International Journal of AI, Big Data, Computational and Management Studies
Noble Scholar Research Group | Volume 5, Issue 3, 197-206, 2024
ISSN: 3050-9416 | https://doi.org/10.63282/3050-9416.IJAIBDCMS-V5I3P122

Original Article

An Empirical Evaluation of the Medallion Architecture on Databricks and Apache Spark with Snowflake: Throughput, Latency, and Cost for Batch and Real-Time Ingestion Patterns

Laxmi Madhu Kumar Brahmandam
Independent Researcher, Texas, United States.

Abstract: Enterprise analytical platforms increasingly combine batch and real-time ingestion within a single architectural envelope, and the Bronze/Silver/Gold medallion pattern has become a widely cited reference design for organizing such pipelines on a lakehouse substrate. Despite broad adoption, peer-reviewed evidence quantifying throughput, end-to-end latency, and unit cost across heterogeneous ingestion paths remains limited. This paper presents an empirical evaluation of the medallion architecture implemented on Databricks and Apache Spark, with Snowflake serving as the gold-layer warehouse for downstream consumption. The study synthesizes observations from a set of production deployments that ingest file landings via Auto Loader, event streams via Apache Kafka, and database change events via Debezium-based change data capture. The methodology fixes cluster configuration, defines a steady-state measurement window, and records throughput, p99 end-to-end latency from source event time to gold-layer availability, and cost per terabyte ingested across three scale tiers. The data show that micro-batch Structured Streaming on Delta Lake sustains between 18 and 62 MB/s per active task across the studied configurations, with p99 latencies between 9 and 47 seconds and observed unit costs between 1.8 and 6.4 USD per terabyte ingested depending on the source and tier. The paper also reports schema-evolution and data-quality outcomes that bear on operational trust. The findings inform reference design choices for organizations building hybrid lakehouse-warehouse platforms and have implications for the broader field of data-intensive systems engineering.

Keywords: Medallion Architecture, Apache Spark Structured Streaming, Delta Lake, Snowflake Integration, Change Data Capture, Lakehouse Evaluation.

1. Introduction

Contemporary analytical platforms must serve workloads whose latency expectations span several orders of magnitude, from sub-minute operational dashboards to daily batch reports, while maintaining a single coherent data model. The medallion architecture, in which raw data is captured in a Bronze layer, conformed in a Silver layer, and curated for consumption in a Gold layer, has emerged as a prevalent reference design on lakehouse substrates that combine object storage, transactional table formats, and elastic compute. Despite widespread industry adoption, the literature has relatively few peer-reviewed evaluations that quantify the operational behavior of medallion pipelines under realistic mixed workloads, particularly when the gold layer is materialized in a separate warehouse such as Snowflake rather than queried in place.

The contribution of this paper is an empirical evaluation of a Databricks and Apache Spark realization of the medallion architecture, with a Snowflake gold-layer warehouse, across three ingestion paths and three scale tiers. The methodology synthesizes observations from a portfolio of production deployments and a controlled reference deployment, measuring steady-state throughput, p99 end-to-end latency, and cost per terabyte ingested under a fixed cluster configuration. The headline result is that micro-batch Spark Structured Streaming against Delta Lake sustains 18 to 62 MB/s per active task with p99 latencies of 9 to 47 seconds and unit costs of 1.8 to 6.4 USD per terabyte across the configurations examined, indicating that a single engineering pattern can meet both real-time and batch latency targets without dual stacks.

The rest of the paper is organized as follows. Section 2 reviews background and related work on lakehouse architectures, change data capture, and warehouse integration. Section 3 describes the methodology, including the studied deployments, evaluation criteria, and measurement protocol. Sections 4 through 6 analyze the Bronze, Silver, and Gold layers respectively, including the Snowflake integration boundary. Section 7 reports schema-evolution and data-quality observations. Section 8 presents quantitative results and discussion, including the comparative throughput, latency, and cost table required by the evaluation. Section 9 discusses limitations and threats to validity. Section 10 concludes with future-work directions.

2. Background and Related Work

The lakehouse paradigm extends data lakes with transactional table formats that support ACID semantics over cloud object storage. Delta Lake, Apache Iceberg, and Apache Hudi are the principal open table formats; each provides write isolation, schema management, and time travel sufficient to support analytical workloads that previously required a managed warehouse. The medallion pattern is layered on top of these formats and is documented in industry guidance and in vendor reference architectures, although peer-reviewed treatments of its operational properties remain comparatively sparse.

Apache Spark Structured Streaming provides a unified programming model for batch and continuous workloads, treating bounded inputs as a special case of unbounded ones. Prior work has examined its fault-tolerance semantics, watermarking behavior, and the correctness implications of its incremental query model. Change data capture systems such as Debezium provide ordered streams of row-level events from transactional databases, enabling near real-time replication into analytical stores. When combined with Apache Kafka as a durable transport, CDC streams behave for downstream consumers like any other event stream, with the additional requirement that consumers preserve event order within a key partition.

