GLOBAL MULTIDISCIPLINARY PERSPECTIVES JOURNAL

A Unified Batch-and-Streaming Data Architecture for Machine Learning Applications Incorporating Predictive Fault Detection and Validation

Sai Kiran Pullela
Data Engineer, Experian, Chicago, IL, USA

* Corresponding Author: **Sai Kiran Pullela**

Article Info

ISSN (online): 3107-3972
Volume: 01
Issue: 04
July-August 2024
Received: 21-07-2024
Accepted: 24-08-2024
Page No: 69-73

Abstract
Machine learning (ML) products increasingly depend on data platforms that must simultaneously support high-throughput batch analytics, low-latency streaming decisions, and continuously evolving schemas, features, and model requirements. Yet many enterprises still operate split architectures where batch ETL, real-time pipelines, and ML lifecycle tooling are assembled as loosely coupled systems, amplifying operational risk, data quality regressions, and silent ML failures. This paper proposes UBSDA (Unified Batch-and-Streaming Data Architecture), a lakehouse-centered reference architecture that unifies batch and streaming ingestion, storage, transformation, and feature publication while embedding predictive fault detection and validation as first-class capabilities. UBSDA introduces (i) a single data truth layer for both offline training and online inference, (ii) contract-driven schema governance with evolution support, (iii) multi-stage validation gates that combine statistical checks, constraint-aware learning, and drift monitoring, and (iv) a fault prediction service trained on pipeline telemetry to anticipate failures before service-level objectives are violated. We detail the architecture, formalize validation and fault-risk scoring, and present an evaluation methodology showing how unified storage plus proactive detection reduces duplicated transformations, shortens recovery loops, and improves ML reliability across domains.

DOI: https://doi.org/10.54660/GMPJ.2024.1.4.69-73

Keywords: lakehouse, batch processing, stream processing, feature store, data quality, schema evolution, drift detection, ML observability, predictive fault detection, validation

1. Introduction

Modern ML systems are not just models; they are end-to-end pipelines in which ingestion, transformation, labeling, training, and serving share a common failure surface. A central architectural trend is the lakehouse design point, which unifies warehouse-style reliability (transactions, governance, time travel) with lake-style openness and direct access for advanced analytics and ML workloads. This unification motivates a platform that treats batch and streaming as two execution modes over a single curated data substrate rather than separate products stitched together. [1]

A second trend is the maturation of lakehouse definitions and requirements beyond vendor branding. Recent surveys highlight recurring requirements such as interoperability across engines, minimized data duplication, controlled concurrency, and explicit support for both batch and streaming computations on the same storage layer. These requirements point toward reference architectures that explicitly connect lakehouse storage with ML lifecycle needs like feature reuse, lineage, and quality enforcement. [2]

In parallel, the underlying data-platform ecosystem remains shaped by the evolution of data lakes: flexible, multi-format repositories that succeed at scale but often struggle with governance, quality, and interoperability when used as a default substrate for production ML. Understanding these roots matters because many failure modes in ML originate from lake-like weak guarantees (for example, ad-hoc schemas and unmanaged evolution) that propagate into training and inference. [3]

2. Background and Motivation

2.1. Stream processing and reliability foundations

Real-time ML increasingly depends on streaming primitives: event-time semantics, windowing, stateful computation, and fault tolerance. Stream-processing research and practice have evolved substantially, with modern systems offering stronger guarantees and richer time semantics, but they also introduce operational complexity (state management, reconfiguration, failure recovery) that directly impacts ML freshness and correctness. [4]

2.2. Scalability under microservice deployment

Enterprises frequently deploy stream processors as microservices on cloud orchestration platforms. Large-scale benchmarking studies show that scalability and throughput behavior differ materially across frameworks when deployed in containerized environments, implying that architectural choices must be paired with measurable performance and operational constraints rather than assumed equivalence. [5]

2.3. Open table formats as a unification lever

A practical enabler for unifying batch and streaming is an open table format that supports atomic commits, metadata evolution, and multi-engine access. Open ecosystems around lakehouse tables have accelerated; release histories illustrate rapid iteration and feature growth that make one storage for many engines feasible in practice. [6]

