Sciforce

Journal of Artificial Intelligence and Machine Learning

Journal homepage: www.sciforce.org
ISSN : 2995-2336

Open Access Research Article

Feature Serving at Scale: Design, Migration, and Empirical Evaluation of a Production Feature Platform

Karthik Perikala

Technology Leader, The Home Depot., United States

Abstract

Modern machine learning systems depend on large, continuously evolving feature sets to support both offlinetrainingandlow-latencyonlineinference. As feature ecosystems grow, bespoke pipelines and tightly coupled application logic becomeincreas- ingly difficult to operate, leading to inconsistent fea- ture definitions, operational fragility, and slow iter- ation cycles.

This paper presents the design and evolution of a production feature serving platform, beginning with an in-house NoSQL-based Feature Mart and fol- lowed by a migration to a managed Feature Store. Thein-housesystemprovidedflexibleschemadesign and low-latency access but required substantial en- gineering effort to maintain ingestion pipelines, fea- ture registries, refresh workflows, and serving mi- croservices.

The managed platform introduced automated in- gestion from analytical sources, built-in feature gov- ernance, and native serving interfaces, fundamen- tally shifting the operational model of feature lifecy- cle management.Using normalized empirical mea- surements, we evaluate latency percentiles, through- put stability, and execution behavior under live con- figuration updates.

Our results show that managed feature stores can preserve predictable serving performance while sig- nificantly reducing operational complexity, making them a compelling foundation for large-scale pro- duction machine learning systems.

Keywords: feature serving, feature store, ma- chine learning systems, NoSQL, production ML, low-latency inference

Introduction

Features form the critical interface between raw data and machine learning models.In production envi- ronments, features are derived from heterogeneous datasources, refreshed at varyingcadences, andcon- sumed by multiplemodelsacrossoffline trainingand online inference workflows. Ensuring consistency, freshness, and low-latency access to these features is a persistent systems challenge.

Early production deployments often manage fea-tures through ad-hoc data pipelines and application- specific retrieval logic.While effective at small scale, these approaches become brittle as feature counts and model consumers grow. Feature compu- tation logic is duplicated, refresh schedules diverge, and mismatches between training and serving data lead to degraded model performance and difficult-to-debug failures.

To address these challenges, many organizations adopt centralized Feature Marts that decouple fea- ture storage and retrieval from application services. In-house Feature Marts, commonly built on scal-able NoSQL data stores, offer high throughput and low-latency serving while allowing teams to tailor schemas, keys, and access patterns to specific work- loads.

However, operating such systems atscaleintro- ducessignificant complexity around dataingestion pipelines, schema governance, refresh orchestration, and serving API maintenance. Managed Feature Stores aim to abstract these concerns by providing standardized ingestion, automated refresh pipelines, built-infeatureregistries, and nativeonline serving interfaces.

In this paper, we present an experience-driven study of feature serving at scale. We detail the de- signofanin-house NoSQL-based Feature Mart, de-scribeits migration to amanaged Feature Store, and empiricallye valu atetheresulting system behavior. Our goal is to provide practical insights into fea- tureplatform design, migration strategies, and oper-ational trade-offs relevanttolarge-scale production machine learning systems.

Background

Feature Serving in Production ML Systems

As machine learning systems mature, features be- come shared assets consumed by multiple models and applications. Features must support both offline training and low-latency online inference, requiring strong guarantees around consistency, freshness, and availability.

Production feature access patterns arehetero- geneous. Some features are accessed at high fre- quency under strict latency budgets, while others are refreshed periodically and consumed in batch workflows.Supporting the sepatterns within asin- gle

Received date: February 15, 2026 **Accepted date:** February 22 2026; **Published date:** March 10, 2026

***Corresponding Author:** Perikala, K, Technology Leader, The Home Depot., United States,; E- mail: karthik.perikala2512@gmail.com

Copyright: © 2026 Perikala, K. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Citation: Perikala. K. (2026) Feature Serving at Scale: Design, Migration, and Empirical Evaluation of a Production Feature Platform. International Journal of Artificial intelligence and Machine Learning, 4(1), 1-7. https://doi.org/10.55124/jaim.v4i1.300

platform introduces system and operational com- plexity.

