International Journal of Computer (IJC)
ISSN 2307-4523 (Print & Online)
https://ijcjournal.org/index.php/InternationalJournalOfComputer/index

Conceptual Approaches to Organizing Feature Stores in High-Load Machine Learning Systems

Artem Korkhov*

Software Engineer, *Stockholm, Sweden*
Email: artyom.korkhov@gmail.com

Abstract

The paper examines conceptual approaches to organizing feature stores in high-load machine learning systems. The growth of production ML has exposed persistent problems, including duplicated feature logic, training–serving skew, and brittle pipelines, particularly under low-latency inference and heavy batch training workloads. This paper proposes an original analytical framework for organizing feature stores under high-load conditions. The framework links internal architectural decisions (registry design, transformation semantics, offline/online materialization, synchronization, geo-distribution, and maintenance control) to operational objectives (latency, point-in-time correctness, reproducibility, governance, and cost discipline), enabling structured design selection rather than a descriptive survey of existing systems. The study characterizes functional blocks of a feature store (feature registry, transformation layer, offline and online materializations, and serving interfaces), then analyzes consistency guarantees, point-in-time correctness, refresh strategies, and geo-distribution patterns. Methods rely on a comparative analysis of recent systems and surveys, with a focus on scalability bottlenecks in joins, cache design, and maintenance scheduling. The conclusion formulates design recommendations for selecting storage layouts, synchronization policies, and operational controls across the ML lifecycle. The paper targets engineers and researchers working on production MLOps and data infrastructure. Implications for feature reuse and compliance are discussed.

Keywords: feature store; MLOps; high-load systems; online inference; offline training; point-in-time correctness; feature freshness; data consistency; geo-distribution; maintenance scheduling.

Received: 12/10/2025
Accepted: 2/10/2026
Published: 2/20/2026

---

* *Corresponding author.*

1. Introduction

Production machine learning increasingly depends on infrastructure that separates feature computation from model development while preserving reproducibility under frequent retraining and continuous serving. High-load settings intensify typical failure modes, including latency spikes caused by synchronous transformations at request time, silent training–serving skew introduced by divergent pipelines, and operational fragility when feature refreshes collide with peak traffic.

Beyond summarizing known architectures, the paper formalizes a conceptual model of feature-store organization for high-load ML systems. The model is expressed as a set of interacting layers and design levers, together with explicit decision links between workload pressure (concurrent batch + low-latency serving) and engineering choices (materialization, synchronization, correctness semantics, and maintenance governance).

The paper formulates an analytical framework for organizing feature stores as a bounded subsystem of a production MLOps stack. The framework is intended to guide architecture selection under sustained high-load, explicitly balancing point-in-time correctness, latency budgets, refresh economics, and governance constraints across training and serving.

The study develops the framework through three steps:

1) Define a unified conceptual decomposition of a feature store for mixed training and inference workloads (registry, transformation semantics, offline/online materialization, and serving interfaces).

2) Specify the framework's decision criteria for correctness and scalability, focusing on point-in-time semantics, synchronization between materializations, refresh control, and maintenance scheduling.

3) Map the resulting design levers to concrete operational practices in MLOps (release discipline, observability, governance, cost controls), yielding an actionable structure for selecting architectures rather than cataloging implementations.

Novelty is determined by an integrated analytical mapping between feature-store internal design decisions (materialization strategy, synchronization, geo-distribution, and maintenance) and MLOps practices that regulate reliability, traceability, and cost discipline in production environments.

