International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 11-No.4 (2025) pp. 9896-9902
http://www.ijcesen.com

ISSN: 2149-9144

Research Article

A Formal Model for Feature Store Architecture and Governance

Sivaramakrishnan Vaidyanathan*

University of Cincinnati, USA
* **Corresponding Author Email:** svaidyanathann@gmail.com- **ORCID:** 0000-0002-1147-0850

Article Info:

DOI: 10.22399/ijcesen.4555
Received : 17 December 2025
Revised : 21 December 2025
Accepted : 24 December 2025

Keywords

Feature Store Architecture,
Training-Serving Skew,
Machine Learning Operations,
Feature Governance,
Optimization Framework

Abstract:

ML systems in production have to address many challenges while ensuring consistency between the features in the training and serving phases. Feature Stores have emerged as one of the key ML infrastructure components to bridge the training and serving gaps. There are tradeoffs between different types of FS, such as latency, consistency guarantees, costs, and operational complexity. Organizations often do not have formal governance frameworks for governing Machine Learning pipelines. One example of the issues that can arise from insufficient frameworks is Training-Serving Skew, whereby feature statistics differ between environments. This leads to challenges in ensuring regulatory compliance and the ability to trace the lineage of features for model auditability and reproducibility. This presents a two-part formal model that enables mathematical optimization and structured governance. The first half frames the FS selection process as a constrained optimisation problem so that the performance of dual-database architectures can be quantitatively compared to that of unified architectures based on business priorities. The second half introduces Versioned Feature Descriptors that are canonical metadata artifacts for the permanent storage of feature definitions, complete lineage from raw data to prediction outputs, and fully machine-enforceable compliance policies. The optimization framework models serving latency, consistency gap, capital expense, and operational complexity for dual-database systems (one for online and another for offline workloads) and for unified systems (which house both workloads). The governance model prevents training-serving skew through runtime validation, ensuring that features input to a deployed model come from the desired descriptor version. Privacy and retention requirements are enforced by formal policy predicates, with the review process showing improvements in operational cost, debugging, audit, and regulatory compliance efforts. The framework formalizes Feature Store architecture evaluation, transforming decision-making from heuristic-based to a systematic architecture evaluation approach based on quantitative analysis for scalable and compliant machine learning adoption.

1. Introduction: The Need for Formal Feature Store Design

Feature stores allow organizations to enforce a standardized way to manage features both during offline training as well as online inference. The workload to maintain ML systems in the production stage of the machine learning (ML) lifecycle is far greater than during the training phase. Indeed, the technical debt in ML systems can grow quickly when building an infrastructure for ML models without an understanding of how the model will operate in production [1]. Additionally, production ML systems are complex because there are many tightly-coupled dependencies between the data pipeline, feature transformations, and the serving infrastructure. There's not a production-ready, systematic approach to architecting feature stores, which are complex systems that bridge the training-serving gap. In existing production ML systems, most decisions about the feature store architecture are made through ad-hoc evaluations, leading to two primary systemic challenges.Another common problem in production systems is Training-Serving Skew (TSS), where the statistical properties of the training data used are different from those of the serving data, which renders offline metrics useless to judge model performance. This is due to mismatches in data transformation logic and in the data synchronization latency between the batch and

real-time serving environments. Even a small mismatch in computations can lead to poor performance of the model in production [1]. TSS can introduce technical debt in many ways. Some examples are hidden feedback loops where the model's effect on the generated data is not accounted for, and undocumented consumers of the feature outputs that induce brittle dependencies across the boundaries of the system. In production machine learning, data lifecycle management efforts have shown that validating and monitoring the feature pipeline is one of the most unstable parts of the ML model performance, as maintaining its consistency requirements across different compute environments is challenging [2]. Cost and operational considerations also interact with prediction accuracy, for example, the delay between iterations for retraining models, the validity of A/B testing, and the credibility of the ML systems.Governance problems are further compounded by technical problems and introduce regulatory, compliance, and business risks, which can weaken the organization's viability. Organizations that fail to adopt a disciplined and formalized manner of managing their features cannot meet their obligations under the law, including GDPR and CCPA, which demand strong auditability and data lineage [2]. A lack of governance results in the inability to perform timely audits, with increasing technical debt as engineering teams spend time manually reconciling or debugging features. By expressing feature store design as an optimization problem, engineers can select an architecture based on trade-offs and apply a consistent framework for governance.

