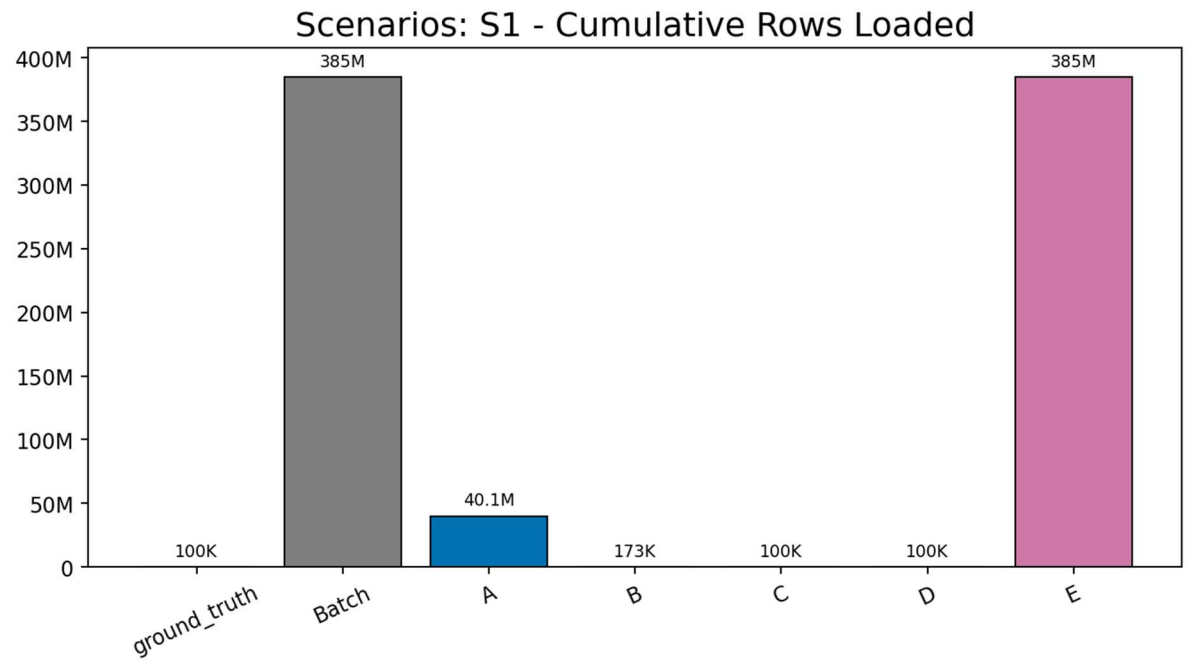

Figure 7. S1 Cumulative rows loaded

Figure 7 makes those hidden differences explicit. Under a 10-minute contract, the closed-snapshot architectures become dramatically heavier than the others because they must repeatedly recreate a sufficiently current analytical state from already seen history. The batch reference is the clearest expression of this burden, and A shows that even a much more adaptive incremental strategy remains structurally expensive when freshness becomes this aggressive. The issue is not simply inefficient implementation, but rather the snapshot-style correctness is being forced into a cadence for which repeated closure is intrinsically costly. Notably, even if the closure mechanism in architecture E is a virtual snapshot, it still bears a similar computational cost to a traditional snapshot closure mechanism. The computational cost however differs from the other architectures, as it is query load instead of preprocessing load.

The streaming architectures remain near source-row scale because they do not need to reconstruct a broadly closed global state every time the architecture state is refreshed. Their analytical contract is maintained through continuous exposure, where the rows are appended in the analytical architecture incrementally. S1 shows that the cost of low-latency analytics depends on what the architecture believes must be made current before a number is allowed to appear meaningful. When meaning depends on repeated closure, the burden explodes. When meaning can be maintained over an open state, the same business-facing contract becomes far cheaper to sustain.

5.8.3 Scenario 2 – Hourly Sales Dashboard

S2 relaxes the live requirement without abandoning it. This matters because it allows distinguishing between two different effects: the cost of being intra-day live in the first place, and the additional cost of being almost instantaneous. The scenario still concerns an open current-state view, but the one-hour target gives the architectures more room to stabilize their visible output before the next required update. Figures 8 and 9 show what that extra room changes, and what it does not.

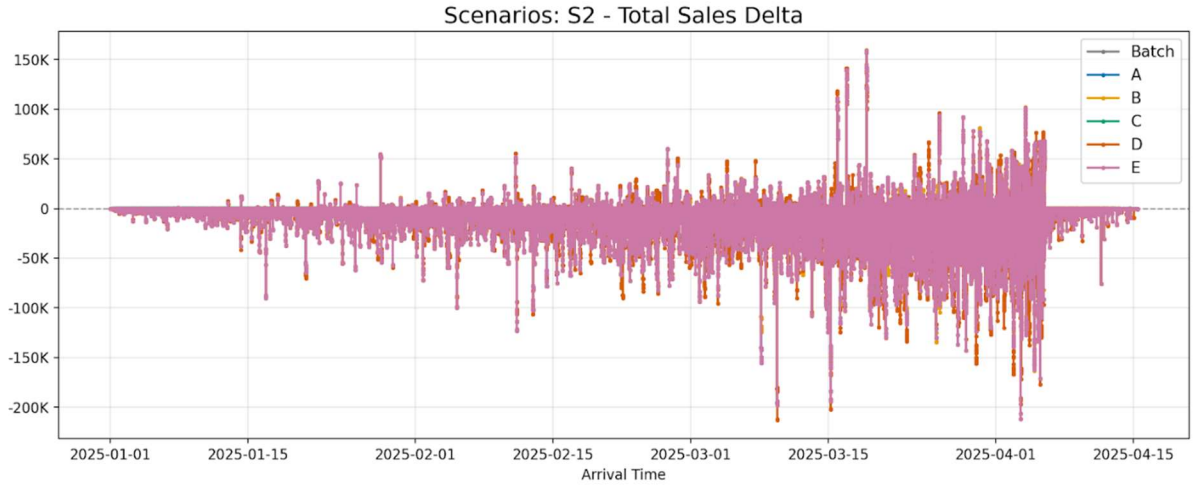

Figure 8. S2 Total sales delta

Relaxing the cadence to hourly, the architectures are allowed a longer timespan to deviate, but they still periodically converge with the ground truth. This is by design, as the parameter fitting essentially constrains the architectures to present the analytical state in a homogeneous manner. Effectively, the scenario stops being a test of exposed semantic difference and becomes more a test of how much computational load each architecture requires to keep that hourly view dependable.

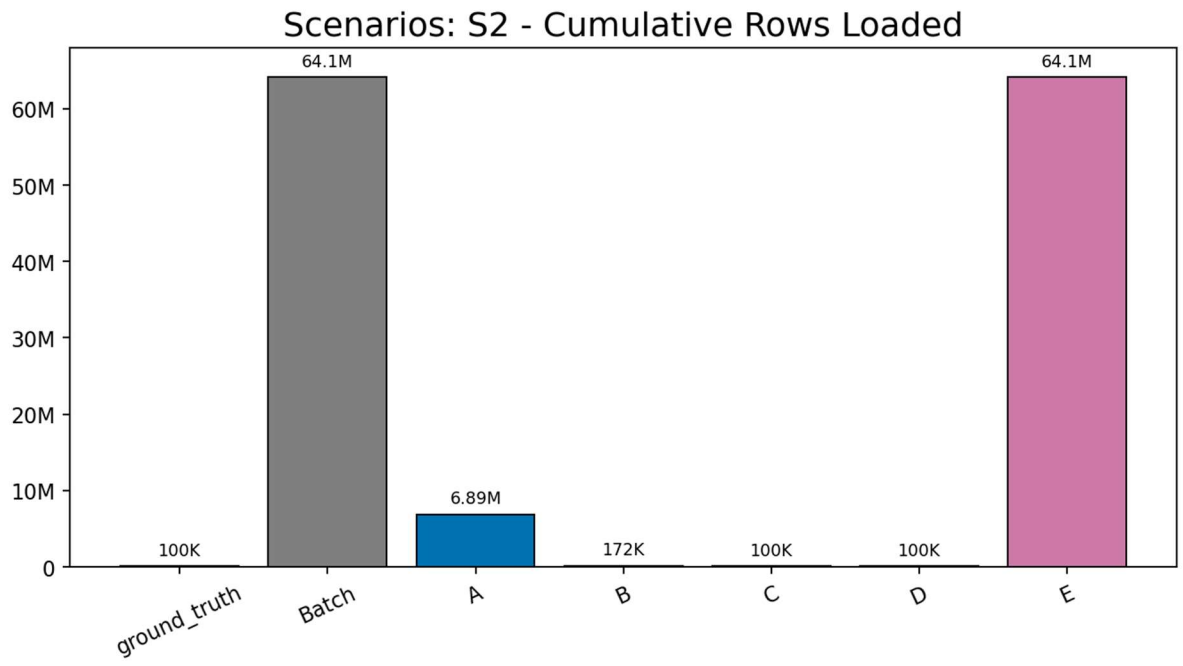

Figure 9. S2 Cumulative rows loaded

That asymmetry reappears immediately in Figure 9. The batch reference, A and E become much less expensive than in S1, which shows that the cost of freshness is highly linear for closure-oriented architectures. Reducing the refresh cadence by a factor of six effectively reduces the total count of rows processed in the Batch-architecture and Architecture A by a factor of six. Yet the same figure also shows that the fundamental ordering has not changed. The closed-snapshot approaches are still clearly heavier than the architectures that keep an open state visible.

The lighter architectures see little to no change, because their operating logic was already well suited to continuous visibility. The main lesson of S2 is that relaxing freshness can quickly narrow the semantic gap at the level of the visible result while leaving the underlying architectural cost structure largely intact.

5.8.4 Scenario 3 – Daily Sales Dashboard

S3 marks a more substantive change in the nature of the analytical contract. The object of interest is no longer the currently visible state, but a recently closed daily window that is allowed time to settle before it is interpreted. This shifts the scenario away from the semantics of continuous exposure and toward the semantics of bounded closure. The practical implications of this shift in both the objective and the cadence are visualized first in Figure 10 and then in the loading comparison of Figure 11.