2. Materials and Methods

Materials for the analytical synthesis were drawn from recent research and surveys on MLOps platforms and feature-store systems: L. Berberi [1] provides an assessment framework of MLOps platforms and tool capabilities; J. de la Rúa Martínez [2] describes a unified feature-store design with APIs supporting heterogeneous query workloads; Z. Fang [3] structures the ML lifecycle as an operations process and aggregates survey evidence across stages; N. Hewage [4] compares tool support for operationalizing ML delivery; D. Kreuzberger [5] proposes an architectural view of MLOps components and workflows; A. Li [6] details architectural components and geo-

distribution patterns for a managed feature store; R. Liu [7] studies performance bottlenecks in feature-store pipelines and formalizes optimizations for point-in-time joins; L. Orr [8] frames feature stores inside end-to-end ML pipelines and highlights embedding-centric pressures; S. Wooders [9] models feature maintenance as a scheduling problem guided by downstream accuracy; M. Zarour [10] synthesizes best practices, challenges, and maturity models for MLOps adoption.

Methods relied on comparative analysis, structured source analysis, and architectural decomposition. The paper employs an analytical approach to reconcile terminology across surveys, derive a unified conceptual model of feature-store components, and evaluate design trade-offs under high-load constraints, utilizing evidence and system descriptions reported in the selected literature.

Methodologically, the work follows a framework-construction approach: recurring architectural primitives and operational constraints are extracted from the literature, then organized into a consistent set of layers, design levers, and decision links that explain how high-load pressures translate into concrete feature-store organization choices.

3. Results

High-load machine learning systems operate under a dual regime of data access, where long-running batch jobs for training and evaluation coexist with latency-sensitive online inference requests. Feature stores operationalize this regime by separating feature definitions from their physical materializations and exposing stable serving interfaces, thereby reducing duplication of feature logic across teams and pipelines [2]. This approach places feature management within broader operations workflows, as described in MLOps architectures and surveys [3,5].

The analytical framework produced in this study is structured around (i) a layered reference decomposition and (ii) a set of high-load design levers with explicit trade-off dimensions. In the remainder of the section, the layers define the stable vocabulary of feature-store organization. At the same time, the levers provide a decision structure for selecting architectures subject to constraints on throughput, latency, and correctness.

A conceptual model that remains consistent across system-oriented sources decomposes a feature store into four interacting layers. The first layer, a feature registry, stores definitions, ownership metadata, lineage signals, and discovery-oriented descriptors, enabling reuse and governance at scale [2, 6]. The second layer, a transformation layer, expresses feature computation as reproducible pipelines that ingest raw data and emit feature values in forms suitable for both historical backfills and incremental updates [6, 8]. The third layer, materialization, maintains two complementary views: an offline store optimized for training-time scans and point-in-time extraction, and an online store optimized for key-based low-latency retrieval at serving time [2, 6, 7]. The fourth layer, serving, exposes online feature retrieval APIs and batch extraction interfaces while tracking versions and freshness signals to support operational monitoring [2, 5].

Figure 1 is introduced as a compact representation of the proposed framework: it defines the boundaries between layers, the control/data-plane separation, and the placement points where synchronization, correctness semantics, and refresh control must be specified under high load. Figure 1 integrates these layers into a single high-load

placement pattern, synthesizes system descriptions of unified feature-store services and managed architectures Reference [2, 6], and aligns them with the MLOps pipeline view, where feature engineering becomes a governed production stage rather than an ad hoc preprocessing step [5].

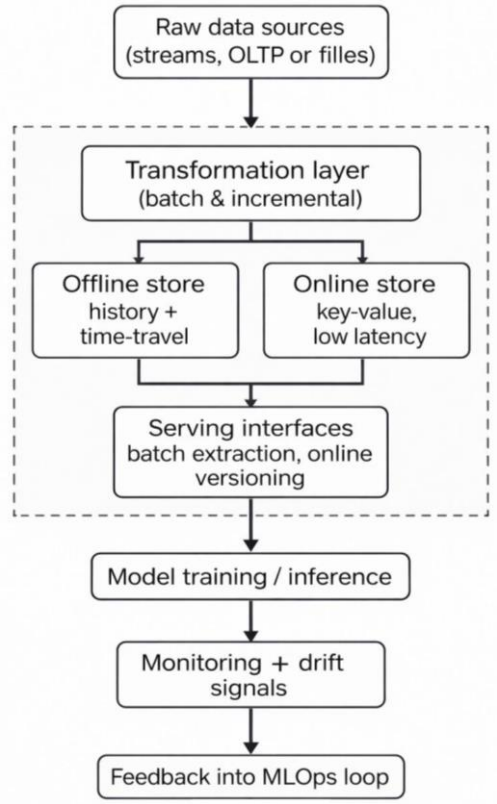

Figure 1: Conceptual placement of a feature store in a high-load ML system (analytical synthesis based on [2, 5, 6])