2. Feature Store Architecture and Trade-offs

Feature Stores can generally be classified into two categories based on how they handle the Consistency, Availability, and Partition Tolerance trade-offs intrinsic in distributed systems. As machine learning systems have matured, better architectures have been created to help deal with both the analytical workloads of training machine learning models and the transactional workloads of serving time-sensitive predictions [3]. These abstractions lead to trade-offs in terms of latency, consistency, and operational complexity, all of which play a role when deploying across a wide range of computing fabrics. The dual-database and unified model are respectively a trade-off along the axis of how to combine these benefits into a unified architecture, and where to apply them along the axis of performance.The two-database architecture has since become the industry standard, consisting of two databases: an Offline Store for high-throughput batch-read queries over historical data, and an Online Store for low-latency point lookup over real-time inference requests [3]. The motivation for this separation is the different workloads. During training, the features are calculated over a large amount of historical data to calculate aggregate statistics and trends. During serving, they are mostly retrieved through looking up individual feature vectors and have deterministic latency requirements. This leads to some operational trade-offs: although the system provides near-optimal serving latency with data stores specialized for different access patterns, transitioning updates from offline to online leads to a window of temporal inconsistency. This is because feature updates are propagated to the data stores via data synchronization pipelines asynchronously. Running distributed transactions and cross-database ETL stages also places a burden on the operational side of the system, requiring advanced orchestration for data quality guarantees and failure recovery.In contrast, the unified model relies on a single database model capable of servicing disparate batch and low-latency access patterns (e.g., Hybrid Transactional/Analytical Processing databases [4]) to satisfy the analytical/transactional gap in a single storage and query access scheme, and potentially additional optimizations such as columnar layouts with row-based access paths, caching, materialization, etc., to accommodate high-performance query workloads. In practice, throughout the MLflow platform, it is found that reducing the number of distinct systems used in the feature pipeline can greatly reduce operational complexity and improve the reproducibility of results. The more integration points in the pipeline, the greater the chances of subtle bugs [4]. By unifying the pipeline, the consistency gap is reduced to having one source of truth and no more complex sync logic to worry about. The pay for some increased resource consumption or worse tail latency during peak serving times. There are cases when OLTP and OLAP workloads can contend for the same set of resources. Care must be taken during capacity planning to avoid long batch processing jobs interfering with latency-sensitive serving requests.

3. The Formal Optimization Model

The Feature Store deployment objective function can be formulated as a constrained optimization through a Mixed Integer Program (MIP) that minimizes the Total Operational Cost, subject to the business-level performance constraints. The resource allocation decision in distributed ML systems has been well studied in the literature,

where it is observed that optimization frameworks can be a powerful tool to provide principled architectural design decisions in the face of conflicting objectives [5]. By capturing the design of the feature store system as an optimization problem, the equipment enterprises with quantitative analytical frameworks that can reason about trade-offs between latency requirements, consistency guarantees, capital expenditure, and operational complexity, and compare the performance of various feature store architectural alternatives.Let A be the Feature Store architecture space. Each architecture a is defined by a tuple of measurable operational properties, including serving latency L (latency to read feature values to perform inference), consistency gap T (the time difference between the moment that a feature value is written to the time that is available to the upper tier storage), capital and operating cost Cost (infrastructure and personnel cost), and operational complexity P. These properties are all optimized in a weight-sum objective function [5]. The weights for each of the components can be tuned for specific use cases and requirements, such as low-latency updates for financial trading applications or batch processing for cost-effective recommendation system updates. Gradient-increasing machine learning algorithms, which are very successful, have shown that weighted objective functions are well-suited to represent trade-offs, and the same is true for optimizing infrastructure architectures [5].Such constraints could be on parameters such as latency, consistency, or other service-level agreements that the application semantics do not allow. Model management experience with production practice shows that the relationship between architecture parameters and such performance constraints is typically non-linear; for example, a slightly different architecture can lead to considerably different tail latency or consistency bounds [6]. To measure the cost of each constraint surface, the measure of behavior of the two-database systems for these workloads is considered. The component functions of the constraint surfaces are defined as follows: serving latency of read is the access time of the online store plus the network communication time, and the consistency gap is the pipeline processing time plus the write propagation time in the database replicas. The total cost in the infrastructure model is the sum of the costs of online storage, offline storage, the synchronization pipeline infrastructure, and the personnel overhead for the three systems [6].For all the unified systems, the component functions have the following simplified forms: a unified system's serving latency is its read latency, a distributed system's consistency gap is its internal replication delay, and the cost of a unified system is the cost of its underlying infrastructure platform. The definition of the component functions allows for a quantitative substitution of architectural parameters into the objective function of the model, so that trade-offs can be computed based on possible values of the parameters, which are estimated or measured. Machine learning model management literature stresses that auditability and reproducibility require explicit specification of system configurations. The formal optimization model provides exactly this rigor in choosing a feature store architecture [6]. The optimization model is flexible and can incorporate additional constraints or objectives as organizational requirements change, such as adding data sovereignty constraints requiring certain features to be stored in certain geographies, or adding energy efficiency objectives with sustainability-focused organizations.