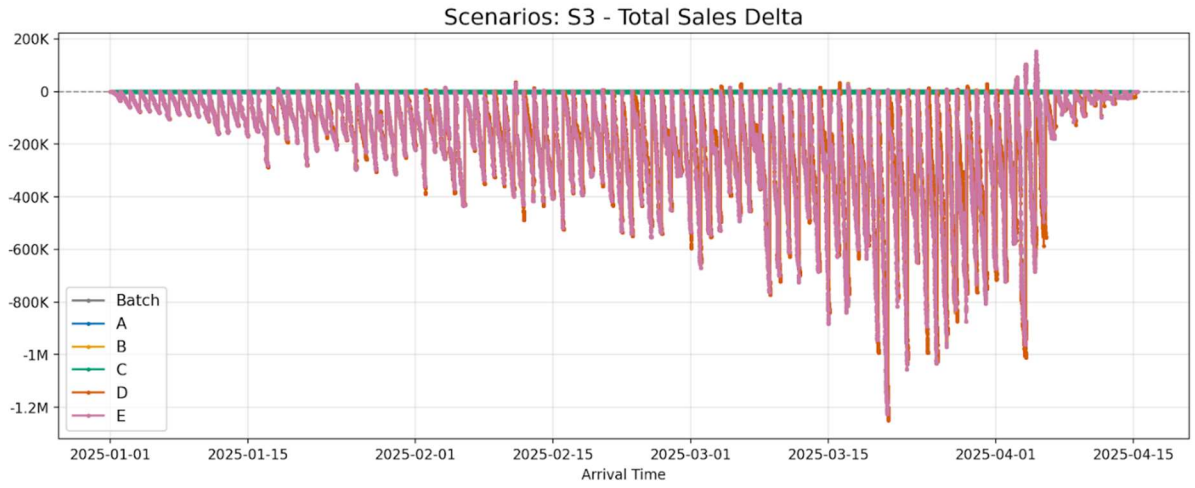

Figure 10. S3 Total sales delta

The total sales delta visualizes that the open portion of the daily window can still look materially different from the final settled daily result, even though the scenario itself is not judged on the still-open state. Several architectures exhibit repeated downward excursions during the active day and then return toward zero when the relevant window is effectively closed and reconciled. In other words, the figure makes visible the distinction between provisional intra-day exposure and the accepted daily result. The interesting difference from S1 and S2 here is that the deviation is consistently negative instead of oscillating around the zero mark. This can be attributed to the longer freshness window, where we by definition have a larger number of new events with positive sales figures rather than deletions with negative sales figures. The effect of an individual event missing from the architecture is therefore not as pronounced when the freshness cadence is longer.

This helps clarify why a scenario can tolerate substantial intra-window deviation and remain analytically coherent. S3 does not require the architectures to present the fully settled truth of the current day at every moment. It asks them to produce a dependable result for the latest closed day after waiting a set amount of time. Architecture C stays especially close to zero because the daily-window object is already close to its native semantics. The other architectures reveal more visibly provisional daily accumulation, but that provisionality is no longer disqualifying, because the business object being evaluated is no longer the open present.

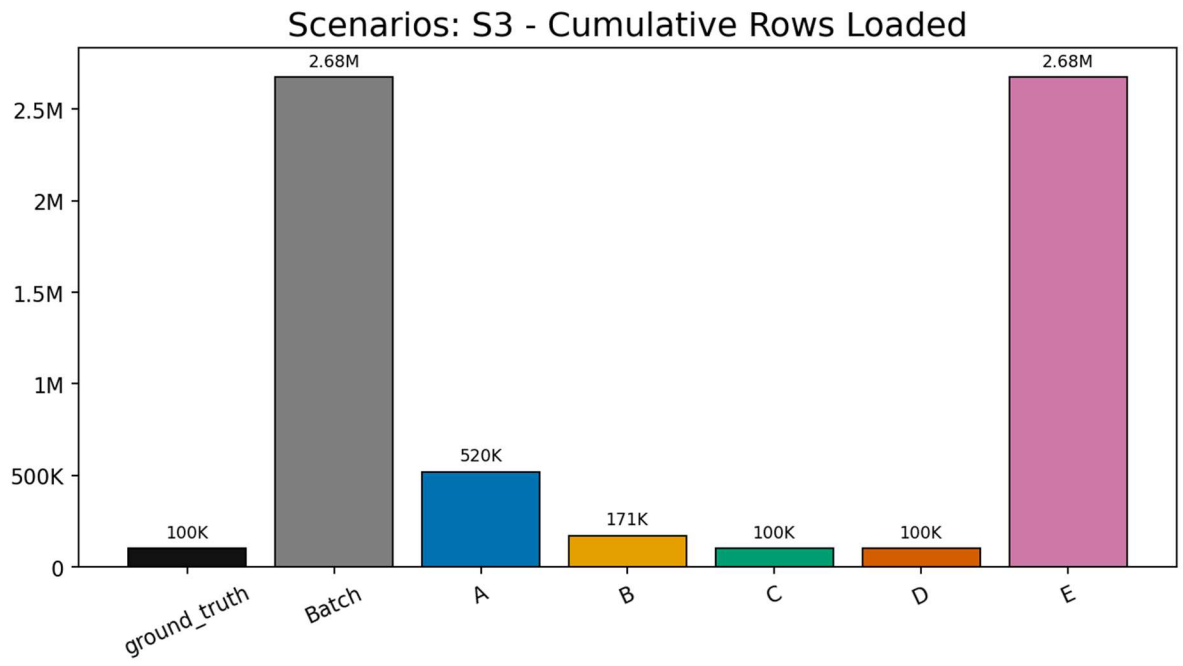

Figure 11. S3 Cumulative rows loaded

This shift further compresses the processing differences between the architectures. The more streaming-like architectures do not lose their advantage, however. They remain close to source-row scale, which indicates that being well aligned with a daily window contract does not require abandoning incremental or continuously maintained semantics. Rather, S3 demonstrates that once the reporting object becomes a settled window, architectural differences begin to reflect fit to a closure model rather than sheer ability to stay fresh.

5.8.5 Scenario 4 – Weekly Management Review

S4 resembles a traditional periodic managerial reporting requirement more than an operational dashboard. The result is no longer meant to function as a live pulse, but as a stable weekly review object that should be both highly accurate and effectively final once exposed. Consequently, the architectures are now allowed long stretches in which the visible state may remain unchanged if that is the most coherent way to produce a stable weekly result. Whether an architecture uses that freedom can be observed in Figure 12, and the corresponding maintenance burden is shown in Figure 13.

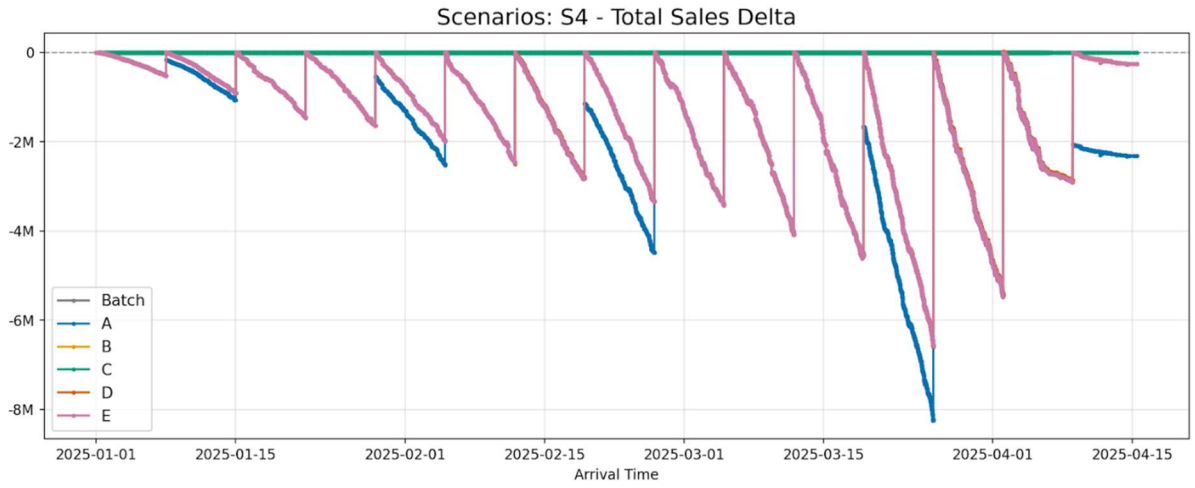

Figure 12. S4 Total sales delta

The batch reference, Architecture A, and Architecture E favour a punctuated weekly pattern in which the visible state remains relatively unchanged for extended periods and then moves sharply when the weekly result is refreshed. In the case of the batch reference and Architecture E, the visible behaviour is so similar that the lines overlap. Yet the semantic route to that similarity is not identical: the batch reference achieves it through classical snapshot replacement, whereas E achieves a very similar visible contract through semantic refresh over an already active state. In practice this means that E can produce intra-week correct results if required, as the architecture materializes the analytical state at query time only. The batch architecture cannot provide similar flexibility without reprocessing the batch.

Architectures B and D remain more continuously responsive even though the scenario does not require them to be. They preserve a more active visible relationship to the source history despite the relaxed weekly cadence. Architecture C again remains close to zero because the scenario evaluates a bounded reporting window instead of the open present.