Maintaining alignment between offline and on- line feature values is critical. Even small discrep- ancies in computation logic or refresh cadencecan introduce training-serving skew and degrade model performance.

Limitations of Ad-Hoc Feature Pipelines

Early feature serving solutions rely on custom pipelines and application-specific retrieval logic. Feature computation and access are embedded di- rectly into training jobs or serving services, leadingto duplicated logic and fragmented ownership.

As feature ecosystems grow, these tightly cou- pled approaches become brittle. Schema changes require coordinated updates across systems, refresh workflows diverge, and operational failures can prop- agate across downstream consumers.

Centralized Feature Marts

Centralized Feature Marts decouple feature storage and retrieval from application logic and provide a shared abstraction for managing schemas, refresh workflows, and serving interfaces.

In-house Feature Marts, typically built on scal- able NoSQL data stores, offer fine-grained control over schema design and access patterns but place significant operational burden on platform owners.

Design Goals

The design of the feature serving platform is guided by system-level goals derived from operating large- scale production machine learning workloads. These goals inform both the architecture of the in-house Feature Mart and the evaluation of the managed Feature Store.

Decoupling Feature Lifecycle from Applications

Applications and models should not embed assump- tions about feature storage, refresh cadence, or re- trieval mechanisms. Externalizing these concerns into a centralized platform allows feature definitions and access semantics to evolve independently of ap- plication release cycles.

Consistency Across Offlineand Online Workflows

The platform should provide a single source of truth for feature computation logic and refresh seman-tics. This reduces training-serving skew and im- proves model reliability.

Operational Simplicity and Au-tomation

As feature ecosystems grow, manual management of ingestion pipelines and refresh workflows becomes unsustainable. The platform should automateinges- tion, validation, and refresh orchestration oreduce operational risk.

Low-Latencyand Scalable Serv-ing

Feature serving must satisfy strict latency re- quirements undersustained load. The serving path should scale horizontally while preserving pre- dictable tail latency.

Governance and Evolvability

The platform must support versioning, ownership, and controlled evolution so that feature definitions can change safely without disrupting downstream consumers.

In-House Feature Mart Ar-chitecture

The in-house Feature Mart was designed to provide low-latency online feature retrieval while supporting a rapidly growing set of features consumed by mul- tiple production models. The system decouples fea- turestorage from application services and exposes a centralized serving interface for online inference workloads.

At its core, the Feature Mart is built on a distributed NoSQL data store optimized for high- throughput reads and predictable access latency. Feature data is organized using carefully designed primary keys and column groupings to support effi- cient retrieval of feature vectors at inference time.

Feature Schema and Data Mod- eling

Each feature entity is represented as a logical row keyed by a composite identifier derived from entity type and entity ID. Featur evalues are stored ascol- umn qualifiers grouped by semantic category, allow- ing selective reads and efficient evolution of feature schemas over time.

This schema design enables backward- compatible feature additions and minimizes the need for disruptive schema migrations. However, it requires careful coordination to ensure consistent interpretation of feature semantics across producers and consumers.

Ingestion and Refresh Pipelines

Feature ingestion is implemented through custom data pipelines that transform source data into the Feature Mart schema. These pipelines handle vali- dation, aggregation, and normalization before writ- ing features to the online store.

To maintain feature freshness, refresh workflows are executed on scheduled intervals or triggered by upstream data availability. Pipeline orchestration and monitoring are maintained separately from the serving layer, increasing operational surface area as the number of features grows.

Serving Interfaceand Access Pat- terns

Online feature retrieval is exposed through dedi- cated serving microservices that translate inference requests into NoSQL read operations. These services enforce access contracts, apply basic validation, and return feature vectors to down stream model serving components.

While this approach provides flexibility and iso- lation, it introduces additional maintenance over- head. Serving services must be scaled, monitored, and versioned independently, and changes to feature schemas often require coordinated updates across pipelines and serving layers.