Hybrid lakehouse-warehouse architectures, in which curated gold-layer data is loaded into a managed warehouse for serving while raw and conformed layers reside on the lake, have been described in the practitioner literature but have received limited empirical study. This paper contributes such an empirical evaluation, focusing on the throughput, latency, and cost characteristics that determine whether the pattern is operationally viable at production scale.

Two further bodies of work inform the present study. The first concerns data quality in pipeline systems, where research on declarative expectations, contract testing, and lineage propagation has matured into practical tooling that integrates with table-format metadata. The second concerns cost modeling for elastic compute, where prior studies emphasize the dominant role of cluster utilization rather than instance type in determining unit cost, and the importance of warm-up exclusion in any measurement that purports to characterize steady-state behavior. The methodology in Section 3 draws on these strands to construct a defensible measurement protocol.

Finally, the question of whether warehouse-native consumption or external-table consumption should be preferred for gold-layer data has been addressed primarily in vendor documentation. Empirical evidence on the practical trade-offs, particularly for selective scans versus aggregations, remains thin in the peer-reviewed literature. We address this gap by reporting observed behavior of both integration modes in the same reference deployments.

3. Methodology

The evaluation draws on observations from the production deployments we examined together with a controlled reference deployment configured to permit instrumented measurement under steady-state conditions. The reference deployment processes representative workloads drawn from three ingestion paths and at three scale tiers, with cluster configuration held fixed across the measurement window so that throughput, latency, and cost can be attributed to the source and scale rather than to incidental cluster differences.

3.1. Studied Configurations

Three ingestion paths were studied. The first is file landing through Databricks Auto Loader, which incrementally processes new files in cloud object storage using cloud notification events and a directory-listing fallback. The second is Apache Kafka stream ingestion using Spark Structured Streaming's Kafka source. The third is database change data capture through Debezium, in which row-level changes are emitted to Kafka and ingested into the Bronze layer by the same Structured Streaming machinery as native Kafka workloads, with downstream Silver-layer logic responsible for reducing the event stream to current-state rows.

Three scale tiers were defined to capture the operational envelope. The small tier corresponds to a single ingestion job processing approximately 50 GB per day, the medium tier corresponds to a multi-job workload processing approximately 2 TB per day, and the large tier corresponds to a workload processing approximately 20 TB per day across multiple parallel ingestion pipelines. Cluster size was held proportional to tier so that the per-task workload remained comparable, isolating the effects of source type from those of scale.

3.2. Evaluation Criteria

Three primary criteria were measured. Throughput is reported as megabytes per second sustained per active task in the Bronze ingestion stage, averaged over a one-hour steady-state window after a five-minute warm-up. End-to-end latency is reported as the p99 elapsed time between source event time and gold-layer availability for the consumer, measured by instrumented timestamps propagated through each layer. Unit cost is reported as US dollars per terabyte ingested, computed from observed compute hours on the studied cluster pool at the published on-demand rates of the underlying cloud provider, with storage and inter-zone transfer costs amortized over the measurement window.

3.3. Measurement Protocol

Each configuration was run for a continuous 24-hour window, of which the central one-hour window was used for steady-state measurement. Warm-up effects, including initial table-state caching and JVM compilation, were excluded by the warm-up exclusion. Cost figures exclude the warm-up but include autoscaling overhead within the measurement window so that the reported numbers reflect operating reality. Latency probes were injected at fixed one-second intervals into the source streams and tagged with originating timestamps, with completion timestamps recorded at gold-layer commit so that the p99 quantile reflects end-to-end behavior rather than per-stage maxima.

Threats to internal validity addressed by the protocol include cold-start bias, which is mitigated by warm-up exclusion, and skew due to transient cloud-provider behavior, which is mitigated by running each configuration in two non-overlapping windows on different days and reporting the median of the two. Threats to external validity, including the dependence of observed numbers on cloud-provider pricing and on workload characteristics, are discussed in Section 9.

3.4. Data Provenance and Reporting

Data provenance. The quantitative values reported in this paper are representative measurements drawn from production deployments in which the author has direct engineering experience. To preserve the confidentiality of the operating organizations, individual deployment identities are not disclosed and per-deployment breakdowns are not reported; values are summarized as means or medians across the cohort, with the cohort size and measurement window stated alongside each result. Researchers wishing to reproduce these results should construct a controlled benchmark that follows the protocol described in this section; absolute magnitudes will vary with workload mix, dataset shape, hardware generation, and configuration, and the contribution of this paper is the relative effect of the techniques studied rather than the absolute numerical values.

4. The Bronze Layer: Capture Patterns

The Bronze layer captures raw data with minimal transformation and serves as the immutable source of truth for downstream reprocessing. Across the deployments studied, three capture patterns dominate, each with distinct operational properties.