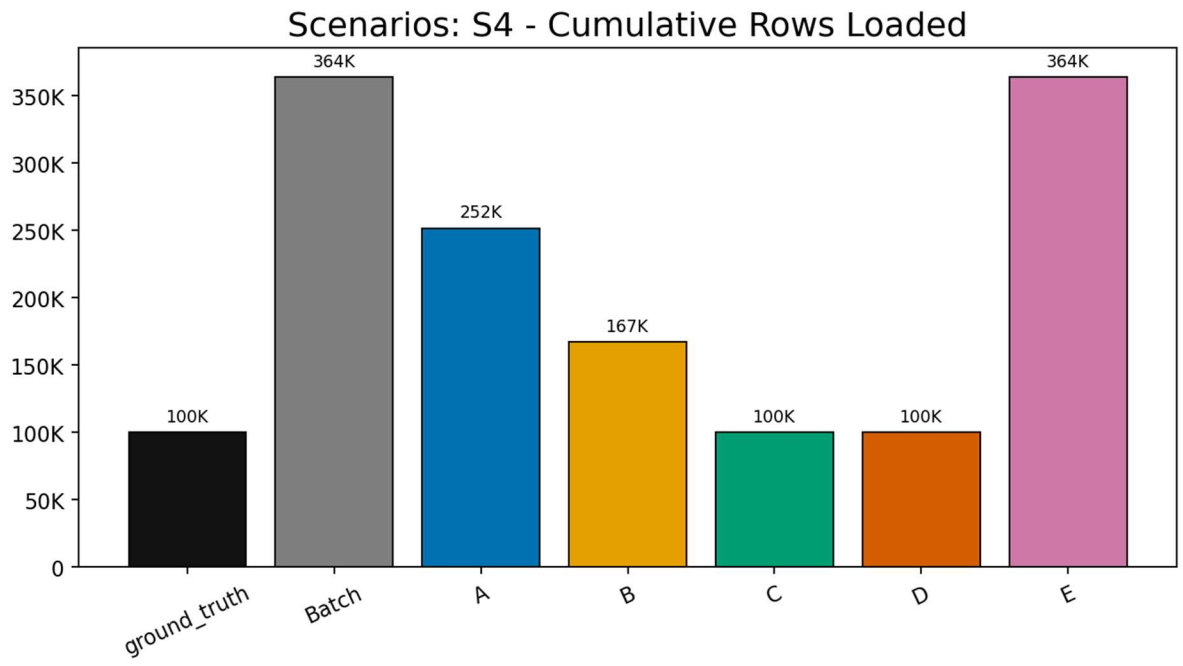

Figure 13. S4 Cumulative rows loaded

The computational costs presented in Figure 13 converge further than in the earlier scenarios, but they do not collapse into equality. The more reconstructive architectures are no longer under the extreme pressure seen in the live cases, yet they still carry extra burden because producing a closed and stable weekly object through repeated rebuilding remains more expensive than maintaining a state that is already incrementally current.

5.8.6 Scenario 5 – Combined Requirements

S5 changes the problem more radically than the preceding scenarios. The earlier cases allow the architectures to adapt separately to different business needs. S5 removes that freedom and asks whether one shared parameterization can satisfy low-latency visibility, daily settlement, and weekly stability at the same time. This is a test of whether multiple analytical contracts can coexist inside one shared model without one of them dominating the others. This question is first illustrated through the visible trajectory in Figure 14 and then through the loading burden in Figure 15.

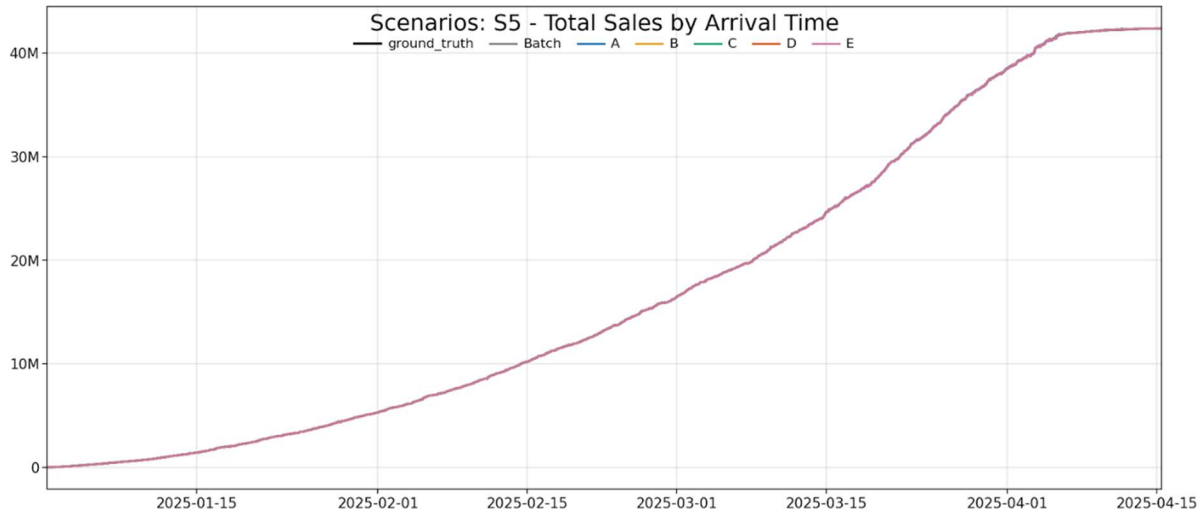

Figure 14. S5 Total sales by arrival time

As illustrated, this scenario almost eliminates visible separation between the architectures, much like in S1. Effectively the visible state of the architectures corresponds exactly to the setting in S1. This behaviour indicates that once all use cases are forced into one shared parameterization, the strictest live requirement pulls the visible model toward a homogeneous present-tense view. In practice this means that the analytical contract suppresses many of the differences that would otherwise appear when the use cases are allowed separate parameterizations. The figure is therefore best read as evidence of dominance inside the shared model: once near-live visibility must be supported continuously, slower reporting requirements no longer determine the visible shape of the system.

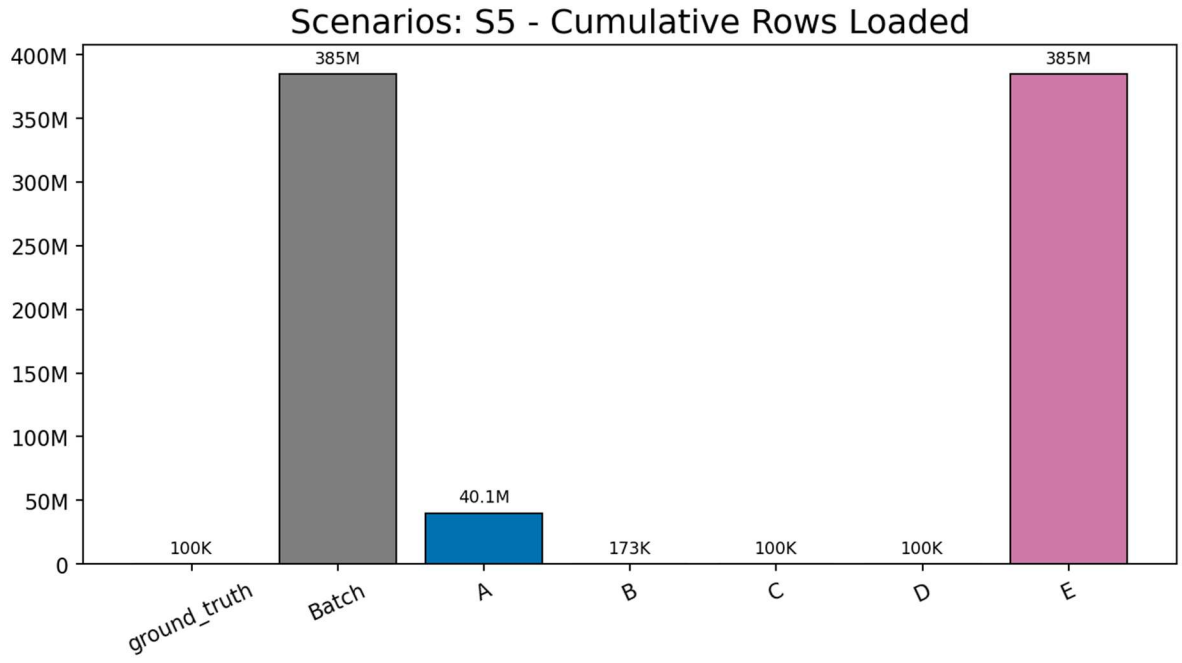

Figure 15. S5 Cumulative rows loaded

Figure 15 shows the consequence of that dominance. The combined case does not generate a compromise profile between the earlier scenarios. Instead, it falls back toward the economics of the strictest live requirement. Once one shared parameterization must remain suitable for near-live visibility, the rest of the contract no longer softens the underlying cost structure. Architectures that rely on repeated closure must continue to refresh aggressively enough for the live requirement, and the daily and weekly use cases simply inherit that burden. Architectures that maintain an open or incrementally reconciled state, by contrast, can absorb the combined contract at a much lower loading cost because supporting the live requirement already gives them most of what the slower scenarios need.

5.9 Interpretation

The results show that the architecture patterns do not only differ in how quickly they produce the same analytical state. The baseline scenario already demonstrates that they begin from different native analytical contracts and separate into visibly different behavioural families. The batch-oriented cases depend on punctuated closure, Architecture B exposes a more openly provisional reconciled state, and Architecture C remains tied to a bounded visible history instead of a fully accumulated cumulative total. The baseline therefore confirms the central conceptual argument of the thesis: temporal closure is not only a delivery mechanism, but part of what makes an analytical result interpretable in the first place.

Under explicit business contracts, visible differences narrow, but the maintenance burden diverges sharply. In user-facing terms, several architectures can be made to present a sufficiently similar live sales view, and the delta plots show repeated return toward ground truth. The divergence therefore moves away from the visible line itself and toward the amount of reconstruction required to keep that line acceptable. The cumulative loading figures show that low-latency visibility is far more expensive for the closure-oriented architectures, because they must repeatedly recreate a sufficiently current analytical state from already observed history. Architectures that keep an open or incrementally reconciled state visible can satisfy the same live contract with far less repeated processing.

This balance changes in S3 and S4 because the reporting object is no longer the open present, but a settled daily or weekly window. Once the result is allowed time to close before it is interpreted, the semantic gap between the architectures narrows and the visible differences become less consequential than in the live dashboard scenarios. The remaining contrast is no longer primarily about whether the architecture can appear current at every moment, but about how it produces a dependable closed result. The processing differences do not disappear, but they become less extreme, which supports the broader claim that the cost of batch-like closure is most severe when a stable reporting object is demanded under near-live conditions.

S5 brings these patterns together by removing the possibility of separate scenario-specific parameterizations. When multiple requirements are forced into one shared model, the strictest live requirement becomes structurally dominant, and slower reporting requirements inherit its maintenance logic. Figure 16 makes that mechanism concrete by showing where the work accumulates: the reconstructive architectures repeatedly revisit the oldest already-known rows, whereas the open-state architectures remain close to one load per row across the history. In other words, the figure shows whether maintaining the visible analytical state mainly requires reprocessing already seen history or simply incorporating each new row once as it arrives.