4. Formal Feature Governance Model

Regardless of the architecture, governance mechanisms are needed to control data drift, enable auditability, and ensure reproducibility at each step of the learning process. The difficulties organizations experience in deploying machine learning systems at scale are not rooted in a lack of algorithmic approaches, but instead in the lack of infrastructure to track data provenance, feature evolution, and access policies across heterogeneous computational environments [7]. The implementation for extending existing software engineering principles of application development to machine learning, where the feedback loop between code, data, and model behavior creates a complex dependency graph that requires dedicated governance structures. To resolve the problems of lineage compliance and regulatory traceability in production environments that serve multiple models and teams, the implementation of a Versioned Feature Descriptor-based model is required.The Versioned Feature Descriptor is the canonical definition of a feature or feature group. It is never updated, and it resides in the centralized Feature Registry that is available to all components of the ML ecosystem [7]. A feature definition includes the feature's human-readable name, version identifier, source data identifiers, a transformation code cryptographic hash, and input parameters, such as configuration values that change the feature's behavior. Lineage enforcement ensures that every feature value served in production retains its lineage to the VFD that computes it. A complete lineage from raw source data to feature value served by a model enables debugging of production

model behavior, allowing teams to build a precise pipeline of computation performed in order to produce a model prediction in the past. Teams at Microsoft operating at scale with machine learning have seen that managing versions and lineage tracking reduces debugging time and increases confidence in model behavior, as engineers can determine the contributing features for an unexpected prediction [8].This system of formalizing training datasets is based on treating them as snapshots of features as they were in a particular moment in the history of versioning of the VFD. If organizations want to reproduce both the training of their models on previously recorded data or the results of their experiments, the VFD versioning can recreate those [8]. In addition to time-stamped data retrieval, this enforces complete specification of transformation logic, library dependencies, and configuration parameters for a feature's semantics. The consistency and compliance constraints based on VFDs provide two key operational functions. To prevent serving-model skew, the runtime performs validation checks when serving models that enforce that only features associated with the VFDs a model trained on are sent to the model and rejects requests that would result in skew. Feature metadata is tagged with attributes, such as privacy labels, retention time, and who can access, and a set of predicates is defined that specify which services can access these features given their compliance posture. In particular, research into machine learning system deployment has identified that differences between training and serving environments are a key cause for machine learning system failures in production, and formal governance mechanisms are effective for systematically addressing these failure modes [7]. The machine-enforceable policy layer allows organizations to show compliance with data protection regulations via audit trails at the level of which data assets were accessed by which services at which times, and ensures that policies cannot be circumvented by ad-hoc access paths [8].

5. Future Work and Empirical Validation

The next steps are in the empirical validation of the proposed optimization model through comparative case studies on systems with varying workloads and quality-of-service requirements. In addition to the careful theoretical development of the model proposed here, further work is required to empirically validate the suitability of the completed approach across a wide variety of deployment scenarios ranging from latency-critical to cost-efficient batch processing [9]. Subsequent longitudinal studies will research deployments of feature stores over longer time frames and investigate the effects of architectural choices on model accuracy, infrastructure costs, incident rates, and time-to-production for machine learning teams. In addition to performance measurements, quantifiable metrics will be combined with qualitative assessments of operational experience to measure technical performance and usability in operational environments.The empirical research program will investigate these deployment scenarios that represent common patterns of use in production ML [9]. For example, low-latency fraud detection systems serve use cases where serving latency is the dominant constraint and are used for real-time predictions that increase the accuracy of fraud detection without a negative impact on user experience. Latency-sensitive applications are generally high-traffic systems with access patterns, data locality, caching, and other optimizations taken into consideration. Sub-10 millisecond latency budgets are typically required. Cost-sensitive recommendation engines describe a different kind of system: the infrastructure needs to be cost-efficient because of high user populations, but the wide latency budgets that are possible mean that the infrastructure does not need to be so performance-sensitive. The comparison of these different uses will indicate the ability of the optimization framework to adapt to different objective weightings and constraints, and therefore its generalizability.Future developments include finding ways to automatically calibrate the weights using machine learning techniques based on past data collected from the system (e.g., past telemetry data collected from the system and its correlation with business metrics). The parent machine learning problem, known as an ensemble, may be useful for this, in which multiple models or data sources are combined based on their observed performance [10]. Likewise, learned weight configurations from the use of a real system can be reused without explicitly assigning weights. Early results show that supervised learning methods can accurately predict good architecture configurations using features such as query workloads, amount of data that is processed, size of teams, UI and different industry verticals that are served by the architecture type. However, more research is needed in order to validate this trend across organizations. [10] Automated weight calibration would turn the optimizer from being human-centric to being self-calibrating and adapting the optimization decisions based on the experiences learned from recent deployments. Completing this validation work will show the practical feasibility of this proposal's fundamental mathematical theory in a production setting with large numbers of