The dominant correctness issue for feature stores in production arises from mismatches between how features are computed for training and how they are retrieved for serving. Sources on managed feature stores view this as an engineering problem involving synchronization between offline and online materializations, version control for feature definitions, and preventing data leakage through temporal misalignment [2, 6]. The point-in-time requirement formalizes this: training datasets must pair labels with feature values that were available at the historical event time, not recomputed using future information. Within feature-store pipelines, point-in-time joins become a primary cost driver because they combine large event tables with feature tables under temporal constraints [7].

Performance evidence from feature-store pipeline research suggests that point-in-time join is a database-style optimization target. The proposed optimizations implemented on Feathr show that feature-store pipelines can be accelerated substantially relative to baselines when joined planning, indexing, and execution are tailored to the point-in-time pattern [7]. Under high-load regimes, this shift in architectural emphasis prioritizes precomputation and incremental maintenance for frequently used features, as repeated on-demand recomputation increases both resource pressure and variance in batch completion times [7, 9].

High-load online inference imposes stricter constraints than batch training: a feature store must return features within an end-to-end latency budget that competes with model execution and network overhead. In the unified feature-store design described for heterogeneous workloads, online access patterns extend beyond key-value retrieval to include row-oriented, columnar, and similarity search interfaces, reflecting modern feature modalities and retrieval demands [2]. This motivates a conceptual separation between logical feature definitions and multiple physical serving backends, each tuned to a query class, while remaining governed by shared metadata and versioning [2, 6].

Geo-distribution becomes a defining requirement when inference traffic originates from multiple regions and when regulatory or availability constraints require locality. A managed geo-distributed feature-store architecture explicitly treats replication, regional materialization, and control-plane/data-plane separation as first-class design elements, with the control plane maintaining definitions and policy, and the data plane ensuring regionally efficient access and refresh [6]. When combined with the MLOps platform landscape, this highlights a recurring organizational pattern: feature stores rarely stand alone and instead integrate into orchestration, monitoring, and deployment stacks whose selection depends on platform capability coverage and adoption maturity [1, 10].

Maintenance and refresh policies define the operational stability of high-load feature stores. Treating feature refresh as a purely time-driven batch job disregards demand heterogeneity and downstream sensitivity. Accuracy-aware scheduling reframes maintenance as a decision problem guided by the impact of model accuracy and query behavior, implying that refresh frequency can be adapted at the granularity of keys and time, rather than being applied uniformly across a feature table [9]. This conceptual move links system efficiency to model-level outcomes and suggests that a feature-store architecture benefits from explicit observability signals (usage, staleness, model sensitivity) that feed maintenance control loops within the broader MLOps pipeline [5, 9, 10].

A separate pressure emerges from embedding-centric pipelines. Feature stores have historically focused on tabular features. Still, modern production systems frequently treat pre-trained embeddings as features whose lifecycle introduces additional management needs: tracking embedding training data, evaluating embedding quality, and monitoring downstream behavior when embeddings shift [8]. Conceptually, this extends feature-store organization from “feature values” to “feature representations,” thereby strengthening the argument for richer metadata, lineage, and monitoring interfaces, as well as integration with platform-level MLOps governance practices, as documented in surveys and systematic reviews.