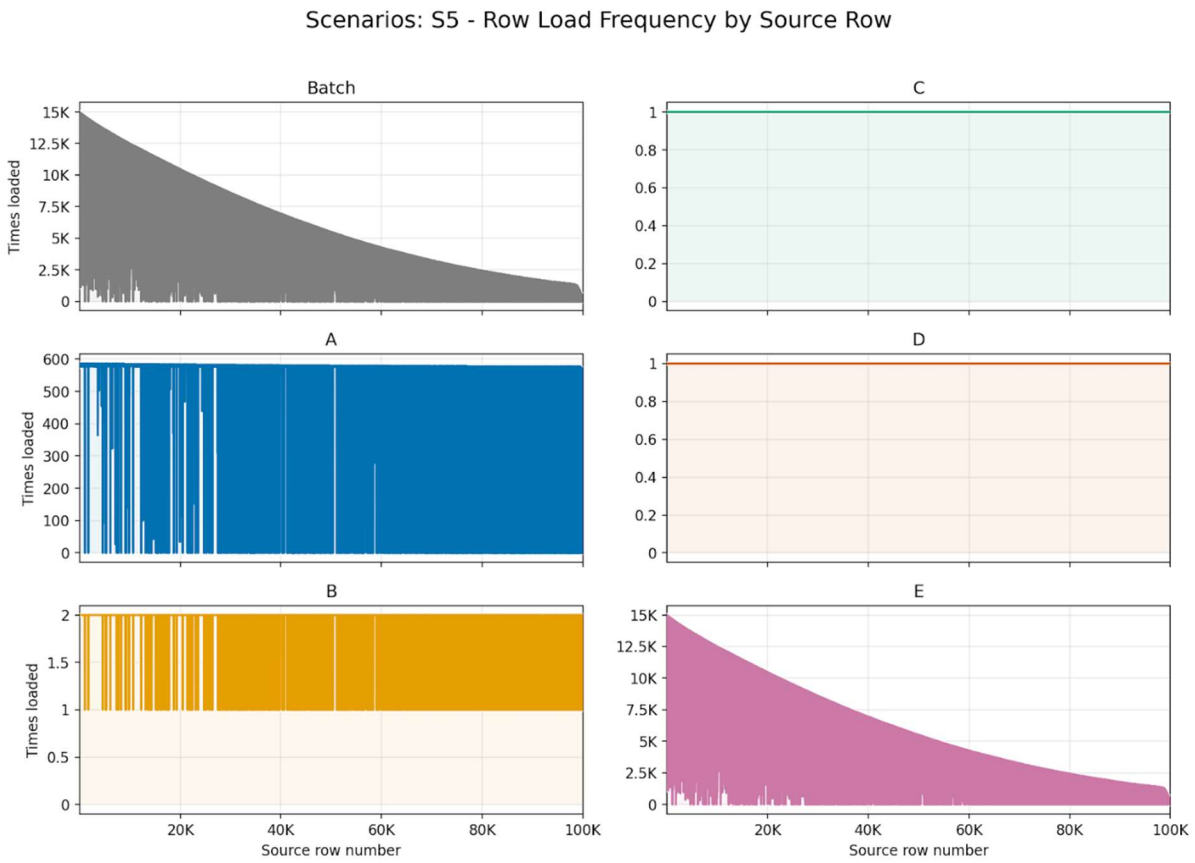

Figure 16. S5 Row load frequency by source row

For snapshot architectures, the earliest rows observed are processed the greatest number of times. The open-state architectures remain much flatter, staying close to one load per row across the history. Architecture A is the middle ground between these two paradigms, as it relies on both an incremental state and a periodic snapshot reconciliation. The computational cost is not only that some architectures process more rows in total, but that reconstructive closure repeatedly reprocesses the already known past to keep the present interpretable. The longer the history grows, and the more events it contains, the larger the batch-overhead grows. In that sense, the move away from batch processing is not simply a matter of reducing latency, but of choosing what kind of analytical contract the analytics layer can afford to maintain.

In relation to the research questions, the simulation therefore shows that reducing batch-oriented reconstruction changes the native analytical contract exposed by the architecture (RQ2), while also demonstrating that architectures with different native semantics can nevertheless be adapted to satisfy similar business-facing analytical requirements through different closure mechanisms and maintenance strategies (RQ3).

6 Discussion

The research questions were examined through two complementary analytical perspectives. The systematic literature review established how the paradigms identified in prior research can be positioned along the six analytical dimensions proposed in this thesis. The simulation study, in turn, evaluated how the corresponding architectural patterns behaved under a shared set of business requirements. This chapter synthesizes these two strands to discuss how the reduction or removal of batch-oriented reconstruction changes the interpretability conditions of analytics-layer models. Accordingly, the discussion is organized around the thesis's research questions, using the combined evidence of the literature review and the simulation study to clarify both the conceptual and practical implications of reduced batch reconstruction for analytics-layer modeling.

6.1 Main findings

The findings show that reducing batch reconstruction changes more than freshness. It changes how interpretability is maintained. In the closed snapshot warehouse pattern, correctness is retrospective, closure follows the completed load cycle, history remains recomputable, and consistency is strong within each processed state. The batch reference and Architecture A most clearly follow this logic, while Architecture E preserves a similar effect by moving snapshot formation to query time. This is broadly consistent with the classical warehouse view, but also begins to move beyond it by treating closure, revision, and consistency not only as implementation consequences of warehousing practice, but as explicit interpretability conditions of analytics-layer models (Inmon, 2002, pp. 31–37; Kimball & Ross, 2013, pp. 7–16, 62–63). A closely related assumption is visible in classical warehouse design itself. Inmon (2002, pp. 31–37) describes the warehouse as time-variant and non-volatile, while Kimball and Ross (2013, pp. 7–16, 62–63) tie analytical correctness to stable grain, conformed dimensions, and historically valid dimensional context. This interpretation is, however, in partial tension with Kimball and Ross's treatment of real-time delivery. While they explicitly recognize that lower latency weakens the time available for cleansing, validation, and conformance, their discussion still largely frames real-time analytics as a demanding extension of dimensional warehousing that can be addressed through architectural adaptations such as real-time partitions, placeholder dimensional context, and later reconciliation, rather than as a qualitative shift in the interpretability conditions of analytics-layer models (Kimball & Ross, 2013, pp. 4, 520–525). The present thesis agrees that familiar dimensional contracts

can often still be preserved, but argues that once batch-oriented reconstruction weakens, truth, closure, revision, and semantic consistency can no longer remain implicit in the architecture to the same extent.

This helps explain why the SLR did not produce one real-time replacement for the batch architecture. The pattern groups differ because they break this batch-oriented alignment in different ways: Architecture B weakens closure and accepts a more provisional analytical truth, Architecture C narrows truth to a maintained window, and Architecture D anchors truth and consistency in transactional visibility. What changes from pattern to pattern is not only the speed of data movement, but the way the six dimensions are combined into an interpretable whole. This emphasis also differentiates the present thesis from the more infrastructure-oriented architectural literature cited earlier. For example, Lakehouse literature frames current analytical challenges primarily through staleness, ETL complexity, reliability, management features, and performance over open storage formats, arguing that these challenges can be mitigated through metadata layers, versioning, caching, indexing, and layout optimization (Armbrust et al., 2021, pp. 1–5). The present findings do not contradict the importance of those capabilities, but they suggest that improved infrastructure does not by itself resolve the interpretive problem. Even when fresh analytics becomes technically feasible, truth, closure, revision, and semantic consistency still need to be defined and maintained explicitly at the analytics layer. This is in line with broader comparative work, which treats decisional architecture as a field of alternatives shaped by constraints in data volume, velocity, storage, and processing rather than as a single path of development. Ben Aissa et al. (2020, pp. 1–8) classify architectures such as BI, Big-ETL, Big-Warehouse, Data-Lake plus Data-Warehouse, and Big-Architecture according to whether the environment requires real-time stream processing, data-driven historical analysis, high-volume storage, or complex distributed analysis. Qu and Wang (2024, pp. 110–117) similarly describe the contemporary real-time warehouse landscape through several distinct forms, including Lambda, Kappa, data lake, lakehouse, HTAP, and HSAP architectures, each emphasizing different trade-offs in freshness, scalability, and integration.

The baseline scenario in the simulation study demonstrates the effects of this. Each architecture observed in its native state has its own notion of what is true, accurate, fresh, and stable. At this point, a contrast appears between the SLR and the empirical study. The literature highlights conceptual divergence between architecture families. The simulated business scenarios, by contrast, hold the analytical requirements, data model, and source data

structure constant, removing much of the variation in analytical abstraction and making the remaining temporal and semantic differences visible through behaviour and computational cost instead. In other words, the literature shows how far apart these paradigms are in principle, while the simulation shows how close some of them can be brought in practice under shared business requirements.

Before any scenario-specific tuning was applied, the architectures separated into distinct behavioural families. The batch reference showed long periods of lag followed by return to a reconciled state. Architecture A stayed closer to ground truth but still did so through a reconstructive logic. Architecture B exposed a more openly provisional present. Architecture C remained tied to a bounded history instead of cumulative truth. Architecture D tracked a highly current state, but on transactional rather than batch terms. The observed differences show that the architectures begin from different positions in analytical truth, temporal closure, and semantic consistency before any business target is imposed. In that sense, the baseline simulation reproduces at the behavioural level the same conceptual distinctions that were identified in the literature review between snapshot-oriented, provisional, window-bounded, and transactionally grounded architectures.

Once explicit business contracts are introduced, a different pattern appears. All architectures pass the formal thresholds, and from the end user perspective they appear interchangeable. Yet the loading burden shows that similar visible results can rest on very different semantic arrangements. The closure-oriented architectures maintain meaning by repeatedly reconstructing already seen history, while the more open architectures maintain meaning by keeping an already current state visible and reconcilable. The main difference under shared business thresholds is no longer whether a familiar analytical contract can be reached, but how much architectural work is required to sustain it. In that sense, the simulation shows that conceptual divergence does not prevent practical convergence at the user-facing level, but shifts the difference into the closure mechanism, maintenance burden, and supporting structures needed to preserve the visible result. At the same time, this result sits in partial tension with the stronger implication in classical OLAP and warehousing literature that stable analytical interaction presupposes evaluation over a sufficiently stable and processed analytical state (Chaudhuri & Dayal, 1997, pp. 65–69; Kimball & Ross, 2013, pp. 7–16, 520–525). The simulation suggests instead that comparable user-facing analytical contracts may also be produced through architectures whose native state is more open, provisional, or transactionally scoped, provided that sufficient stabilization is introduced elsewhere. This is

also compatible with comparative architecture work, which presents different decisional structures as responses to different requirements rather than as mutually exclusive notions of what a real-time analytical system must be. In both Ben Aissa et al. (2020, pp. 1–8) and Qu and Wang (2024, pp. 110–112), architectures are compared not as better and worse versions of the same model, but as alternatives that become appropriate under different combinations of stream processing needs, historical analysis requirements, storage scale, and processing complexity.