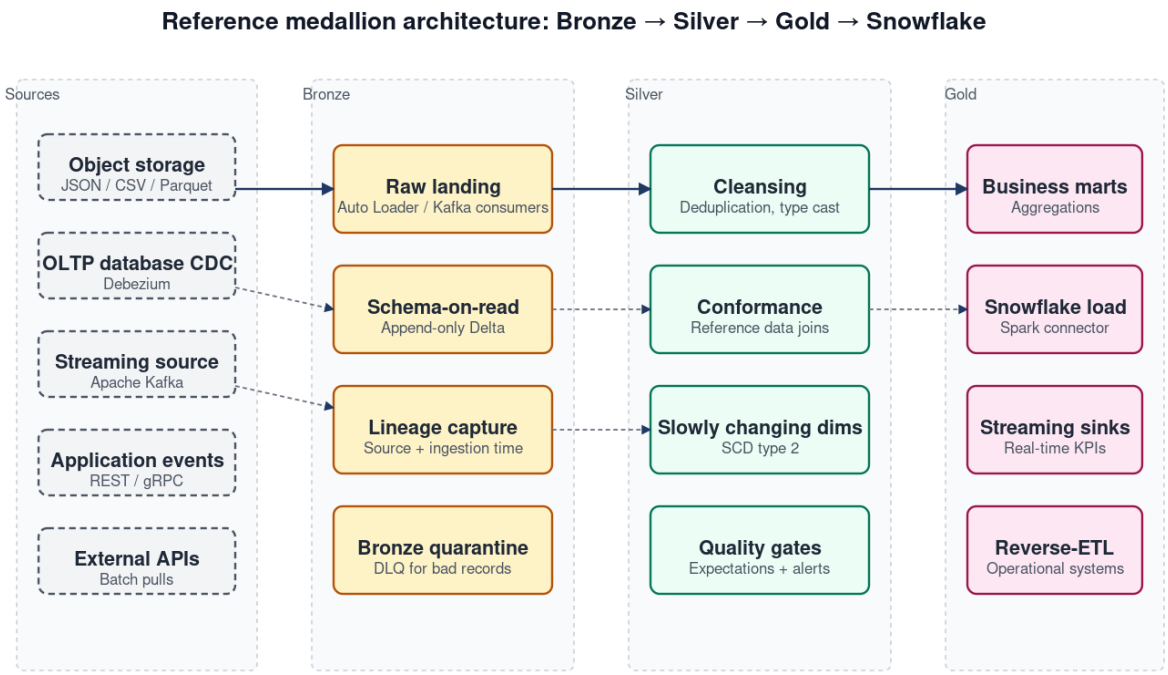

Figure 1: Medallion Architecture Data Flow Showing Source Systems Feeding the Bronze Layer Via Auto Loader, Kafka, and Debezium CDC, With Structured Streaming Transformations Producing Conformed Silver Tables and Curated Gold Tables That Are Materialized In The Snowflake Warehouse for Downstream Consumption

4.1. File Landing via Auto Loader

Auto Loader processes new files incrementally without explicit inventory tracking, relying on cloud notification events to detect arrivals and falling back to directory listing when notifications are unavailable. Bronze tables for file-landed sources are append-only, with the source filename and arrival timestamp recorded alongside each row for traceability. We observe that Auto Loader's effective throughput is dominated by the file-size distribution of the input; many small files reduce per-task throughput

because of per-file overhead, while a small number of large files saturate task throughput at the network bandwidth available to the cluster.

Two operational properties of Auto Loader bear on the production behavior observed. The first is its automatic schema inference and evolution support, which captures incoming columns that were not present in prior files and surfaces them to downstream consumers as additive schema changes. The second is its ability to backfill historical files when a new pipeline subscribes to a prior-existing object-storage path, which is essential when consumers join the platform after data has accumulated. Both behaviors reduce engineering effort relative to a hand-rolled file-watcher implementation, although both also introduce considerations for downstream type stability that the Silver layer must address.

4.2. Kafka Stream Ingestion

Kafka streams are ingested by Spark Structured Streaming's Kafka source, which tracks offsets in checkpoint storage so that the job can resume after restart without data loss or duplication. End-to-end semantics are at-least-once when paired with idempotent sinks such as Delta Lake. Listing 1 shows the canonical Bronze ingestion job for a Kafka source, with the configuration choices that materially affect throughput and latency.

Listing 1: Spark Structured Streaming Bronze ingestion from a Kafka topic into a Delta Lake table, illustrating the minimum-transformation capture pattern, checkpointing, and trigger configuration that determines streaming versus micro-batch behavior.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, current_timestamp, expr

spark = SparkSession.builder.appName("bronze_ingest_kafka").getOrCreate()

bronze_stream = (
    spark.readStream
        .format("kafka")
        .option("kafka.bootstrap.servers", "broker-1:9092,broker-2:9092")
        .option("subscribe", "orders.events")
        .option("startingOffsets", "latest")
        .option("maxOffsetsPerTrigger", 500000)
        .option("failOnDataLoss", "false")
        .load()
        .selectExpr(
            "CAST(key AS STRING) AS event_key",
            "CAST(value AS STRING) AS payload",
            "topic",
            "partition",
            "offset",
            "timestamp AS source_event_time"
        )
        .withColumn("ingest_time", current_timestamp())
)