3. Problem Statement

Despite convergence trends, many organizations still run: (i) separate batch ETL pipelines for training sets, (ii) separate streaming pipelines for online decisions, and (iii) separate validation and monitoring stacks bolted onto each. This fragmentation creates four recurring failure classes:

1. Training-serving skew from duplicated transformation logic.
2. Silent data regressions (schema drift, distribution drift, missingness spikes) that pass traditional uptime checks.
3. Slow failure reaction loops, because detection is post-hoc and diagnosis lacks unified lineage.
4. Operational fragility, because streaming recovery and batch backfills are handled with different semantics and tools.

To address these failures, ML platforms need observability tailored to ML pipeline semantics, not just infrastructure health. Observability must support assisted detection, diagnosis, and reaction to ML-related bugs across pipeline components. [7]

Additionally, software-engineering research identifies that pipeline quality is influenced by many interacting factors and root causes spanning specification gaps, transformation complexity, and change management. A useful architecture must therefore embed quality enforcement into the platform, not treat it as an afterthought. [8]

4. Data Quality and Validation as First-Class Concerns

4.1. Dataset quality impacts ML outcomes

Dataset quality spans completeness, accuracy, consistency, timeliness, and representativeness across the dataset lifecycle. Surveys of dataset quality research emphasize that ML quality is bounded by data quality and that systematic quality evaluation must be a lifecycle activity rather than a one-time cleaning step. [9]

4.2. Drift-aware scoring for dynamic environments

In industrial streaming contexts, quality is not static: distributions and operational regimes shift. Drift-aware data quality scoring frameworks formalize how quality metrics should adapt under changing conditions, motivating validation gates that explicitly model non-stationarity rather than relying on fixed thresholds. [10]

4.3. Contracts to shift quality left

A complementary strategy is shift-left governance: encode expectations about structure and semantics as data contracts so producers own conformance and consumers can rely on compatibility and documented meaning. In streaming systems, contracts are commonly implemented via schema registries and compatibility rules, but they must be integrated into end-to-end ML pipelines to prevent downstream breakage. [11]

4.4. Beyond syntactic compatibility: generalized schema evolution

Schema evolution in ML pipelines is especially challenging because compatible is not purely syntactic; a rename or type change may be semantically compatible but syntactically disruptive. Emerging approaches propose compound or AI-assisted registries that learn mappings and represent transformations as intermediate representations, expanding what can be handled without manual intervention and downtime. [12]

5. UBSDA Reference Architecture

5.1. Design principles

UBSDA is guided by five principles: (1) One storage truth for batch and streaming outputs. (2) Contract-governed evolution for schemas and features. (3) Validation gates everywhere (ingest to curate to publish to serve). (4) Predict first, then react: fault prediction complements reactive alerts. (5) Reproducibility: time travel and lineage make training sets replayable.

5.2. Logical layers

UBSDA organizes data into four logical layers: Ingress Layer (events plus batches): raw append-only ingestion. Curated Layer: deduplication, schema enforcement, entity resolution, and PII handling. Feature Layer: point-in-time correct feature publication for training and serving. Serving and Feedback Layer: model outputs, labels or feedback, and monitoring signals.

5.3. Unified compute model

Batch and streaming workloads share the same transformation definitions via one of two patterns: Dual-mode pipelines where identical transformations run in streaming mode for freshness and batch mode for backfills; and Streaming-as-default where streaming pipelines write incrementally to the curated and feature layers and batch reads become bounded queries over the same tables.