Under the near-real-time requirements of S1, the closure-oriented architectures required orders of magnitude more repeated loading than the more open architectures, with the batch reference and Architecture E at the highest end and Architectures C and D remaining closest to source-row scale. The main contrast here is not between accurate and inaccurate architectures, but between architectures that must repeatedly recreate closure and architectures that can operate with a more open state. Earlier near-real-time warehousing work pointed in the same direction. Ferreira et al. (2013, pp. 73–75) show that stronger warehouse-style refresh under tighter update cycles quickly becomes expensive, while Liu and Iftikhar (2015, pp. 1015–1022) treat faster refresh as an ETL optimization problem requiring partitioning and parallelization. Comparative performance work similarly reports accuracy-resource trade-offs, for example when Lambda outperformed Kappa at the cost of longer processing time and higher CPU and memory usage (Sanla & Numnonda, 2019, pp. 1–5). The present simulation extends that line of work by showing that the same underlying burden is not only technical but interpretive. In this respect, the result is less compatible with infrastructure-first accounts that primarily frame low-latency analytics as a problem of platform design, metadata management, and performance over open storage formats (Armbrust et al., 2021, pp. 1–5). The simulation suggests that even when such architectures make freshness technically attainable, the decisive cost still depends on what degree of closure must be reconstructed before a result is treated as analytically interpretable. The more strongly an architecture depends on repeated closure, the more expensive it becomes to preserve a stable analytical contract under low-latency conditions.

The shared-model scenario S5 adds a useful qualification to this discussion because it removes the freedom to tune architectures separately for different use cases. Comparative architecture literature usually presents alternative architectures as appropriate for different requirement profiles, for example depending on the relative importance of real-time processing, historical analysis, storage scale, or processing complexity (Ben Aissa et al.,

2020, pp. 1–8; Qu & Wang, 2024, pp. 110–112). The present simulation suggests that when those requirement profiles are instead forced into one shared analytics layer, the strictest live requirement can become semantically dominant, causing slower reporting needs to inherit its maintenance logic. In that sense, S5 does not contradict the comparative literature, but qualifies it by showing that architectural plurality at the design-space level does not automatically translate into plurality within one shared model.

The literature review suggests that weaker closure and more provisional truth lead to different analytical behaviour and consumption semantics. The simulation confirms this in the baseline, but it also shows that under explicit business thresholds those differences can be compressed at the user-facing level. The main contrast then shifts from visible output to the cost and supporting structures required to maintain that output. In particular, the streaming-style architectures required an intermediate streaming layer between the operational database and the loading logic, indicating that the move away from batch semantics may relocate stabilization instead of eliminating it. This is consistent with broader developments in analytics architecture, but not without qualification. Bomma (2023, pp. 94–98) places semantic stabilization in a governed semantic layer above the warehouse, where business-facing measures, hierarchies, calculated fields, and access rules are defined separately from the underlying storage structures. The present thesis partly supports this view but also places it under tension by showing that semantic-layer governance alone does not remove the need for temporal stabilization when the underlying analytical state remains open, revisable, or only partially closed. In the classical warehouse literature, similar stabilizing work was embedded more directly in integrated warehouse state and dimensional design conventions, such as fixed fact grain, conformed dimensions, and historically managed dimensional context (Inmon, 2002, pp. 31–37, Kimball & Ross, 2013, pp. 7–16, 50–56, 62–63). The finding of this thesis is therefore not simply that stabilization may move, but that once batch reconstruction weakens, some other mechanism must take over its role if interpretability is to be preserved.

This may indicate that streaming architectures provide additional ways of tuning analytical guarantees, but that claim remains speculative within the scope of this thesis and requires further research. Even so, the combined evidence from the literature review and the simulation supports one broader conclusion. The transition toward real-time analytics should not be understood only as a move toward lower latency. It should be understood as a shift in how the analytics layer constructs and maintains the conditions under which its outputs remain interpretable.

6.2 Answering the research questions

This section answers the research questions directly by drawing together the conceptual framework of Section 2.5 with the literature review in Chapter 4 and the simulation results in Chapter 5.

6.2.1 RQ1 - What temporal and semantic assumptions are embedded in analytics-layer models, and how does batch-based processing support these assumptions?

The thesis explored the assumptions embedded in analytics-layer models through six dimensions: analytical truth, temporal closure, history mutability, semantic stabilization, analytical abstraction, and semantic consistency. A central finding is that these assumptions are treated differently across architectural paradigms. In some paradigms they remain implicit and are inherited from the surrounding architecture, whereas in others they are articulated more explicitly through declared rules of correctness, closure, revision, and consistency.

The literature review shows that batch-oriented analytics supports these assumptions by aligning them before analytical exposure. In the closed snapshot warehouse pattern, correctness is retrospective, closure follows the completed load cycle, history remains recomputable, and consistency is strong within each processed state. Analytical meaning is therefore stabilized in advance rather than negotiated at the moment of use.

The simulation supports the same conclusion. In the baseline scenario, the batch reference and the more closure-oriented architectures exposed punctuated, reconciled states whose meaning depended on prior closure. Batch-based processing therefore does more than move data into analytical storage. It acts as an implicit interpretive contract by aligning the temporal and semantic conditions under which analytical results can be treated as meaningful.

6.2.2 RQ2 - What modeling challenges emerge when batch-based reconstruction of analytical history is removed?

Removing batch-based reconstruction turns previously implicit interpretive guarantees into explicit modeling problems.

The literature review shows that once analytical exposure is no longer delayed until after a completed reconstruction cycle, the analytics layer can no longer assume that all relevant

events have arrived, that facts and dimensions are synchronized, or that late corrections have already been absorbed into a coherent historical state. What batch processing previously settled by default must now be defined and maintained deliberately.

The simulation makes this visible by showing that the architectures do not simply expose the same analytical state at different speeds. They expose different analytical contracts. Some present punctuated and reconciled states, some provisional continuously open states, some bounded window-relative states, and some transactionally visible states. Reducing batch reconstruction therefore changes not only latency, but the native semantics of the analytically visible state.

More specifically, these findings point to five recurring modeling challenges. First, correctness must be defined without relying on a fully closed historical state. Second, closure must be made explicit, because the architecture must decide when a result is complete enough to interpret even though further events or corrections may still arrive. Third, revision becomes a first-class concern, because exposed results may need to change after initial consumption. Fourth, semantic consistency becomes harder to preserve when facts, dimensions, and corrections evolve asynchronously. Fifth, multiple use cases may impose competing temporal and semantic expectations on the same analytics layer.

The shared-model scenario makes the final challenge especially clear. When several requirement profiles were forced into one common architecture, the strictest live requirement became dominant and the slower reporting needs inherited its maintenance logic. The central challenge is therefore not simply how to expose fresher data, but how to maintain a coherent analytical contract when the analytical state remains open, revisable, or continuously updated.

6.2.3 RQ3 - What modeling strategies or paradigms have been proposed to address these challenges in continuous analytics environments?

Continuous analytics environments are not supported by one universal replacement for batch-oriented analytics, but by several distinct strategies for reconstructing enough stability for analytical interpretation.

The literature review identifies five principal architecture families: the closed snapshot warehouse, the open evolving stream, the window-bounded stream, the log-consistent HTAP architecture, and the virtual semantic snapshot. These families differ in how they define truth, establish closure, admit revision, stabilize semantics, and expose analytical meaning. Some

preserve stronger snapshot-style interpretability by accelerating or virtualizing closure. Others accept more openly evolving state and bounded inconsistency, narrow correctness to bounded windows, or anchor correctness in transactional visibility.

The simulation sharpens this result. Although these architectures begin from different native temporal and semantic positions, they can still be adapted to satisfy similar business-facing analytical requirements under controlled conditions. The key difference is therefore not primarily whether a familiar analytical contract can be achieved, but how it is achieved. What varies is the closure mechanism, the maintenance burden, and the need for supporting intermediary structures.

The answer to RQ3 is therefore that continuous analytics requires alternative ways of reconstructing truth, closure, revision control, and consistency once batch-oriented reconstruction weakens. The contribution of this thesis is to show that these alternatives can be compared systematically through the six-dimensional framework and that, under shared business requirements, their most consequential differences may appear less in the visible analytical output than in the architectural work required to sustain it.

6.3 Limitations of the study

Both parts of the thesis necessarily abstract from some of the complexity of real-world enterprise analytics environments. Most of the papers included in the review describe systems architecturally, except for a few papers on deployed implementations. The simulation is similarly controlled by design: the whole design concept was to hold source conditions constant across architectures, which means the results do not by themselves support direct generalization to how these patterns would behave in real infrastructure under actual user behavior. That is a deliberate trade-off, but it does mean the findings are more useful for comparing architectural logic than for judging production readiness.

The literature review is also limited by the size and fragmentation of the field. Only 14 studies met the final inclusion criteria, which reflects the genuine sparsity of research that addresses explicitly analytics-layer modeling under real-time constraints. The framing of the literature review was more aligned with analytics modeling than the design of the simulation study. This poses a slight misalignment in the results, as the corpus of the literature review may not have captured some relevant studies that were primarily aligned with analytics architectures rather than with analytics-layer modeling implications. In addition, the screening and coding

of the literature were conducted by the author alone, without a second reviewer or formal inter-rater agreement procedure. Although the review followed a transparent and conservative process, this introduces a risk of interpretive subjectivity in study selection, coding, and pattern assignment.

The empirical burden comparison is intentionally narrower than a full performance benchmark. Computational cost is represented by rows loaded into the architecture, or in the case of Architecture E, rows read at query time. This gives a useful indication of how much work is required to maintain or materialize the analytical state under comparable conditions, but it does not directly translate into real-world performance. In a live environment, actual performance depends on hardware, the software stack, source database behaviour, storage layout, networking, orchestration, and concurrency. The simulation therefore captures relative maintenance burden rather than absolute runtime behaviour.