Operational Characteristics

Operating thein-house Feature Mart requires active management of ingestion pipelines, refreshsched- ules, schema evolution, and serving infrastructure. As feature counts and model consumers increase, coordination overhead and operational risk grow ac- cordingly.

These characteristics motivated exploration ofa managed Feature Store that could preserve low- latency serving behavior while reducing the opera- tional complexity of feature life cycle management.

Citation: Perikala. K. (2026) Feature Serving at Scale: Design, Migration, and Empirical Evaluation of a Production Feature Platform. International Journal of Artificial intelligence and Machine Learning, 4(1), 1-7. https://doi.org/10.55124/jaim.v4i1.300

Operational Characteristics

Operatingthein-house Feature Martrequiresactive management of ingestion pipelines, refreshsched- ules, schema evolution, and serving infrastructure. As feature counts and model consumers increase, coordination overhead and operational risk grow accordingly.

These characteristics motivated exploration ofa managed Feature Store that could preserve low- latency serving behavior while reducing the opera- tional complexity of feature life cycle management.

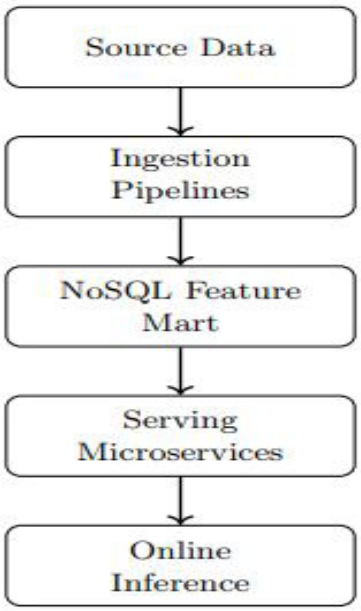

Figure 1: High-level architecture of the in-house NoSQL-based Feature Mart, illustrating custom in- gestion pipelines and dedicated serving microser- vices.

Managed Feature Store Ar- chitecture

The migration to a managed Feature Store was driven by the growing operational complexity of maintaining custom ingestion pipelines, feature reg- istries, and serving services in thein-house Feature Mart. While the underlying storage characteristics remained similar, the managed platform introduced a fundamentally different control plane for feature lifecycle management.

In the managed Feature Store, features are de- fined declaratively and sourced directly from analyt- ical datasystems. Feature definitions capture com- putation logic, refresh cadence, and entity associa- tions, enabling the platform to automate ingestion and materialization without custom pipeline devel- opment.

Automated Ingestion and Re-fresh

Unlike thein-house system, where refresh workflows were implemented and orchestrated manually, the managed Feature Store provides automate dinges- tion pipelines. Feature data is continuously or peri- odically materialized based on declared refresh poli- cies, with the platform handling scheduling, backfills, and retries.

This automation significantly reduces opera- tional overhead and minimizes the risk of stale or inconsistent features. Platform-managed ingestion also centralizes monitoring and failure handling, im- proving observability across feature refresh workflows.

Built-in Feature Registry

The managed Feature Store exposes a first-class fea- tureregistry that tracks feature definitions, schemas, ownership, and usage metadata.This registry re- placesad-hoc documentation and custom metadata services used in the in-house Feature Mart.

By providing a single source of truth for feature metadata, the platform improves discoverability, en- forces governance policies, and simplifies coordina- tion across feature producers and consumers.

Native Online Serving Interfaces

Online feature retrieval is exposed through native serving interfaces provided by the managed plat- form.These interface eseliminate the need for custom serving microservices, reducing maintenance burden and standardizing access patterns for downstream model serving systems.

Serving requests are resolved against the active feature definitions and materialized data, ensuring consistent behavior across consumers. Execution paths are optimized for low-latency access while re- maining isolated from control-plane activities such as schema updates or refresh orchestration.

Migration Strategy

Migration from the in-house Feature Mart to the managed Feature Store was performed in astaged manner. Feature definitions were incrementally re- expressed in the managed platform, and dual reads were used to validate correctness and latency behav- ior during transition periods.