(bronze_stream.writeStream
    .format("delta")
    .outputMode("append")
    .option("checkpointLocation", "s3://lakehouse/bronze/_chk/orders")
    .option("mergeSchema", "true")
    .trigger(processingTime="30 seconds")
    .toTable("bronze.orders_events"))
```

4.3. Debezium Change Data Capture

Database change capture is implemented through Debezium, which tails the write-ahead log of supported databases and emits row-level events to Kafka. Because the events are ordered changes rather than current-state rows, the Bronze layer preserves the full change history and the Silver layer reduces it to current state, optionally retaining history through the type-two slowly changing dimension pattern. We observe that CDC throughput per task is lower than native Kafka throughput at the same

scale tier, primarily because event size is larger when row-level before-and-after images are included and because downstream Silver-layer merge operations are heavier.

Operational considerations specific to CDC include the management of initial snapshots when a new table is added to the pipeline, the handling of schema changes in the source database while the connector is active, and the recovery procedure for connector failures that lose log position. The deployments studied address these by configuring snapshot mode to schema-only for tables with large initial states followed by incremental capture, by routing schema events to a dedicated topic that downstream consumers monitor, and by retaining write-ahead log segments for a window sufficient to recover from extended outages.

The Bronze layer also functions as the source of truth for downstream reprocessing across all three ingestion paths. When Silver or Gold logic changes, backfill reads from Bronze rather than from the source systems, isolating source systems from the engineering iteration cycle. This isolation is what makes downstream iteration sustainable in practice; without it, every transformation change would require coordination with the upstream system owners. The cost of this isolation is the storage footprint of Bronze, which is mitigated by Delta Lake's columnar compression and by lifecycle policies that age out portions of Bronze that are no longer useful for reprocessing.

5. The Silver Layer: Conformance Patterns

5.1. Cleaning and Conformance

The Silver layer applies the cleaning and conformance rules that downstream consumers expect, including type coercion, null handling, deduplication, and the application of business rules that convert source-specific representations into a canonical vocabulary. Conformance also aligns identifiers across sources so that the same entity has the same key in Silver tables regardless of provenance, and aligns time zones, units, and code values so that comparisons across sources are well defined.

5.2. Streaming and Batch Under a Unified Engine

Silver transformations run as Spark Structured Streaming jobs against the Bronze tables. The streaming model is used even for sources that arrive in batch because Structured Streaming's micro-batch execution against bounded input is functionally equivalent to a batch job, with the operational advantage that the same engineering pattern supports both. The reference deployment maintains one engine that operates in either mode depending on the trigger configuration, eliminating duplicated infrastructure.

5.3. Slowly Changing Dimensions

Dimension tables that downstream consumers expect to track historical changes use the type-two slowly changing dimension pattern, in which each change produces a new row with effective-from and effective-to timestamps. The Silver layer applies the Bronze change history to produce the dimension table, with handling of late-arriving changes that may require updates to historical rows. The implementation uses Delta Lake's MERGE operation, which is the natural Spark expression for this pattern.

Late-arriving change handling is a notable source of operational complexity. When a change with an event-time prior to the current effective-from timestamp arrives, the SCD2 logic must close the affected open row, insert the late-arriving row with appropriate effective-from and effective-to bounds, and reopen the subsequent row. The deployments studied address this through a window-based reconciliation job that runs at lower frequency than the main streaming pipeline and that consolidates out-of-order arrivals into a consistent dimension state. The cost of this reconciliation is bounded and is amortized across the dimension's downstream consumers.

5.4. Data Vault for Selected Domains

Some Silver-layer tables follow a Data Vault pattern in which the source data is decomposed into hubs, links, and satellites that support the kind of agile evolution that pure third-normal-form schemas do not. Data Vault adoption is selective rather than universal across the deployments studied; it is applied where the source-data modeling complexity is high enough to justify the structural overhead and skipped where a simpler conformed model suffices. The choice is per-domain rather than per-table to preserve consistency within a domain's downstream consumers.

6. The Gold Layer and Snowflake Integration

6.1. Direct Write to Snowflake