This choice of metric also means that one part of performance is observed unevenly. Query-time performance is measured explicitly only for Architecture E, because in that architecture query-time materialization is itself the mechanism through which the analytical state becomes visible. For the remaining architectures, query cost was not included in the comparative metric, since the purpose of the simulation was to evaluate how the analytical state is maintained and not how fast each already materialized state can be queried. This makes the comparison appropriate for the research question addressed here, but it also means that full end-to-end user-facing performance remains outside the scope of the study.

Another limitation follows from the deliberately simplified analytical model. All architectures were evaluated with one fact table, one additive measure, and minimal transformation logic. This was necessary to keep the comparison focused on temporal and semantic architecture behaviour, but it also means that the observed performance characteristics should not be assumed to transfer directly to richer analytical models. In practice, conformed dimensions, slowly changing dimensions, multi-fact designs, join-heavy transformations, and more complex business measures can change both maintenance burden and query behaviour in major ways.

The source setting introduces another limitation. The simulation uses a mutable operational database as the shared source and evaluates the streaming-oriented cases through an intermediate streaming layer and not through a native event-stream source. This choice keeps the architectures comparable under one shared source history and reflects the fact that many

organizations still source analytics from mutable operational systems. Likewise, it means that the results should not be read as a direct benchmark of streaming systems operating over naturally event-oriented sources with source-native semantics. In addition, the computational cost associated with maintaining architecture-specific requirements, such as intermediate streaming layers, is not measured. The performance implications of an additional streaming layer are highly dependent on the environmental factors and cannot therefore be accounted for within the scope of this study.

A further limitation concerns the design of the business scenarios themselves. The scenario runs were constructed by the author as exemplary requirement settings intended to represent plausible analytical use cases, but they were not meant to cover the full diversity of real-world business requirements. In particular, the scenarios were strongly motivated by requirement profiles commonly associated with batch-based analytics environments, such as stable reporting windows, bounded refresh cycles, and explicit correctness targets. This makes them useful for comparative analysis in the context of the thesis, but it also means that the results may not generalize directly to use cases built around different priorities, such as exploratory streaming consumption, approximate analytics, or event-native decision processes. In addition, the scenario runs were designed as acceptance-condition evaluations rather than as attempts to engineer the strongest possible production implementation of each architecture family. Once a passing parameterization existed, the simulation selected settings that met the active scenario without stronger tuning than the scenario required. This is consistent with the purpose of the simulation, but it also means that the results should not be interpreted as the maximum attainable performance of each pattern in absolute terms.

6.4 Topics for further research

The findings of this thesis suggest several directions for further research. The most immediate need is to test the six-dimensional framework in live enterprise environments. Applying the framework to real organizational analytics stacks would make it possible to examine whether the same dimensions remain useful when interpretability, governance, and operational trade-offs are observed in practice. Such work would also help determine whether additional dimensions are needed, especially around provenance communication, governance, or user-facing disclosure of provisionality.

Further empirical work is also needed on more complex analytics-layer models. The present study intentionally isolates architecture behaviour by holding the analytical model almost

constant. Future research should examine whether the same patterns hold when the analytical layer includes conformed dimensions, slowly changing dimensions, non-additive measures, multiple fact tables, semantic-layer logic, and more substantial transformation pipelines.

The role of the source contract also deserves closer study. In this thesis, the streaming-oriented architectures achieve their results through an intermediate streaming layer built on top of a mutable operational database. That already suggests that the move from batch semantics to streaming semantics might be dependent on adding an intermediate layer on top of the source database. Future work should therefore further examine the role of the intermediate layer, its role in the analytics architecture stack and the performance implications of maintaining one.

More realistic performance testing in deployed production-like environments should be conducted to evaluate the true relationship between architectural behaviour and scalability in real environments. Rows loaded proved to be a useful comparative indicator of computational cost, but since the goal of transitioning away from batch-based analytics is reducing latency, especially the processing times of maintaining each architecture should be evaluated. For example, deploying a micro-batch architecture on a 10-minute refresh schedule is hardly practical if processing the micro-batch requires more than 10 minutes.

7 Conclusion

This thesis examined how analytics-layer modeling changes when batch-oriented reconstruction is no longer the default way of stabilizing analytical state. It has shown that the transition toward real-time analytics is not only a matter of reduced latency, but a change in the temporal and semantic conditions under which analytics-layer models remain interpretable. To study that shift, the thesis developed a six-dimension framework covering analytical truth, temporal closure, history mutability, semantic stabilization, analytical abstraction, and semantic consistency. The framework made it possible to compare architectures that may appear similar at the level of latency but differ substantially in the interpretability they provide.

The literature review showed that the move away from batch-oriented analytics does not produce one universal replacement architecture, but several distinct design families. These families differ in how they define correctness, establish closure, admit revision, stabilize semantics, and present analytical meaning to downstream consumers. The simulation study complements this by showing that architectures with different native semantics can nevertheless be adapted to satisfy similar analytical contracts. The main difference is therefore not necessarily whether a familiar user-facing result can be achieved, but how that result is produced, including the closure mechanism, the maintenance burden, and the supporting structures required to sustain it.

At the same time, the simulation makes clear that these similarities at the level of the visible analytical contract do not imply equivalent architectural behavior. Some architectures achieved compliance through stronger closure and repeatability at the cost of higher maintenance effort, while others relied on more incremental or continuously updated mechanisms with different load behavior and operational burdens. A key qualification is that the streaming-native architectures met the shared requirements only when an intermediary streaming layer was introduced between the mutable operational source and the streaming architecture itself. This suggests that moving away from batch-oriented reconstruction does not remove the need for stabilization but may instead relocate it into a separate architectural layer.

The thesis answers the three research questions followingly. First, analytics-layer models rely on temporal and semantic assumptions that batch-oriented architectures have historically

stabilized, often without making those assumptions explicit. Second, when batch-based closure and historical reconstruction are weakened, the main challenge is the loss of implicit alignment between truth, closure, correction, and repeatability. Third, the literature does not point to one universal real-time replacement for this lost alignment, but to several architectural responses, each of which preserves different aspects of interpretability under continuous data arrival. The main contribution of the thesis lies in clarifying these trade-offs and in showing that the different architectures may under certain conditions be adapted to satisfy the same analytical requirements.

For practitioners, the findings suggest that architectural decisions about real-time analytics should not be evaluated only in terms of latency, freshness, or infrastructure efficiency, but also in terms of the interpretability guarantees that the analytics layer is expected to provide. The simulation results indicate that all tested architectures could satisfy similar analytical contracts, including requirements commonly associated with batch-native behavior, but they differed substantially in how those requirements were achieved. One possible implication is that the more streaming-oriented architectures may provide greater room to tune stability, accuracy, freshness, and history mutability to the needs of a specific use case, because they are not tied as tightly to a single inherited snapshot logic. At the same time, this apparent flexibility depends on additional supporting structures and comes with different maintenance patterns and operational trade-offs. These conclusions should be read within the limits of a conceptually oriented study based on a literature synthesis and a controlled simulation and not as full production benchmarking.

This thesis found that batch-oriented architectures quietly settled a set of interpretability questions that never had to be asked. The transition to real-time architectures requires answering those same questions explicitly rather than abandoning them, and the choices made while doing so shape what the analytics layer can promise to the practitioners and end-users who rely on it.

8 Disclosure on the use of Artificial Intelligence

Artificial intelligence tools were used during the preparation of this thesis. Elsevier Scopus AI was used to assist in identifying core literature around the research topic. OpenAI GPT-5 was used during the early stages of formulating the research topic, refining the research questions, and identifying initial reference candidates.

OpenAI GPT 5.2 and GPT 5.4 were used to assist in drafting and stylizing portions of the thesis text.

GitHub Copilot was used to generate the project template and portions of the simulation code, and to refine code written by the author.

The author takes full responsibility for all content, arguments, and conclusions presented in this thesis.

References

Abadi, D. J., Boncz, P. A., & Harizopoulos, S. (2009). Column-oriented database systems. *Proceedings of the VLDB Endowment, 2*(2), 1664–1665. https://doi.org/10.14778/1687553.1687625

Armbrust, M., Ghodsi, A., Xin, R., & Zaharia, M. (2021). Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics. In *Proceedings of the 11th Conference on Innovative Data Systems Research (CIDR 2021)*. https://www.cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf

Ben Aissa, M. M., Sfaxi, L., & Robbana, R. (2020). Decisional architectures from business intelligence to big data: Challenges and opportunities. In *Proceedings of the 2nd International Conference on Digital Tools & Uses Congress* (pp. 1–9). Association for Computing Machinery. https://doi.org/10.1145/3423603.3424049

Bhagat, A., & Deshpande, M. (2023). Real-time assessment of live feeds in big data. In A. Joshi, M. Mahmud, & R. G. Ragel (Eds.), *Information and Communication Technology for Competitive Strategies (ICTCS 2021)* (pp. 29–39). *Lecture Notes in Networks and Systems* (Vol. 400). Springer. https://doi.org/10.1007/978-981-19-0095-2_4

Bomma, H. P. (2023). Enhancing business analytics with a semantic layer. *International Journal of Business Quantitative Economics and Applied Management Research, 7*(9), 93–99. https://doi.org/10.5281/zenodo.14969896

Braun, V., & Clarke, V. (2012). Thematic analysis. In H. Cooper, P. M. Camic, D. L. Long, A. T. Panter, D. Rindskopf, & K. J. Sher (Eds.), *APA handbook of research methods in psychology, Vol. 2: Research designs: Quantitative, qualitative, neuropsychological, and biological* (pp. 57–71). American Psychological Association. https://doi.org/10.1037/13620-004

Chaudhari, A. V., & Charate, P. A. (2025). Optimizing data lakehouse architectures for scalable real-time analytics. *International Journal of Scientific Research in Science, Engineering and Technology, 12*(2), 809–822. https://doi.org/10.32628/IJSRSET25122198

Chaudhuri, S., & Dayal, U. (1997). An overview of data warehousing and OLAP technology. *SIGMOD Record, 26*(1), 65–74. https://doi.org/10.1145/248603.248616

Chen, H., Chiang, R. H. L., & Storey, V. C. (2012). Business intelligence and analytics: From big data to big impact. *MIS Quarterly, 36*(4), 1165–1188. https://doi.org/10.2307/41703503

Codd, E. F., Codd, S. B., & Salley, C. T. (1993). *Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate*. Codd & Associates.