Once parity was established, traffic was gradu- ally shifted to the managed Feature Store, allowing rollback in the event of anomalies. This approach enabled a smooth transition without disrupting pro- duction inference workloads.

Citation: Perikala. K. (2026) Feature Serving at Scale: Design, Migration, and Empirical Evaluation of a Production Feature Platform. International Journal of Artificial intelligence and Machine Learning, 4(1), 1-7. https://doi.org/10.55124/jaim.v4i1.300

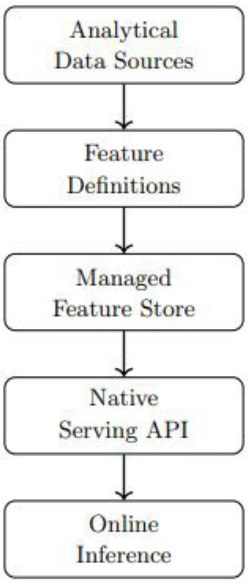

Figure 2: Managed Feature Store architecture illus- trating declarative feature definitions, automated in- gestion, and native online serving interfaces.

Migration Trade-offs

Migrating from an in-house NoSQL-based Feature Mart to a managed Feature Store involved a set of deliberate trade-offs spanning system control, opera- tional responsibility, and platform flexibility. While both systems support low-latency feature serving, their design philosophies and operational models dif- fer substantially.

Control Versus Abstraction

The in-house Feature Mart provided fine-grained control overs chema design, key composition, and re- fresh behavior.This flexibility enabled tailored opti- mizations for specific workloads but required contin- uous engineering investment to maintain correctness and performance as feature usage evolved.

In contrast, the managed Feature Store abstracts many of these decisions behind a standardized con- trol plane. While this reduces customization op- portunities, it significantly simplifies feature lifecy- cle management by enforcing consistent schemas, re- fresh semantics, and access patterns across teams.

Operational Ownership

Operating thein-house Feature Mart required main- tainingingestion pipelines, refresh schedulers, serv- ing microservices, and monitoring infrastructure. Failures in any of these components required manual diagnosis and remediation, increasing operational risk as system scale grew.

The managed Feature Store shifts much of this responsibility to the platform. Automate dingestion, built-in observability, and standardized serving in- terfacesreduce the operational surfaceareaandal- low teams to focus on feature definition rather than infrastructure maintenance.

Schema Evolution and Gover- nance

Schema evolution in the in-house system relied on coordination across pipeline and serving compo- nents.Introducing new features or modifying ex- isting ones required careful backward compatibility planning and synchronized deployments.

The managed Feature Store enforcesschema validation and versioning as first-class concepts. Feature definitions evolve through controlled up- dates, reducing the likelihood of accidental break-ingchanges and improving governance acrossf eature producers and consumers.

System Implications

The migration altered not only system architecture but also development and operational workflows. Feature development shifted from pipeline-centric engineering to declarative definition and validation, reducing iteration time and coordination over head.

Serving Path Characteristics

Both systems provide low-latency access suitable for online inference. However, the managed Feature Store removes the need for custom serving microser- vices, reducing deployment complexity and stan- dardizing access semantics across consuming sys- tems.

By resolving feature definitions and refresh logic prior to request execution, the managed serving path remains isolated from control-plane churn, preserv- ing predictable latency behavior under configuration updates.

Migration Risk and Validation

To mitigate migration risk, features were validated incrementally using parallel reads from both sys- tems. This dual-serving approach enabled correct- ness and latency comparison without disrupting pro- duction traffic.

Gradual traffic shifts and rollback-safe deploy- mentstrategies ensured that a nomalies could bede- tectedearly, maintaining service-level expectations throughout the transition.

Motivation for Empirical Evalua- tion

While architectural simplification and operational benefits are clear, it is critical to validate that these improvements do not come at the expense of serving performance. Then extsection presents a nempirical evaluation of latency stability, through put behavior, and execution isolation under representative produc- tion workloads.

Evaluation Methodology

The objective of this evaluation is to assess whether a managed feature serving platform can preserve predictable latency behavior, stable throughput, and operational isolation when compared to an in- ternally managed NoSQL-based feature serving sys- tem. The evaluation focuses on end-to-end serv- ing characteristics rather than feature semantics or model quality.