4. Discussion

The results in Section “Results” support a single interpretation: in high-load production, a feature store is best analyzed as a coupled socio-technical subsystem rather than as a storage component. The layered decomposition (registry → transformations → offline/online materializations → serving interfaces) provides the minimum stable vocabulary for describing how feature definitions, physical layouts, and operational controls interact under mixed workloads. In this view, high-load pressure is not a generic “scale” label; it is a concrete combination of concurrent backfills and point-in-time dataset assembly, strict tail-latency constraints for online inference, and frequent refresh and retraining cycles that compete for compute and I/O.

The analytical framework derived from prior system and survey evidence isolates five design levers that repeatedly determine failure and success modes in this regime: dual materialization and its synchronization policy, point-in-time extraction strategy and its join plan/metadata design, maintenance scheduling logic and its dependency on downstream model sensitivity, multi-backend serving aligned with heterogeneous feature modalities, and geo-distributed placement separating control-plane decisions from data-plane execution. The contribution of the framework is not a restatement of these topics; it is the explicit mapping between each lever and the operational objective it primarily affects (latency budgets, point-in-time correctness, reproducibility, governance overhead, and cost discipline), so that design selection can be reasoned about as an optimization under constraints rather than as an implementation preference.Prior studies provide strong building blocks for this mapping, but treat them at different granularities. Unified feature-store designs emphasize stable APIs and the separation of feature definitions from multiple access patterns [2], while managed, geo-distributed architectures formalize control-plane and replication concerns as first-class elements [6]. Pipeline research demonstrates that point-in-time extraction is a dominant cost center and benefits from database-grade optimizations tailored to temporal joins [7]. Maintenance scheduling work reframes refresh as a decision problem guided by accuracy impact on accuracy rather than as a fixed, periodic job [9]. Surveys and systematic reviews position these technical choices inside organizational MLOps practices and maturity trajectories [1, 3–5, 10]. The framework in this paper consolidates these lines into a single decision structure: each lever is discussed only to the extent that it changes the feasibility of meeting production objectives under sustained load.

Table 1 operationalizes the framework by enumerating the levers, pairing each with its intended high-load effect and dominant operational trade-off, forming a decision checklist that can be applied during feature-store design and platform selection. Table 1 should be read as an explanatory device for the empirical-looking statements in the Results section. For example, when the Results emphasize point-in-time joins as a primary cost driver, Table 1 identifies the corresponding lever (“point-in-time extraction optimization”) and makes the trade-off explicit: runtime efficiency is purchased at the expense of metadata, planning, and implementation complexity [7]. When the Results highlight the need to adapt refresh to demand and model sensitivity, Table 1 links that observation to “accuracy-aware maintenance scheduling,” clarifying why naive periodic refresh increases load variance and why instrumentation maturity becomes a prerequisite for stable operation at scale [5, 9, 10]. This connection between observation and lever is the intended mechanism of clarification: each reported pressure in Results is paired with the concrete knob that practitioners can actually set.

Table 1: Design levers for high-load feature stores and their operational effects [2, 6, 7, 9]

Design lever	**Intended effect in high-load regimes**	**Operational trade-off**
Dual materialization (offline + online)	Stable training extraction and low-latency retrieval	Synchronization complexity; staleness management
Point-in-time extraction optimization	Faster training-set construction; lower resource burn	Implementation complexity; planning/metadata overhead
Accuracy-aware maintenance scheduling	Resource-efficient refresh aligned with model impact	Requires sensitive signals and monitoring maturity
Multi-backend serving (row/column/similarity)	Fit-for-query retrieval under mixed modalities	Operational burden of multiple backends
Geo-distributed data plane	Regional latency reduction; availability	Replication policy complexity; consistency choices

Text Table 1 clarifies an architectural implication: when the offline store and online store are treated as independent products, operational debt accumulates around correctness guarantees. The managed-system view treats correctness violations (notably training–serving skew) as a primary engineering concern [2, 6]. Pipeline-optimization work makes this constraint explicit: when point-in-time extraction is treated as an afterthought, training-set assembly becomes join-dominated and resource-intensive, amplifying batch completion variance under frequent retraining [7].