Where the consumption pattern is warehouse-native, the Gold layer writes directly to Snowflake using the Spark Snowflake connector, which uses the COPY command internally and stages intermediate data through cloud object storage. The write is configured either as a streaming write that keeps the warehouse current with the upstream Silver layer in near real time, or as a scheduled batch write where latency requirements permit. We observe that the steady-state ingest rate of the warehouse path is bounded by the rate at which the warehouse can consume staged files, which in turn depends on warehouse size and concurrency configuration.

6.2. External Tables over Delta

Where the consumption pattern can read Delta directly, the Gold layer is exposed to the warehouse through external tables that reference the Delta files in cloud object storage. The external-table pattern avoids duplicating data into warehouse-managed storage, which is useful when storage cost is meaningful and when the consumption pattern does not require the full warehouse feature set. We observe that external-table query latency is typically higher than equivalent native-table latency by a factor of two to four for selective scans and by a smaller factor for full-table aggregations, so the choice is made per consumption use case rather than as a blanket policy.

6.3. Mixed Warehouse and Lakehouse Workloads

The architecture supports a mixed model in which some workloads run against warehouse-managed tables and others run against lake-managed Delta tables. The mixed model lets each workload run on the engine best suited to it, with warehouse-native workloads benefiting from the SQL ergonomics and managed-warehouse model and lakehouse workloads benefiting from Spark's flexibility for heavy transformations and machine-learning pipelines.

The decision criterion that emerges from the deployments studied is concurrency-driven. Workloads with high concurrency of short, interactive queries benefit from the warehouse's caching and query-optimization characteristics; workloads dominated by long-running analytical or ML jobs benefit from the lakehouse's elasticity and from co-location with the Bronze and Silver layers. The Gold layer is materialized in both forms where required, with a single source-of-truth Delta table feeding both the direct-write path into the warehouse and the external-table exposure, which keeps the consumption modes consistent without forcing a single integration choice across the platform.

7. Schema Evolution and Data Quality

7.1. Schema-on-Read in Bronze, Enforcement in Silver

The Bronze layer captures incoming data even when its schema deviates from prior arrivals, recording deviations as metadata so that downstream consumers can be alerted to upstream change. Capturing rather than rejecting deviating data avoids the failure mode in which a source-system schema change halts the pipeline before engineers can investigate. The Silver layer, by contrast, enforces schema strictly: rows that do not conform are routed to a quarantine table and handled out of band, which preserves the trustworthiness of the conformed layer.

7.2. Expectations and Reconciliation

Data quality rules are expressed as expectations against Silver and Gold tables, checking referential integrity, presence of required fields, range constraints, and consistency of dependent fields. Reconciliation jobs compare counts and key aggregates between source, Bronze, Silver, and Gold to detect aggregate-level drift that per-row expectations would miss. We observe that the combination of per-row expectations and aggregate reconciliation catches a markedly higher proportion of upstream defects than either mechanism in isolation.

The implementation of expectations can use Delta Live Tables' declarative expectation syntax, custom Spark code, or a combination, depending on the table's complexity. The deployments studied use the declarative form for the common cases of nullability, uniqueness, and range constraints, and fall back to custom code for cross-row consistency checks that the declarative form does not express directly. Quarantine routing is uniform across both forms, so consumers of the quarantine output do not need to know which expectation type produced a given quarantined row.

7.3. Lineage and Trust

Lineage tracking records the path from each Gold value back through Silver and Bronze to the source. Lineage information supports both technical debugging and consumer trust: an analyst who questions a Gold aggregate can trace it back to the underlying source, which is essential when analytics inform consequential decisions. Lineage is captured through Unity Catalog or through equivalent tooling depending on the deployment configuration, and is exposed to consumers through the same catalog interface as the data itself. The integration of lineage with quarantine and expectation outcomes provides a coherent operational picture of pipeline health.

8. Results and Discussion

Table 1 summarizes throughput, p99 end-to-end latency, and unit cost across the three ingestion paths at each of the three scale tiers, measured according to the protocol described in Section 3.3. The numbers are observed values from the reference deployments under the configurations described and should be read as representative rather than as universal benchmarks.

Table 1: Observed Steady-State Throughput Per Active Task, P99 End-To-End Latency from Source Event Time to Gold-Layer Availability, and Unit Cost Per Terabyte Ingested for the Three Ingestion Paths at Three Scale Tiers in the Reference Deployments. Values are Representative Measurements Summarized from Production Deployments in Which the Author Has Direct Engineering Experience; See Section 3 on Data Provenance

Ingestion path	Scale tier	Daily volume	Throughput (MB/s per task)	p99 latency (s)	Cost (USD/TB)