Serving Workload

Each online inference request retrieves multiple fea- ture groups that are resolved at request time. A single request fans out to several independent fea- ture sources, resulting in multiple concurrent read operations perinference. Under sustained load, the serving system processes hundreds of inference- quests per second, translating to several thousand internal feature retrieval

Citation: Perikala. K. (2026) Feature Serving at Scale: Design, Migration, and Empirical Evaluation of a Production Feature Platform. International Journal of Artificial intelligence and Machine Learning, 4(1), 1-7. https://doi.org/10.55124/jaim.v4i1.300

operations per second.

The workload includes a mix of lightweight lookups and moderately complex feature vectors, re- flecting production ranking, retrieval, and person- alization scenarios with varying degrees of request fan-out.

Measurement Approach

Latency is measured end-to-end from request ad- mission to featurere solution completion. Percentile statistics are collected under steady-state load and during controlled refresh and configuration update events. Report devalues are representative and nor- malized to preserve relative behavior while avoiding disclosure of sensitive deployment details.

Normalization and Confidential- ity

All reported metrics are normalized representations of observed system behavior. This normalization preserves relative latency ordering, tail behavior, and stability trends while ensuring that no propri-etary deployment details, absolute capacities, orin-ternal system characteristics are exposed.

Empirical Results

Serving Latency Characteristics

The first set of results summarizes tail latency be- havior under sustained load when each request re- solves multiple feature groups. Across evaluated scenarios, the managed feature serving platform ex- hibits higher tail latency than the in-house system, primarily due to additional coordination and service- layer abstraction introduced by managed execution.

Inference Scenario	Managed Serving	In-House Serving
Single-entity resolution	∼25–30 ms	∼18–20 ms
Multi-entity feature reso- lution	∼50–55 ms	∼30–36 ms

Despite higher absolute latency, tail behavior re- mains stable as request fan-out increases, indicating effective internal parallelization and bounded execu- tion variance.

Stability During Refresh and Configuration Changes

Latency behavior was evaluated before, during, and after live refresh and configuration updates. Both systems preserv eexecution isolation; however, the managed platform exhibits a slightly higher sensitiv-ity duringup date windows dueto background syn- chronization and metadata coordination.

Update Phase	Managed Serv- ing	In-House Serv- ing
Before update	∼baseline	∼baseline
During update	↑ slight–moderate increase	↑ slight increase
After update	∼baseline	∼baseline

Operational Observations

Under sustained load, both systems maintain low error rates and predictable behavior. The in-house serving system achieves lower tail latency through tighter integration, while the managed platform trades additional latency for standardized workflows and reduced operational overhead.

Latency variance remains bounded as request fan-out increases, confirming that parallel feature resolution and execution coordination scale effec- tively for high-throughput online feature serving.

Analysis and Design Impli- cations

The evolution from an in-house NoSQL-based Fea- ture Mart to a managed Feature Store highlights several system-level implication srelevant to large- scale feature serving platforms. While both ap- proaches can satisfy low-latency serving require- ments, their operational characteristics and design trade-offs differ substantially.

Centralization Versus Control

The in-house Feature Mart provided fine-grained control overs chema design, ingestionlogic, and serv- ing behavior. This control enabled workload-specific optimizations but required sustained engineering ef- fort to maintain correctness and performance as fea- ture usage evolved.

In contrast, the managed Feature Store central- izesmany of these responsibilities behinda platform- managed control plane. Although this reduces cus- tomization, it improves consistency across teams by enforcing standardized feature definitions, refresh semantics, and access patterns.

Operational Surface Area

Operating the in-house Feature Mart involved coor- dinating ingestion pipelines, serving micro services, and schema management processes.As feature ecosystems grew, this increased the operational sur- face area and the likelihood of configuration drift or partial failures.

The managed Feature Store reduces the number of independently operated components. Automated ingestion, built-in registries, and native serving in- terfaces simplify operational workflows and improve system reliability.