Table 2 connects feature-store capabilities to MLOps practices and maturity-oriented adoption patterns discussed in surveys and systematic reviews [3–5, 10], as well as to platform-capability framing in MLOps tool-landscape work [1].

Table 2: Mapping feature-store capabilities to MLOps practices and maturity signals [1, 3–5, 10]

Feature-store capability	**Operational practice in MLOps**	**Maturity signal in organizational adoption**
Feature registry with ownership and lineage	Traceability, reproducibility, governance controls	Standardized processes for reuse and audit readiness
Versioned transformations and backfills	Repeatable pipelines, controlled releases	Automated pipeline orchestration and rollback discipline
Online serving with freshness contracts	SLO-driven monitoring and incident handling	Continuous monitoring tied to retraining triggers
Policy-driven access and sharing	Cross-team collaboration with compliance controls	Formal data-product management for ML artifacts
Platform integration (orchestration/monitoring)	End-to-end lifecycle coordination	Consolidated toolchain selection and platform rationalization

Table 2 extends prior work by aligning technical capabilities with operational practices in a way that is actionable during platform selection. Surveys describe MLOps as an end-to-end discipline and enumerate broad categories (data management, orchestration, deployment, monitoring) [3–5]. At the same time, landscape and maturity-model studies focus on tool coverage and organizational adoption patterns [1, 10]. System papers on feature stores, in contrast, dive into interfaces, storage backends, consistency, and geo-distribution [2, 6]. The mapping in Table 2 bridges these levels: it shows, for instance, that "versioned transformations and backfills" are not merely an engineering convenience but a direct prerequisite for release discipline and rollback semantics in production pipelines [5]; similarly, "online serving with freshness contracts" operationalizes the survey-level monitoring narrative into measurable SLO-driven incident handling that can trigger retraining or refresh adjustments [3, 10]. This literature-grounded linkage clarifies why feature-store design decisions cannot be evaluated independently of the surrounding MLOps control surface.

Across surveys and systematic reviews, MLOps is framed as a coordination layer spanning data engineering, ML engineering, and operations, with the feature store serving as the point where data contracts meet deployment constraints and monitoring obligations [3–5, 10]. Landscape analyses further indicate that platform choices tend to follow capability coverage and integration fit rather than conceptual completeness, which increases the value of sharply defined interface boundaries and explicit ownership metadata in the feature registry [1, 2]. In combination, these studies support a concrete implication of the framework: as load and organizational coupling increase, governance and observability move from "nice-to-have" properties to engineering constraints that shape refresh control, versioning discipline, and geo-distributed placement decisions [5, 6, 9, 10].

This study is based on an analytical synthesis of published systems, surveys, and peer-reviewed evidence rather than on an original benchmark or production incident dataset. As a result, the framework formalizes design levers and trade-offs but does not provide quantitative effect sizes for specific workloads, hardware profiles, or cloud environments. Source selection can bias emphasis toward well-documented architectures (managed systems, widely cited pipelines, and representative surveys) while under-reporting undocumented operational patterns from proprietary deployments. The framework focuses on high-load correctness and performance mechanics (materialization, temporal extraction, refresh control, geo-distribution). It therefore treats adjacent concerns—such as security threat modeling, privacy-preserving feature computation, and organizational change management—only to the extent they intersect with the governance controls described in the surveyed literature. Finally, the paper abstracts away from vendor-specific implementations, so practitioners should validate the decision checklist against their concrete constraints (data locality rules, tail-latency targets, retraining cadence, and storage cost envelopes) before committing to a platform choice.

5. Conclusion

A coherent conceptual organization of feature stores for high-load machine learning systems relies on a layered architecture, comprising registry and metadata, reproducible transformations, dual materialization for offline and online use, and stable serving interfaces. In this work, the organization is presented as an analytical framework: a stable conceptual decomposition paired with decision levers that connect correctness semantics and performance constraints to concrete synchronization, storage, and maintenance policies in high-load production environments.