features, multi-jurisdictional compliance, and the heavy computational load of thousands of simultaneous requests. This will make the formal optimization of algorithms a standard practice in the design of MLOps infrastructure.

Table 1: *Feature Store Architecture Comparison [3, 4]*

Architecture Type	Latency Characteristics	Consistency Properties	Operational Complexity	Cost Structure
Dual-Database	Minimal serving latency through specialized online stores	Temporal consistency gap due to asynchronous synchronization	High complexity with multiple systems and ETL pipelines	Separate costs for online storage, offline storage, and synchronization infrastructure
Unified Platform	Moderate serving latency with a single system handling diverse workloads	Near-zero consistency gap with a single source of truth	Reduced complexity with consolidated infrastructure	Integrated cost structure with premium compute requirements
Dual-Database Components	Online store optimized for point lookups, offline store for batch analysis	Pipeline processing delay plus write propagation lag	Distributed transaction management across databases	Personnel overhead for maintaining distinct systems
Unified Platform Components	HTAP capabilities reconciling transactional and analytical patterns	Internal replication delay approaching minimal values	Simplified operations with fewer integration points	Resource contention during concurrent workload execution

Table 2: *Formal Optimization Model Components [5, 6]*

Component	Symbol	Definition	Dual-Database Formulation	Unified Platform Formulation
Serving Latency	L(a)	Time required for feature retrieval during inference	Online store access time plus network transmission delay	Single platform read performance characteristics
Consistency Gap	T(a)	Temporal lag in feature availability across storage tiers	Pipeline processing duration plus write propagation lag	Internal replication delay within the distributed system
Operational Cost	Cost(a)	Capital and operational expenditure for infrastructure	Sum of online storage, offline storage, and pipeline costs	Integrated platform expense with unified infrastructure
Operational Complexity	P(a)	Management burden and personnel requirements	Multiple systems with synchronization orchestration	Consolidated platform with simplified maintenance
Objective Weights	w_i	Business-critical priority coefficients	Application-specific weight assignments reflecting domain requirements	Identical weighting mechanism adapted to use case priorities
Constraint Thresholds	L_max, T_max	Maximum acceptable performance bounds	Service level agreements defining acceptable limits	Same constraint framework ensuring business requirements

Table 3: *Versioned Feature Descriptor Structure [7, 8]*

VFD Component	Purpose	Governance Function	Implementation Mechanism
Feature Name	Human-readable identifier for feature reference	Enables team communication and documentation	Unique naming convention across the organization

Version Identifier	Temporal tracking of definition evolution	Supports reproducibility and historical analysis	Semantic versioning with immutable history
Source Data Identifier	Links to upstream data assets	Establishes data provenance and lineage	References to cataloged data sources
Transformation Code Hash	Cryptographic verification of implementation	Ensures computational fidelity and auditability	SHA-256 hash of transformation logic
Input Parameters	Configuration values affecting computation	Documents execution context for reproducibility	Structured metadata with parameter specifications
Lineage Enforcement	Complete traceability chain	Debugging and root cause analysis capabilities	Automated linkage from predictions to source data
Compliance Metadata	Privacy classifications and retention policies	Regulatory adherence and access control	Machine-enforceable policy predicates