Dageville, B., Cruanes, T., Zukowski, M., Antonov, V., Avanes, A., Bock, J., Claybaugh, J., Engovatov, D., Hentschel, M., Huang, J., Lee, A. W., Motivala, A., Munir, A. Q., Pelley, S., Povinec, P., Rahn, G., Triantafyllis, S., & Unterbrunner, P. (2016). The Snowflake elastic data warehouse. In *Proceedings of the 2016 International Conference on Management of Data* (pp. 215–226). Association for Computing Machinery. https://doi.org/10.1145/2882903.2903741

Ferreira, N., Martins, P., & Furtado, P. (2013). Near real-time with traditional data warehouse architectures: Factors and how-to. In *Proceedings of the 17th International Database Engineering & Applications Symposium* (pp. 68–75). Association for Computing Machinery. https://doi.org/10.1145/2513591.2513650

Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Pellow, F., & Pirahesh, H. (1996). Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *Proceedings of the Twelfth International Conference on Data Engineering* (pp. 152–159). IEEE. https://doi.org/10.1109/ICDE.1996.492099

Im, J.-F., Gopalakrishna, K., Subramaniam, S., Shrivastava, M., Tumbde, A., Jiang, X., Dai, J., Lee, S., Pawar, N., Li, J., & Aringunram, R. (2018). Pinot: Realtime OLAP for 530 million users. In *Proceedings of the 2018 International Conference on Management of Data* (pp. 583–594). Association for Computing Machinery. https://doi.org/10.1145/3183713.3190661

Inmon, W. H. (2002). *Building the data warehouse* (3rd ed.). Wiley.

Karpathiotakis, M., Özcan, F., Floratou, A., & Ailamaki, A. (2017). No data left behind: Real-time insights from a complex data ecosystem. In *Proceedings of the 2017 Symposium on Cloud Computing (*pp. 108–120). Association for Computing Machinery. https://doi.org/10.1145/3127479.3131208

Kimball, R., & Ross, M. (2013). *The data warehouse toolkit: The definitive guide to dimensional modeling* (3rd ed.). Wiley.

Kitchenham, B., & Charters, S. (2007). *Guidelines for performing systematic literature reviews in software engineering* (EBSE Technical Report EBSE-2007-01). Keele

University and Durham University Joint Report. https://legacyfileshare.elsevier.com/promis_misc/525444systematicreviewsguide.pdf

Liu, X., & Iftikhar, N. (2015). An ETL optimization framework using partitioning and parallelization. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing* (pp. 1015–1022). Association for Computing Machinery. https://doi.org/10.1145/2695664.2695846

Milkai, E., Yu, X., & Patel, J. M. (2025). Hermes: Off-the-shelf real-time transactional analytics. *Proceedings of the VLDB Endowment, 18*(8), 2334–2347. https://doi.org/10.14778/3742728.3742731

Müller, M., & Pfahl, D. (2008). Simulation methods. In F. Shull, J. Singer, & D. I. K. Sjøberg (Eds.), *Guide to advanced empirical software engineering* (pp. 117–152). Springer. https://doi.org/10.1007/978-1-84800-044-5_5

Page, M. J., McKenzie, J. E., Bossuyt, P. M., Boutron, I., Hoffmann, T. C., Mulrow, C. D., Shamseer, L., Tetzlaff, J. M., Akl, E. A., Brennan, S. E., Chou, R., Glanville, J., Grimshaw, J. M., Hróbjartsson, A., Lalu, M. M., Li, T., Loder, E. W., Mayo-Wilson, E., McDonald, S., ... Moher, D. (2021). The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. *BMJ, 372*, n71. https://doi.org/10.1136/bmj.n71

Perera, S., & Suhothayan, S. (2015). Solution patterns for realtime streaming analytics. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems* (pp. 247–255). Association for Computing Machinery. https://doi.org/10.1145/2675743.2774214

Qu, J., & Wang, J. (2024). Real-time data warehousing in the big data environment: A comprehensive review of implementation in the internet industry. *Applied and Computational Engineering, 88*, 110–119. https://doi.org/10.54254/2755-2721/88/20241643

Ramanathan, C., & Venkatesh, B. (2024). STRCUBE—An architecture and reference implementation for streaming star schema. In *Proceedings of the 7th Joint International Conference on Data Science & Management of Data* (pp. 533–537). Association for Computing Machinery. https://doi.org/10.1145/3632410.3632479

Ramnarayan, J., Mozafari, B., Wale, S., Menon, S., Kumar, N., Bhanawat, H., Chakraborty, S., Mahajan, Y., Mishra, R., & Bachhav, K. (2016). SnappyData: A hybrid transactional analytical store built on Spark. In *Proceedings of the 2016 International*

Conference on Management of Data (pp. 2153–2156). Association for Computing Machinery. https://doi.org/10.1145/2882903.2899408

Sanla, A., & Numnonda, T. (2019). A comparative performance of real-time big data analytic architectures. In *2019 IEEE 9th International Conference on Electronics Information and Emergency Communication* (pp. 1–5). IEEE. https://doi.org/10.1109/ICEIEC.2019.8784580

Sargent, R. G. (2013). Verification and validation of simulation models. *Journal of Simulation, 7*, 12–24. https://doi.org/10.1057/jos.2012.20

Shoshani, A. (1997). OLAP and statistical databases: Similarities and differences. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (pp. 185–196). Association for Computing Machinery. https://doi.org/10.1145/263661.263682

Silva, B., Moreira, J., & Costa, R. L. de C. (2023). Logical big data integration and near real-time data analytics. *Data & Knowledge Engineering, 146*, 102185. https://doi.org/10.1016/j.datak.2023.102185

Siripurapu, D. K. (2025). Real-time data analytics integration: A technical deep dive. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 11*(1), 2484–2492. https://doi.org/10.32628/CSEIT251112253

Vassiliadis, P., & Sellis, T. (1999). A survey of logical models for OLAP databases. *ACM SIGMOD Record, 28*(4), 64–69. https://doi.org/10.1145/344816.344869

Wohlin, C., & Aurum, A. (2015). Towards a decision-making structure for selecting a research design in empirical software engineering. *Empirical Software Engineering, 20*(6), 1427–1455. https://doi.org/10.1007/s10664-014-9319-7

Yang, F., Merlino, G., Ray, N., Léauté, X., Gupta, H., & Tschetter, E. (2017). The RADStack: Open source lambda architecture for interactive analytics. In *Proceedings of the 50th Hawaii International Conference on System Sciences* (pp. 1703–1712). AIS Electronic Library. https://doi.org/10.24251/HICSS.2017.206

Zhang, F., Wu, M., Xu, C., Bao, Y., Qiao, J., Zhou, Y., Fan, H., Yin, C., Zhou, W., & Li, F. (2025). Streaming View: An efficient data processing engine for modern real-time data warehouse of Alibaba Cloud. *Proceedings of the VLDB Endowment, 18*(12), 5153–5165. https://doi.org/10.14778/3750601.3750634

Appendix A: PRISMA checklist

Section and Topic	Item #	Checklist item	Location where item is reported
TITLE			
Title	1	Identify the report as a systematic review.	Section 4
ABSTRACT			
Abstract	2	See the PRISMA 2020 for Abstracts checklist.	Abstract
INTRODUCTION			
Rationale	3	Describe the rationale for the review in the context of existing knowledge.	Section 1
Objectives	4	Provide an explicit statement of the objective(s) or question(s) the review addresses.	Section 1.1
METHODS			
Eligibility criteria	5	Specify the inclusion and exclusion criteria for the review and how studies were grouped for the syntheses.	Section 4.2
Information sources	6	Specify all databases, registers, websites, organisations, reference lists and other sources searched or consulted to identify studies. Specify the date when each source was last searched or consulted.	Section 4.1
Search strategy	7	Present the full search strategies for all databases, registers and websites, including any filters and limits used.	Appendix B
Selection process	8	Specify the methods used to decide whether a study met the inclusion criteria of the review, including how many reviewers screened each record and each report retrieved, whether they worked independently, and if applicable, details of automation tools used in the process.	Section 4.2
Data collection process	9	Specify the methods used to collect data from reports, including how many reviewers collected data from each report, whether they worked independently, any processes for obtaining or confirming data from study investigators, and if applicable, details of automation tools used in the process.	Section 4.3
Data items	10a	List and define all outcomes for which data were sought. Specify whether all results that were compatible with each outcome domain in each study were sought (e.g. for all measures, time points, analyses), and if not, the methods used to decide which results to collect.	Appendix C
	10b	List and define all other variables for which data were sought (e.g. participant and intervention characteristics, funding sources). Describe any	Appendix C

		assumptions made about any missing or unclear information.	
Study risk of bias assessment	11	Specify the methods used to assess risk of bias in the included studies, including details of the tool(s) used, how many reviewers assessed each study and whether they worked independently, and if applicable, details of automation tools used in the process.	Not assessed. The review synthesized heterogenous studies. No formal risk-of-bias tool was applied
Effect measures	12	Specify for each outcome the effect measure(s) (e.g. risk ratio, mean difference) used in the synthesis or presentation of results.	No quantitative effect measures were used
Synthesis methods	13a	Describe the processes used to decide which studies were eligible for each synthesis (e.g. tabulating the study intervention characteristics and comparing against the planned groups for each synthesis (item #5)).	Section 4.3
	13b	Describe any methods required to prepare the data for presentation or synthesis, such as handling of missing summary statistics, or data conversions.	Section 4.3
	13c	Describe any methods used to tabulate or visually display results of individual studies and syntheses.	Section 4.3
	13d	Describe any methods used to synthesize results and provide a rationale for the choice(s). If meta-analysis was performed, describe the model(s), method(s) to identify the presence and extent of statistical heterogeneity, and software package(s) used.	Section 4.3
	13e	Describe any methods used to explore possible causes of heterogeneity among study results (e.g. subgroup analysis, meta-regression).	Section 4.3
	13f	Describe any sensitivity analyses conducted to assess robustness of the synthesized results.	No sensitivity analyses were conducted
Reporting bias assessment	14	Describe any methods used to assess risk of bias due to missing results in a synthesis (arising from reporting biases).	Not assessed; reporting bias was not evaluated because the review did not perform a quantitative effect synthesis
Certainty assessment	15	Describe any methods used to assess certainty (or confidence) in the body of evidence for an outcome.	Not assessed; no formal certainty-of-evidence framework was applied because the review was interpretive and non-effect-based
RESULTS			
Study selection	16a	Describe the results of the search and selection process, from the number of records identified in the search to the number of studies included in the review, ideally using a flow diagram.	Figure 2
	16b	Cite studies that might appear to meet the inclusion criteria, but which were excluded, and explain why they were excluded.	Figure 2
Study characteristics	17	Cite each included study and present its characteristics.	Appendix C
Risk of bias in studies	18	Present assessments of risk of bias for each included study.	Not Applicable