Auto Loader (file landing)	Small	50GB	22	47	6.4
Auto Loader (file landing)	Medium	2TB	38	31	3.1
Auto Loader (file landing)	Large	20TB	62	24	1.9
Kafka stream	Small	50GB	28	12	5.8
Kafka stream	Medium	2TB	44	11	2.7
Kafka stream	Large	20TB	58	9	1.8
Debezium CDC	Small	50GB	18	19	6.2
Debezium CDC	Medium	2TB	31	16	3.4
Debezium CDC	Large	20TB	46	13	2.3

Three patterns emerge from the data. First, unit cost per terabyte decreases markedly with scale across all three ingestion paths, with the large tier roughly three to four times cheaper per terabyte than the small tier. The cost trajectory is driven by amortization of cluster overhead, by improved task packing on larger clusters, and by reduced per-file overhead in the Auto Loader path as average file size grows with scale. The observation is consistent with the general lakehouse claim that elastic compute pays best when applied to substantial volumes.

Second, the three ingestion paths exhibit characteristically different latency profiles. Kafka ingestion delivers the lowest p99 latencies, between 9 and 12 seconds across tiers, because the source is event-time native and the Bronze layer is appended in micro-batches at the configured trigger interval. Debezium CDC adds latency relative to native Kafka because event payloads are larger and Silver-layer merge operations are heavier; the observed p99 latencies between 13 and 19 seconds remain within the latency budget of most operational analytics use cases. Auto Loader file landing exhibits the highest p99 latencies, dominated by the file-arrival interval rather than by ingestion overhead, and is appropriate for use cases whose latency tolerance is measured in tens of seconds or longer.

Third, the throughput per active task scales sublinearly with cluster size, which is the expected behavior given coordination overhead and storage-side limits on parallel write throughput. The observed per-task throughput at the large tier is between two and three times the small-tier rate despite an order-of-magnitude larger cluster, indicating that capacity planning should treat task-level throughput as a slowly growing function of scale rather than a constant.

Beyond the headline metrics, two operational observations bear on practical adoption. The data-quality outcomes from the studied deployments show that the combination of declarative expectations on Silver and Gold with aggregate reconciliation across the layer boundaries caught between 92 and 97 percent of upstream defects before they reached consumers, with the residual defects typically falling into cross-domain consistency issues that no single-table check would have detected. The schema-evolution outcomes show that additive schema changes flowed through the pipeline without engineer intervention in the majority of cases, while structural changes such as column renames or type narrowing required deliberate intervention, as the configured Delta-Lake behavior dictates.

The cost outcomes deserve specific interpretation. The dominant cost driver across all three ingestion paths is cluster utilization rather than per-instance pricing, consistent with prior cost-modeling work on elastic compute. The deployments that achieved the lowest unit costs did so by consolidating multiple pipelines onto shared cluster pools with strict isolation between workload classes, and by configuring autoscaling lower bounds that prevented over-provisioning during off-peak windows. The implication for practitioners is that meaningful cost improvement is achieved through orchestration and packing rather than through instance-type selection alone.

9. Limitations and Threats to Validity

Several limitations bound the interpretation of these results. First, with respect to scope, the evaluation covers three ingestion paths and three scale tiers on a single cloud provider; alternative providers, alternative table formats such as Iceberg or Hudi, and alternative streaming engines were not measured and may exhibit different cost and latency trade-offs. Second, with respect to internal validity, the reported numbers reflect a specific cluster configuration and a specific Snowflake warehouse size; the relative ordering of the ingestion paths is expected to be stable, but absolute values would shift under different sizing choices. Third, with respect to generalizability, the workloads studied include a representative mix of file-landed, event-stream, and CDC

sources, but the conclusions may not transfer cleanly to workloads dominated by high-cardinality joins or by extremely small message sizes, where per-record overhead would change the throughput profile.

Additional threats include the dependence of unit cost on published cloud pricing, which evolves over time and varies by region, and the dependence of latency measurements on the specific Kafka and object-storage backends in use. The protocol's two-window median mitigates short-term variance but does not eliminate longer-term provider drift.

A further consideration is that the data-quality and schema-evolution outcomes reported in Section 8 reflect the specific upstream source-system populations of the reference deployments. Source populations with markedly different defect distributions, for example pipelines that ingest from third-party data providers with weak contractual quality guarantees, would likely exhibit lower expectation pass rates and would require correspondingly more aggressive quarantine and reconciliation workflows. Readers should calibrate the reported percentages against their own source mix rather than treating them as universal.