Table 4: *Training-Serving Skew Prevention Mechanisms [9, 10]*

Mechanism	Function	Enforcement Method	Operational Impact
VFD Version Validation	Ensures deployed models receive features from training-compatible descriptors	Runtime checks comparing the current VFD against model requirements	Prevents version mismatches, causing prediction degradation
Automated Blocking	Rejects serving requests with incompatible feature versions	Policy predicate evaluation before feature retrieval	Maintains model integrity with minimal latency overhead
Feature Registry	Centralized repository of canonical feature definitions	Single source of truth accessible across infrastructure	Eliminates ambiguity in feature specifications
Reproducibility Snapshots	Training datasets as frozen feature versions	Immutable dataset definitions with VFD references	Enables exact reconstruction of training environments
Compliance Policy Predicates	Formal access control based on privacy classifications	Automated evaluation of service permissions	Machine-verifiable regulatory adherence
Audit Trail Generation	Comprehensive logging of feature access patterns	Detailed records linking services to data consumption	Supports regulatory reporting and incident investigation

4. Conclusions

A central component of ML infrastructure is the Feature Store, the glue between offline feature engineering and production model serving. Designing a Feature Store is fundamentally a quantitative problem, not a heuristic one. The mathematical optimization model that captures latency, consistency guarantees, cost, and operational complexity to ease systematic evaluation of a Feature Store architecture. An organization can choose whether to use two different databases to separate online and offline storage or a single database to interleave these workloads under business priorities and performance constraints in an optimal manner. The mathematical description is flexible enough to capture latency-sensitive applications, such as financial applications, and cost-sensitive applications, such as recommender systems. The Versioned Feature Descriptor (VFD) governance framework, together with the optimization model, solves some of the oldest problems in production machine learning: data lineage, training-serving skew, and regulatory compliance. The cryptographic immutability of these definitions allows a VFD system to enable full end-to-end traceability from data sources through data transformations and finally to the predictions of machine learning models. Organizations that have adopted VFD-based governance have reported that they have dramatically reduced the time spent debugging, preparing for compliance audits, and fixing inconsistent features. VFDs also offer mechanisms for complying with privacy and data protection regulations in a machine-verifiable way. For example, organizations can show compliance with data retention and privacy classification requirements. As machine learning systems grow and organizations amass tens of thousands of features across hundreds of models, mathematically formal approaches to feature infrastructure become critical. Introducing the optimization and governance model as a formal mathematical system allows treating Feature Store management as a mathematically formal system with formal metrics for feature store architecture evaluation and formal metrics for feature data lifecycle governance.

Validation in other production contexts will further establish formal methods as best practices for enabling sustainable, compliant, and secure machine learning at scale.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

References

[1] D. Sculley et al., "Hidden Technical Debt in Machine Learning Systems". Available: https://proceedings.neurips.cc/paper_files/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf

[2] Neoklis Polyzotis et al., "Data Lifecycle Challenges in Production Machine Learning: A Survey," ACM SIGMOD Record, 2018. Available: https://dl.acm.org/doi/10.1145/3299887.3299891

[3] Alexander Ratner et al., "MLSys: The New Frontier of Machine Learning Systems," arXiv preprint arXiv:1904.03257, 2019. Available: https://arxiv.org/abs/1904.03257

[4] Matei Zaharia et al., "Accelerating the Machine Learning Lifecycle with MLflow," Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2018. Available: https://people.eecs.berkeley.edu/~matei/papers/2018/ieee_mlflow.pdf

[5] Tianqi Chen and Carlos Guestrin, "XGBoost: A Scalable Tree Boosting System," arxiv>cs>arXiv:1603.02754, 2016. Available: https://arxiv.org/abs/1603.02754

[6] Sebastian Schelter, "On Challenges in Machine Learning Model Management," Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2018. Available: http://sites.computer.org/debull/A18dec/p5.pdf

[7] Andrei Paleyes et al., "Challenges in Deploying Machine Learning: A Survey of Case Studies," ACM Computing Surveys, 2022. Available: https://dl.acm.org/doi/10.1145/3533378

[8] Saleema Amershi et al., "Software Engineering for Machine Learning: A Case Study," 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 2019. Available: https://ieeexplore.ieee.org/document/8804457

[9] Christoph Molnar et al., "Interpretable Machine Learning -- A Brief History, State-of-the-Art and Challenges," arXiv preprint arXiv:2010.09337, 2020. Available: https://arxiv.org/abs/2010.09337

[10] R. Maclin and D. Opitz, "Popular Ensemble Methods: An Empirical Study," Journal Of Artificial Intelligence Research, 2011. Available: https://arxiv.org/abs/1106.0257