5.4. Validation gates

UBSDA combines rule-based checks (schema, constraints, null rates), statistical tests (distribution shifts), and constraint-aware learning. In particular, shape-constrained regression methods can incorporate domain expectations (monotonicity and convexity) to detect subtle invalid data that violates expected relationships. Comparative evaluations of shape-

constrained approaches motivate selecting validation learners that balance accuracy and runtime for industrial adoption. [13] UBSDA further operationalizes expert knowledge in validation by training constrained models and using training error patterns to identify invalid regions, enabling automated validation that remains interpretable to domain experts and effective on subtle errors. [14]

6. Drift Monitoring and ML-Specific Validation

6.1. Unsupervised drift monitoring

Many production scenarios lack immediate labels, so unsupervised drift detection becomes essential. Recent surveys provide taxonomies and guidance for unsupervised drift detection schemes applicable to monitoring and anomaly detection, supporting UBSDA's requirement for label-free validation in streaming contexts. [15]

6.2. Practical drift detection landscape

Systematic reviews show that drift detection spans window-based tests, distance measures, ensemble approaches, and neural methods, but real deployments must consider computational efficiency, imbalance, and non-tabular modalities. UBSDA therefore treats drift detection as a configurable capability with policy-driven triggers, not a single algorithm. [16]

6.3. Statistical measures for drift detection

Statistical-measure approaches for drift detection provide pragmatic detectors that can run online with low overhead, making them attractive as always-on sentinels in the validation pipeline, especially when combined with escalation tiers (warn to quarantine to rollback). [17]

6.4. Advanced online drift objectives

For higher-stakes services, UBSDA can adopt modern online drift detection objectives that explicitly measure concept discrepancy and optimize detection sensitivity under streaming constraints, improving detection under complex, high-dimensional regimes. [18]

6.5. Tooling and operationalization

Operational success depends on tooling that supports controlled drift patterns, real-time detection workflows, and user interaction for diagnosis. Drift-detection tools that focus on practical monitoring workflows illustrate how to bridge research detectors into usable platform features, aligning with UBSDA's assisted diagnosis goal. [19]

7. Predictive Fault Detection in Data and ML Pipelines

7.1. From observability to prediction

Reactive monitoring reports failures after they occur; UBSDA adds a fault prediction service that forecasts a failure risk score using pipeline telemetry. Telemetry features include lag growth, error bursts, schema change frequency, unusual missingness, backpressure signals, checkpoint durations, and anomaly scores from validation gates.

7.2. Risk scoring formulation

Let x_t be a feature vector derived from telemetry and data-quality signals over a time window ending at t. UBSDA trains a classifier f_theta(x_t) that outputs fault probability pt in [1].

A practical platform score is shown in Equation (1).

Equation (1)

$$\text{Risk}(t) = \text{sigma}(\text{alpha} * \log(p\_t / (1 - p\_t)) + \text{beta} * \text{DeltaDQ}(t) + \text{gamma} * \text{DeltaLag}(t))$$

7.3. Governance and QA alignment

Predictive fault detection is most valuable when it triggers controlled responses (quarantine, auto-rollback, staged retries) aligned with software lifecycle governance. Architectures that integrate defect prediction with automated testing and governance workflows support UBSDA's view that data pipeline QA and software QA must be orchestrated together. [20]

7.4. Borrowing from defect prediction research

Pipeline faults often mirror software defects: they correlate with change frequency, complexity hotspots, and historical failure patterns. Comparative studies of ML models for defect prediction motivate UBSDA's model selection approach (tree ensembles for structured telemetry, calibrated linear models for interpretability, and kNN-style detectors for local anomaly neighborhoods). [21]

7.5. Automation economics

Because validation and fault prediction add compute and operational overhead, the architecture must justify costs. Empirical work on automation ROI motivates an explicit automation economics lens: quantify time and cost savings from early detection, reduced incidents, and faster recovery, and optimize the gate placement accordingly. [22]

7.6. Fault prediction model families

Additional fault-prediction analyses comparing Random Forest, Logistic Regression, and kNN highlight trade-offs among interpretability, nonlinear fit, and sensitivity to local neighborhoods. UBSDA leverages these trade-offs by using ensembles for early warning and simpler models for explainability and governance sign-off. [23]

8. Use Cases and Security Considerations

8.1. High-stakes ML domains

In domains like clinical practice, ML systems face heightened requirements for correctness, traceability, and robust monitoring because errors impact real people. UBSDA's unified data substrate and validation-first approach supports auditable training and serving datasets and controlled evolution, aligning with the operational realities of AI-enabled decision support. [24]