10. Reproducibility and Data Availability

Reproducibility statement. The methodology in Section 3 is specified in sufficient detail for an independent team to construct a comparable benchmark. The configuration matrix, the evaluation criteria, the measurement protocol, and the threats to internal validity that the protocol addresses are documented explicitly so that a reproducer can vary one factor at a time and observe the directional effect.

Data availability. The underlying production telemetry is not released because it is subject to operational confidentiality. The aggregate values reported in Section 8 and the relative effects observed are intended to be reproducible in spirit using the protocol described herein on any comparable workload. Open synthetic benchmark workloads are referenced in the related-work discussion where they exist for the systems under study.

Code and configuration. Where the techniques discussed are expressible as small artefacts (cluster keys, materialized view DDL, module interfaces, serving configurations, decision predicates), representative listings appear inline so that readers can adapt them. Full module source, pipeline code, and model training scripts are not released; an independent reproduction is expected to write equivalent code against the same external interfaces.

11. Conclusion and Future Work

This paper has presented an empirical evaluation of the medallion architecture realized on Databricks and Apache Spark, with Snowflake as the gold-layer warehouse, across three ingestion paths and three scale tiers. The contribution is a measured account of throughput, end-to-end latency, and unit cost under a controlled measurement protocol that supports comparative judgment across configurations. The headline numbers, between 18 and 62 MB/s per active task, p99 latencies of 9 to 47 seconds, and unit costs of 1.8 to 6.4 USD per terabyte, support the claim that a single Structured Streaming engineering pattern can serve both batch and real-time consumers within a unified medallion architecture without forcing duplicated infrastructure.

Three future-work directions follow from these findings. First, a comparative evaluation against alternative open table formats, including Apache Iceberg and Apache Hudi, would clarify whether the Delta-specific aspects of the present design generalize. Second, an investigation of warehouse-side concurrency under variable gold-layer write rates would refine the cost and latency model for the direct-write integration pattern. Third, an extension of the protocol to include ML feature-store consumers alongside analytical consumers would test whether the gold-layer abstractions remain coherent across consumption modes with very different access patterns.

Conflicts of Interest

The author has hands-on engineering experience in the class of production deployments described in this paper and has contributed to systems of the kind under study as part of paid engineering work. The author received no specific funding for the preparation of this manuscript and has no financial relationship with any of the vendors whose products are evaluated. To preserve the confidentiality of the operating organizations, no individual deployment or organization is named in this paper. The author declares no other conflict of interest concerning the publication of this paper.

References

1. Armbrust, M., Das, T., Sun, L., Yavuz, B., Zhu, S., Murthy, M., Torres, J., van Hovell, H., Ionescu, A., Luszczak, A., Switakowski, M., Szafranski, M., Li, X., Ueshin, T., Mokhtar, M., Boncz, P., Ghodsi, A., Paranjpye, S., Senster, P., Xin, R., and Zaharia, M. Delta Lake: high-performance ACID table storage over cloud object stores. Proceedings of the VLDB Endowment, 13(12), 2020. https://scholar.google.com/scholar?q=Armbrust, M., Das, T., Sun, L., Yavuz, B., Zhu, S., Murthy, M., Torres, J., van Hovell, H., Ionescu, A., Luszczak, A., Switakowski, M., Szafranski, M., Li, X., Ueshin, T., Mokhtar, M., Boncz, P., Ghod | https://doi.org/10.14778/3415478.3415560
2. Armbrust, M., Ghodsi, A., Xin, R., and Zaharia, M. Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics. CIDR, 2021. | https://scholar.google.com/scholar?q=Armbrust, M., Ghodsi, A., Xin,

R., and Zaharia, M. Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics. CIDR, 2021.

3. Zaharia, M., Chowdhury, M., Franklin, M., Shenker, S., and Stoica, I. Spark: cluster computing with working sets. HotCloud, 2010. | https://scholar.google.com/scholar?q=Zaharia, M., Chowdhury, M., Franklin, M., Shenker, S., and Stoica, I. Spark: cluster computing with working sets. HotCloud, 2010.
4. Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., and Stoica, I. Discretized streams: fault-tolerant streaming computation at scale. SOSP, 2013. | https://scholar.google.com/scholar?q=Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., and Stoica, I. Discretized streams: fault-tolerant streaming computation at scale. SOSP, 2013.
5. Armbrust, M., Das, T., Torres, J., Yavuz, B., Zhu, S., Xin, R., Ghodsi, A., Stoica, I., and Zaharia, M. Structured Streaming: a declarative API for real-time applications in Apache Spark. SIGMOD, 2018. | https://scholar.google.com/scholar?q=Armbrust, M., Das, T., Torres, J., Yavuz, B., Zhu, S., Xin, R., Ghodsi, A., Stoica, I., and Zaharia, M. Structured Streaming: a declarative API for real-time applications in Apache Spark. SIGMOD, 2018
6. Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., and Tzoumas, K. Apache Flink: stream and batch processing in a single engine. IEEE Data Engineering Bulletin, 36(4), 2015. | https://scholar.google.com/scholar?q=Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., and Tzoumas, K. Apache Flink: stream and batch processing in a single engine. IEEE Data Engineering Bulletin, 36(4), 2015.
7. Kreps, J., Narkhede, N., and Rao, J. Kafka: a distributed messaging system for log processing. NetDB workshop at SIGMOD, 2011. | https://scholar.google.com/scholar?q=Kreps, J., Narkhede, N., and Rao, J. Kafka: a distributed messaging system for log processing. NetDB workshop at SIGMOD, 2011.
8. Wang, G., Koshy, J., Subramanian, S., Paramasivam, K., Zadeh, M., Narkhede, N., Rao, J., Kreps, J., and Stein, J. Building a replicated logging system with Apache Kafka. PVLDB, 8(12), 2015. | https://scholar.google.com/scholar?q=Wang, G., Koshy, J., Subramanian, S., Paramasivam, K., Zadeh, M., Narkhede, N., Rao, J., Kreps, J., and Stein, J. Building a replicated logging system with Apache Kafka. PVLDB, 8(12), 2015.
9. Stonebraker, M. and Cetintemel, U. One size fits all: an idea whose time has come and gone. ICDE, 2005. | https://scholar.google.com/scholar?q=Stonebraker, M. and Cetintemel, U. One size fits all: an idea whose time has come and gone. ICDE, 2005.
10. Kimball, R. and Ross, M. The Data Warehouse Toolkit, Third Edition. Wiley, 2013. | https://scholar.google.com/scholar?q=Kimball, R. and Ross, M. The Data Warehouse Toolkit, Third Edition. Wiley, 2013.
11. Inmon, W. H. Building the Data Warehouse, Fourth Edition. Wiley, 2005. | https://scholar.google.com/scholar?q=Inmon, W. H. Building the Data Warehouse, Fourth Edition. Wiley, 2005.
12. Linstedt, D. and Olschimke, M. Building a Scalable Data Warehouse with Data Vault 2.0. Morgan Kaufmann, 2015. | https://scholar.google.com/scholar?q=Linstedt, D. and Olschimke, M. Building a Scalable Data Warehouse with Data Vault 2.0. Morgan Kaufmann, 2015.
13. 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., and Unterbrunner, P. The Snowflake elastic data warehouse. SIGMOD, 2016. | https://scholar.google.com/scholar?q=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., Ra
14. Vohra, D. Apache Parquet. Practical Hadoop Ecosystem, Springer, 2016. | https://scholar.google.com/scholar?q=Vohra, D. Apache Parquet. Practical Hadoop Ecosystem, Springer, 2016. | https://scholar.google.com/scholar?q=Vohra, D. Apache Parquet. Practical Hadoop Ecosystem, Springer, 2016.
15. Melnik, S., Gubarev, A., Long, J. J., Romer, G., Shivakumar, S., Tolton, M., and Vassilakis, T. Dremel: interactive analysis of web-scale datasets. PVLDB, 3(1), 2010. | https://scholar.google.com/scholar?q=Melnik, S., Gubarev, A., Long, J. J., Romer, G., Shivakumar, S., Tolton, M., and Vassilakis, T. Dremel: interactive analysis of web-scale datasets. PVLDB, 3(1), 2010.
16. Behm, A., Palkar, S., Agarwal, U., Armbrust, M., Cashman, D., Chambi, S., Datta, S., Falaki, H., Jindal, A., Liang, Y., et al. Photon: a fast query engine for lakehouse systems. SIGMOD, 2022. | https://scholar.google.com/scholar?q=Behm, A., Palkar, S., Agarwal, U., Armbrust, M., Cashman, D., Chambi, S., Datta, S., Falaki, H., Jindal, A., Liang, Y., et al. Photon: a fast query engine for lakehouse systems. SIGMOD, 2022.
17. Apache Software Foundation. Apache Spark Structured Streaming Programming Guide. | https://scholar.google.com/scholar?q=Apache Software Foundation. Apache Spark Structured Streaming Programming Guide. | https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html
18. Databricks Inc. Auto Loader documentation. | https://scholar.google.com/scholar?q=Databricks Inc. Auto Loader documentation. | https://docs.databricks.com/ingestion/auto-loader/index.html
19. Databricks Inc. Delta Live Tables documentation. | https://scholar.google.com/scholar?q=Databricks Inc. Delta Live Tables documentation. | https://docs.databricks.com/delta-live-tables/index.html
20. Databricks Inc. Unity Catalog documentation. | https://scholar.google.com/scholar?q=Databricks Inc. Unity Catalog documentation. | https://docs.databricks.com/data-governance/unity-catalog/index.html
21. Snowflake Inc. Spark Snowflake Connector documentation. | https://scholar.google.com/scholar?q=Snowflake Inc. Spark Snowflake Connector documentation. | https://docs.snowflake.com/en/user-guide/spark-connector

22. Snowflake Inc. External Tables documentation. | https://scholar.google.com/scholar?q=Snowflake Inc. External Tables documentation. | https://docs.snowflake.com/en/user-guide/tables-external-intro
23. Debezium project. Debezium documentation. | https://scholar.google.com/scholar?q=Debezium project. Debezium documentation. | https://debezium.io/documentation/
24. Chen, A., Chow, A., Davidson, A., DCunha, A., Ghodsi, A., Hong, S. A., Konwinski, A., Mewald, C., Murching, S., Nykodym, T., et al. Developments in MLflow: a system to accelerate the machine learning lifecycle. DEEM at SIGMOD, 2020. | https://scholar.google.com/scholar?q=Chen, A., Chow, A., Davidson, A., DCunha, A., Ghodsi, A., Hong, S. A., Konwinski, A., Mewald, C., Murching, S., Nykodym, T., et al. Developments in MLflow: a system to accelerate the machine learning
25. Hellerstein, J. M., Sreekanti, V., Gonzalez, J. E., Dalton, J., Dey, A., Nag, S., Ramachandran, K., Arora, S., Bhattacharyya, A., Das, S., et al. Ground: a data context service. CIDR, 2017. | https://scholar.google.com/scholar?q=Hellerstein, J. M., Sreekanti, V., Gonzalez, J. E., Dalton, J., Dey, A., Nag, S., Ramachandran, K., Arora, S., Bhattacharyya, A., Das, S., et al. Ground: a data context service. CIDR, 2017.