Results of individual studies	19	For all outcomes, present, for each study: (a) summary statistics for each group (where appropriate) and (b) an effect estimate and its precision (e.g. confidence/credible interval), ideally using structured tables or plots.	Appendix C
Results of syntheses	20a	For each synthesis, briefly summarise the characteristics and risk of bias among contributing studies.	Section 4
	20b	Present results of all statistical syntheses conducted. If meta-analysis was done, present for each the summary estimate and its precision (e.g. confidence/credible interval) and measures of statistical heterogeneity. If comparing groups, describe the direction of the effect.	Not Applicable
	20c	Present results of all investigations of possible causes of heterogeneity among study results.	Section 4.3
	20d	Present results of all sensitivity analyses conducted to assess the robustness of the synthesized results.	Not Applicable
Reporting biases	21	Present assessments of risk of bias due to missing results (arising from reporting biases) for each synthesis assessed.	Not assessed. Reporting bias was not evaluated because the review did not perform a quantitative effect synthesis
Certainty of evidence	22	Present assessments of certainty (or confidence) in the body of evidence for each outcome assessed.	Not assessed. No formal certainty-of-evidence framework was applied because the review was interpretive
DISCUSSION			
Discussion	23a	Provide a general interpretation of the results in the context of other evidence.	Section 6
	23b	Discuss any limitations of the evidence included in the review.	Section 6.3
	23c	Discuss any limitations of the review processes used.	Section 6.3
	23d	Discuss implications of the results for practice, policy, and future research.	Section 6
OTHER INFORMATION			
Registration and protocol	24a	Provide registration information for the review, including register name and registration number, or state that the review was not registered.	Not registered
	24b	Indicate where the review protocol can be accessed, or state that a protocol was not prepared.	No publicly accessible protocol was prepared / available
	24c	Describe and explain any amendments to information provided at registration or in the protocol.	Not applicable; no protocol was registered/prepared
Support	25	Describe sources of financial or non-financial support for the review, and the role of the funders or sponsors in the review.	No specific funding/support for the review
Competing interests	26	Declare any competing interests of review authors.	The author declares no competing interests

Availability of data, code and other materials	27	Report which of the following are publicly available and where they can be found: template data collection forms; data extracted from included studies; data used for all analyses; analytic code; any other materials used in the review.	Appendices B and C

Appendix B: Search Queries per Database

Scopus

Search query
TITLE-ABS-KEY (("real-time" OR "real time" OR "low-latency" OR "low latency" OR "event-driven" OR "event driven" OR "micro-batch*" OR "microbatch*" OR "operational analytic*" OR "streaming analytic*" OR "streaming data*" OR "incremental load" OR "change data capture") AND ("dimensional model*" OR "multidimensional model*" OR "multi-dimensional model*" OR "tabular model*" OR "cube model*" OR "denormalized model*" OR "dimensional schema" OR "star schema" OR "galaxy schema" OR "fact constellation" OR "snowflake schema" OR "semantic layer" OR "lambda" OR "kappa" OR "medallion architecture") AND ("business intelligence" OR "data analytics" OR "business analytics" OR "operational analytics" OR "analytics layer" OR "metrics layer" OR "enterprise analytics" OR "data warehouse"))

Web of Science

Search query
TS= (("real-time" OR "real time" OR "low-latency" OR "low latency" OR "event-driven" OR "event driven" OR "micro-batch*" OR "microbatch*" OR "operational analytic*" OR "streaming analytic*" OR "streaming data*" OR "incremental load" OR "change data capture") AND ("dimensional model*" OR "multidimensional model*" OR "multi-dimensional model*" OR "tabular model*" OR "cube model*" OR "denormalized model*" OR "dimensional schema" OR "star schema" OR "galaxy schema" OR "fact constellation" OR "snowflake schema" OR "semantic layer" OR "lambda architecture" OR "kappa architecture" OR "medallion architecture") AND ("business intelligence" OR "data analytics" OR "business analytics" OR "operational analytics" OR "analytics layer" OR "metrics layer" OR "enterprise analytics" OR "data warehouse"))

IEEE Xplore

Due to limitations in IEEE Xplore advanced search, the total count of wildcard characters used in the query was limited to 10. Wildcards used in search terms “multidimensional model*” and “multi-dimensional model” were truncated. Due to a large number of conference papers, the scope was narrowed down to only include journal articles.

Search query
(("real-time" OR "real time"   OR "low-latency" OR "low latency"   OR "event-driven" OR "event driven"   OR "micro-batch*" OR "microbatch*"   OR "operational analytic*"   OR "streaming analytic*"   OR "streaming data*"   OR "incremental load"   OR "change data capture"   )   AND   ("dimensional model*"   OR "multidimensional model" OR "multi-dimensional model"   OR "tabular model*"   OR "cube model*"   OR "denormalized model*"   OR "dimensional schema"   OR "star schema"   OR "galaxy schema" OR "fact constellation"   OR "snowflake schema"   OR "semantic layer"   OR "lambda architecture"   OR "kappa architecture"   OR "medallion architecture")   AND   ("business intelligence"   OR "data analytics"   OR "business analytics"   OR "operational analytics"   OR "analytics layer"   OR "metrics layer"   OR "enterprise analytics"   OR "data warehouse"   ))

ACM Digital Library

Due to limitations in ACM Digital Library's metadata, the search scope was exceptionally set to include matches from all available fields. In contrast, the scope was narrowed down to only include journal articles. To improve precision while preserving conceptual coverage, generic terms such as lambda and streaming were restricted to analytics-specific phrases, and business analytics terminology was used as a contextual gatekeeper.

Search query
(("real-time" OR "real time" OR "low-latency" OR "low latency" OR "event-driven" OR "event driven" OR "micro-batch*" OR "microbatch*" OR "operational analytic*" OR "streaming analytic*" OR "streaming data*" OR "incremental load" OR "change data capture") AND ("dimensional model*" OR "multidimensional model*" OR "multi-dimensional model*" OR "tabular model*" OR "cube model*" OR "denormalized model*" OR "dimensional schema" OR "star schema" OR "galaxy schema" OR "fact constellation" OR "snowflake schema" OR "semantic layer" OR "lambda architecture" OR "kappa architecture" OR "medallion architecture") AND ("business intelligence" OR "data analytics" OR "business analytics" OR "operational analytics" OR "analytics layer" OR "metrics layer" OR "enterprise analytics" OR "data warehouse"))

AIS eLibrary

Search query
(("real-time" OR "real time" OR "low-latency" OR "low latency" OR "event-driven" OR "event driven" OR "micro-batch*" OR "microbatch*" OR "operational analytic*" OR "streaming analytic*" OR "streaming data*" OR "incremental load" OR "change data capture") AND ("dimensional model*" OR "multidimensional model*" OR "multi-dimensional model*" OR "tabular model*" OR "cube model*" OR "denormalized model*" OR "dimensional schema" OR "star schema" OR "galaxy schema" OR "fact constellation" OR "snowflake schema" OR "semantic layer" OR "lambda" OR "kappa" OR "medallion architecture") AND ("business intelligence" OR "data analytics" OR "business analytics" OR "operational analytics" OR "analytics layer" OR "metrics layer" OR "enterprise analytics" OR "data warehouse"))

Appendix C: Full data-extraction and coding matrix

ID	Study	Analytical truth	Temporal closure	History mutability	Semantic stabilization	Analytical abstraction	Semantic consistency
1	(Ferreira et al., 2013)	Retrospective snapshot	Micro-Batch closure	Fully recomputable	Batch ETL / preprocessing	Star schema	Deterministic and repeatable
2	(Liu & Iftikhar, 2015)	Retrospective snapshot	Batch closure	Fully recomputable	Batch ETL / preprocessing	Star schema	Deterministic and repeatable
3	(Perera & Suhothayan, 2015)	Evolving	No temporal closure	Append-only immutable	Streaming engine semantics	Event / log-based tables	Bounded inconsistency
4	(Ramnarayan et al., 2016)	Log-consistent transactional correctness	Micro-Batch closure	Log-based transactional revision	Transactional engine / logs	Relational analytical queries	Approximate / contract-based
5	(Karpathiotakis et al., 2017)	Evolving	No temporal closure	Source-driven mutable	Source-system semantics	Virtualized analytical views	Bounded inconsistency
6	(Yang et al., 2017)	Window-bound	Event-time windows	Append-only immutable	Engine-internal semantics	Engine-defined analytical structures	Bounded inconsistency
7	(Im et al., 2018)	Window-bound	Event-time windows	Append-only immutable	Engine-internal semantics	Engine-defined analytical structures	Bounded inconsistency
8	(Sanla & Numnonda, 2019)	Snapshot vs Evolving	Batch closure vs Streaming	Fully recomputable vs append-only	Batch ETL vs streaming engine	Not explicitly defined	Architecture-dependent
9	(Ben Aissa et al., 2020)	Multiple	Multiple	Multiple	Multiple	Multiple	Multiple
10	(Bhagat & Deshpande, 2023)	Retrospective snapshot	Batch closure	Fully recomputable	Batch ETL / preprocessing	Aggregated event tables	Deterministic and repeatable
11	(Silva et al., 2023)	Retrospective snapshot	Logical snapshot (no physical closure)	Fully recomputable	Business semantic layer	Star schema	Deterministic and repeatable
12	(Ramanathan & Venkatesh, 2024)	Window-bound	Event-time windows	Append-only immutable	Streaming MOLAP engine	Streaming star schema	Bounded inconsistency

ID	Study	Analytical truth	Temporal closure	History mutability	Semantic stabilization	Analytical abstraction	Semantic consistency
13	(Milkai et al., 2025)	Log-consistent transactional correctness	Commit snapshot	Log-based transactional revision	Transactional engine / logs	Relational analytical queries	Multiple
14	(Zhang et al., 2025)	Evolving	No temporal closure	Selective backfill	Incremental view maintenance	SQL views	Eventually consistent