8.2. Security as part of pipeline correctness

Security failures (tampering, data poisoning pathways, unsafe data exposure) manifest as correctness failures in ML outcomes. Bridging information security and cybersecurity viewpoints is necessary to protect streaming and batch pipelines end-to-end, including the governance layer (contracts and registries) and the storage truth layer (immutability and access control). [25]

8.3. Practical lakehouse operations

For implementers, UBSDA's lakehouse layer requires concrete operational patterns: ACID commits, time travel, compaction, and unified batch and stream reads and writes. Practical engineering guidance on lakehouse table

operations, including patterns where streaming is both a sink and a source, supports UBSDA's single-substrate implementation strategy. [26]

8.4. Smart infrastructure and public utility pipelines

When ML supports smart infrastructure, failures can cascade into operational disruption. Pipeline security and resilience must therefore be designed for continuous operation, including robust monitoring, controlled evolution, and incident response automation. [27]

8.5. Online safety and governance

ML pipelines increasingly touch online safety and user-facing trust. UBSDA's contracts, validation, and drift monitoring reduce silent regressions that undermine trust, while its governance hooks support policy compliance and accountability across the data lifecycle. [28]

9. Evaluation Methodology (Proposed)

To evaluate UBSDA in a reproducible manner, we recommend:

1. Workloads: mixed batch backfills plus streaming ingestion with evolving schemas and changing distributions.
2. Baselines: split architectures (separate batch warehouse plus stream store) versus unified lakehouse substrate.
3. Metrics: (i) training-serving skew rate, (ii) time-to-detect quality regressions, (iii) false-positive quarantine rate, (iv) recovery time, (v) incremental compute cost, and (vi) end-to-end decision freshness.
4. Fault injection: controlled schema evolution events, missingness spikes, delayed partitions, and streaming worker failures.
5. Ablations: remove predictive fault detection; remove drift detection; remove constraint-aware validation; compare.

This methodology emphasizes measuring reliability outcomes rather than only throughput.

10. Threats to Validity

External validity: results may differ across industries due to different label delays, regulatory constraints, and data distributions.

Internal validity: fault labels for training the predictor may be noisy; careful incident taxonomy and labeling practices are required.

Construct validity: data quality is multi-dimensional; selecting a narrow metric set risks optimizing the wrong proxy.

Operational validity: success depends on governance adoption; contracts and validation must be treated as product requirements, not optional engineering tasks.

11. Conclusion and Future Work

UBSDA unifies batch and streaming data architectures for ML by centering on a shared lakehouse truth layer while embedding validation, drift detection, and predictive fault detection as first-class platform capabilities. The architecture reduces duplicated transformations, mitigates silent data regressions, and shortens the loop from detection to corrective action by treating ML pipeline reliability as a systems problem, not a monitoring afterthought.

Future work includes: (i) adaptive policy learning for validation gate placement, (ii) causal diagnosis for pipeline failures, (iii) privacy-preserving drift monitoring, and (iv) standardized benchmarks for ML data reliability comparable to established system benchmarks.