Correctness in production depends on explicit synchronization and temporal semantics for training extraction, so point-in-time correctness and lifecycle-aware versioning must be treated as architectural requirements rather than implementation details. High-load efficiency is achieved by combining materialization choices with execution optimizations for time-aware joins and by adopting maintenance strategies that reflect downstream sensitivity, rather than fixed refresh schedules. Linking these design decisions to MLOps operational practices yields a practical framework: feature stores serve as a governed data backbone, whose effectiveness depends on observability, disciplined releases of feature definitions, and integration with monitoring and orchestration processes throughout the ML lifecycle.

References

[1] Berberi, L., Kozlov, V. I., Nguyen, G., Sáinz-Pardo Díaz, J., Calatrava, A., Moltó, G., Tran, V., & López García, Á. (2025). Machine learning operations landscape: platforms and tools. Artificial Intelligence Review, 58, Article 167. https://doi.org/10.1007/s10462-025-11164-3

[2] de la Rúa Martínez, J., Buso, F., Kouzoupis, A., Ormenisan, A. A., Niazi, S., Bzhalava, D., Mak, K., Jouffrey, V., Ronström, M., Cunningham, R., Zangis, R., Mukhedkar, D., Khazanchi, A., Vlassov, V., & Dowling, J. (2024). The Hopsworks Feature Store for Machine Learning. In Companion of the 2024 International Conference on Management of Data (SIGMOD/PODS '24) (pp. 135–147). ACM. https://doi.org/10.1145/3626246.3653389

[3] Fang, Z., Yi, Y., Zhang, J., Liu, Y., Mu, Y., Lu, Q., Xu, X., Wang, J., Wang, C., Zhang, S., & Chen, S.

(2023). MLOps spanning whole machine learning life cycle: A survey (arXiv:2304.07296). arXiv.

[4] Hewage, N., & Meedeniya, D. (2022). Machine learning operations: A survey on MLOps tool support (arXiv:2202.10169). arXiv.

[5] Kreuzberger, D., Kühl, N., & Hirschl, S. (2022). Machine learning operations (MLOps): Overview, definition, and architecture (arXiv:2205.02302). arXiv

[6] Li, A., Ranganathan, B., Pan, F., Zhang, M., Xu, Q., Li, R., Raman, S., Shah, S. P., & Tang, V. (2023). Managed geo-distributed feature store: Architecture and system design (arXiv:2305.20077). arXiv.

[7] Liu, R., Park, K., Psallidas, F., Zhu, X., Mo, J., Sen, R., Interlandi, M., Karanasos, K., Tian, Y., & Camacho-Rodríguez, J. (2023). Optimizing data pipelines for machine learning in feature stores. Proceedings of the VLDB Endowment, 16, 4230–4239. https://doi.org/10.14778/3625054.3625060

[8] Orr, L., Sanyal, A., Ling, X., Goel, K., & Leszczynski, M. (2021). Managing ML pipelines: Feature stores and the coming wave of embedding ecosystems (arXiv:2108.05053). arXiv.

[9] Wooders, S., Mo, X., Narang, A., Lin, K., Stoica, I., Hellerstein, J. M., Crooks, N., & Gonzalez, J. E. (2023). RALF: Accuracy-aware scheduling for feature store maintenance. Proceedings of the VLDB Endowment, 17(3), 563–576. https://doi.org/10.14778/3632093.3632116

[10] Zarour, M., Alzabut, H., & Al-Sarayreh, K. T. (2025). MLOps best practices, challenges and maturity models: A systematic literature review. Information and Software Technology, 183, 107733. https://doi.org/10.1016/j.infsof.2025.107733

[11] Kovalchuk A. A comprehensive model of business consulting for small and medium-sized enterprises. - K.: Vidavnichy house "Internauka", 2025. - 98 p.