References

1. Armbrust M, *et al*. Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics. In: CIDR; 2021.
2. Schneider J, *et al*. The lakehouse: State of the art on concepts and technologies. SN Comput Sci. 2024;5:449. doi:10.1007/s42979-024-02737-0.
3. Azzabi S, Alfughi Z, Ouda A. Data lakes: A survey of concepts and architectures. Computers. 2024;13(7):183. doi:10.3390/computers13070183.
4. Fragkoulis M, *et al*. A survey on the evolution of stream processing systems. VLDB J. 2024;33:507–541. doi:10.1007/s00778-023-00819-8.
5. Henning S, Hasselbring W. Benchmarking scalability of stream processing frameworks deployed as microservices in the cloud. J Syst Softw. 2024;208:111879. doi:10.1016/j.jss.2023.111879.
6. Apache Iceberg. Releases (0.12.0) [Internet]. 2021 Aug 15.
7. Shankar S, Parameswaran A. Towards observability for machine learning pipelines. In: CIDR; 2022.
8. Foidl H, Golendukhina V, Ramler R, Felderer M. Data pipeline quality: Influencing factors, root causes of data-related issues, and processing problem areas for developers. J Syst Softw. 2024;207:111855. doi:10.1016/j.jss.2023.111855.
9. Gong Y, Liu G, Xue Y, Li R, Meng L. A survey on dataset quality in machine learning. Inf Softw Technol. 2023;162:107268. doi:10.1016/j.infsof.2023.107268.
10. Bayram F, Ahmed BS, Hallin E. Adaptive data quality scoring operations framework using drift-aware mechanism for industrial applications. J Syst Softw. 2024;217:112184. doi:10.1016/j.jss.2024.112184.
11. Yokota R. Using data contracts with Confluent Schema Registry [Internet]. Confluent Blog; 2023 Oct 18.
12. Fu SD, Chen X. Compound schema registry. arXiv [Preprint]. 2024. arXiv:2406.11227.
13. Bachinger F, Kronberger G. Comparing shape-constrained regression algorithms for data validation. In: Proceedings; 2022. p. 147–154. doi:10.1007/978-3-031-25312-6_17.
14. Bachinger F, Ehrlinger L, Kronberger G, Woss W. Data validation utilizing expert knowledge and shape constraints. J Data Inf Qual. 2024;16(2). doi:10.1145/3661826.
15. Hinder F, Vaquet V, Hammer B. One or two things we know about concept drift—a survey on monitoring in evolving environments. Part A: detecting concept drift. Front Artif Intell. 2024;7:1330257. doi:10.3389/frai.2024.1330257.
16. Hovakimyan G, Barseghyan JMB. Evolving strategies in machine learning: A systematic review of concept drift detection. Information. 2024;15(12):786. doi:10.3390/info15120786.
17. Ali Abdu NA, Basulaim KO. Machine learning in concept drift detection using statistical measures. Int J Comput Appl. 2024;46(5):281–291. doi:10.1080/1206212X.2023.2289706.
18. Wan K, Liang Y, Yoon S. Online drift detection with

maximum concept discrepancy. In: KDD; 2024. p. 2924–2935. doi:10.1145/3637528.3672016.
19. Greco S, *et al*. DriftLens: A concept drift detection tool. In: EDBT; 2024.
20. Sivva SD, Thalakanti RR, Bandari SSG, Yettapu SDR. AI-driven decision intelligence for agile software lifecycle governance: An architecture-centered framework integrating machine learning defect prediction and automated testing. 2023. Available from: https://www.ijetcsit.org/index.php/ijetcsit/article/view/554
21. Gunda SK. Comparative analysis of machine learning models for software defect prediction. In: Proc Int Conf Power Energy Control Transm Syst (ICPECTS); 2024. p. 1–6. doi:10.1109/ICPECTS62210.2024.10780167.
22. Kacheru G, Bajjuru R, Arthan N. The ROI of software automation: Measuring time and cost savings. Int J Commun Netw Inf Secur. 2023;15(4):774–785.
23. Gunda SK. Fault prediction unveiled: Analyzing the effectiveness of random forest, logistic regression, and KNeighbors. In: Proc Int Conf Self Sustain Artif Intell Syst (ICSSAS); 2024. p. 107–113. doi:10.1109/ICSSAS64001.2024.10760620.
24. Kacheru G. Revolutionizing healthcare: The role of artificial intelligence in clinical practice. J Comput Anal Appl. 2023;31(4):1546–1554.
25. Pittala SK, Ashok VKC. A new era in security: Bridging information security and cybersecurity. Int J Multidiscip Futur Dev. 2023;4(1):69–72. doi:10.54660/IJMFD.2023.4.1.69-72.
26. Lee D, Wentling T, Haines S, Babu P. Delta Lake: The definitive guide—modern data lakehouse architectures with data lakes. 1st ed. 2024.
27. Ashok VKC. Cybersecurity for smart infrastructure and public utilities. Int J Multidiscip Res Growth Eval. 2023;4(2):947–949. doi:10.54660/.IJMRGE.2023.4.2.947-949.
28. Pittala SK. Cybersecurity and online safety: A critical asset in the information era. J Front Multidiscip Res. 2023;4(1):576–579. doi:10.54660/.jfmr.2023.4.1.576-579.