Latency Predictability Under Change

Despite differences in control-plane design, both sys- tems exhibit predictable latency behavior suitable for online inference. Empirical results demonstrate that abstracting feature life cycle management does not introduce measurable degradation in tail latency when execution paths are isolated from configura-tion dynamics.

Resolving feature definitions and refresh logic prior to request execution is therefore critical to maintaining stable serving behavior under continuous change.

Lessons Learned

Feature Lifecycle as a First- Class Concern

Treating features as first-class system entities with explicit definitions, ownership, and refresh seman- tics implifies development and improves consistency across models and applications.

Automation Is Essential for Scale

Manual ingestion and refresh workflows do not scale with growing feature ecosystems. Automating data materialization,

Citation: Perikala. K. (2026) Feature Serving at Scale: Design, Migration, and Empirical Evaluation of a Production Feature Platform. International Journal of Artificial intelligence and Machine Learning, 4(1), 1-7. https://doi.org/10.55124/jaim.v4i1.300

validation, and refresh scheduling is essential for sustaining reliable feature serving at scale.

Isolation Enables Operational Stability

Isolating serving paths from control-plane activities is critical for maintaining predictable latency during configuration changes. Ensuring that feature reso- lution occurs at request boundaries prevents partial updates and inconsistent execution.

Incremental Migration Reduces Risk

Incremental migration strategies, including dualreads and gradual traffic shifts, enable correctness and performance validation without disrupting pro- duction workloads.

Observability Isa Prerequisite for Trust

Reliable feature serving requires visibility into fea- ture freshness, access patterns, and serving latency. Centralized metrics and monitoring are essential for detecting staleness, diagnosing performance regres- sions, and validating system behavior during migra- tion and steady-state operation.

Abstraction Shifts Complexity

Managed platforms reduce the burden of operat-ing feature serving infrastructure but do not elim- inate complexity entirely. Instead, complexity shifts toward configuration, governance, and integration concerns, which must be addressed explicitly in system design.

Limitations

The empirical analysis presented in this paper is in-tentionally scoped to protect confidential and pro- prietary system details. Absolute latency values, in-fra structure configurations, and deployment-specific tuning parameters are therefore not disclosed. La- tency characteristics are reported using normalized representations that preserve relative behavior and comparative trends without exposing sensitive oper- ational information.

A key limitation of the current evaluation is the constrained set of data ingestion patterns exercised. While the managed feature store internally relies on NoSQL storage for online serving, all evaluated fea- ture data is sourced from analytics database tables and subsequently ingested and refreshed into the un- derlying NoSQL-backed serving layer.Alternative ingestion paths and heterogeneous upstream sources supported by earlier Feature Mart architectures are not evaluated in this study.

Real-time feature ingestion based on streaming raw signals is not explored in this evaluation. This omission is not due to limitations of the managed feature store, which support slow-latency ingestion mechanisms, but rather reflects the absenceofcur- rent production use cases requiring real-time feature materialization from streaming sources.

In addition, the system does not maintain inde- pendent feature representations for model training and online serving. Training pipelines consume fea- turesderived from the same analytics-backed defi- nitions that feed the feature storeingestion process. While this design simplifies consistency and gover- nance, it limits analysis of architectures that decou- ple training and serving feature materialization.

Finally, the evaluation reflects behavior observed under production-aligned access patterns and fea- ture compositions. Workloads involving substan- tially different feature fan-out, access locality, or cross-entity joins may exhibit different performance characteristics and warrant separate investigation.

Future Directions

Future work will focus on completing the migration of remaining Feature Mart feature tables and serv- ing workflows into managed feature views and fea- ture graphs, while continuing to rely on the feature store's internal NoSQL serving layer for low-latency access.

Additional directions involve extending feature graph support to enable richer dependency mod- eling, incremental refresh semantics, and declar- ative feature–feature relationships, while preserv-ing bounded latency guarantees for online inference workloads.

Future Directions

Future work will focus on completing the migration of remaining Feature Mart feature tables and serv- ing workflows into managed feature views and fea- ture graphs, while continuing to rely on the feature store's internal NoSQL serving layer for low-latency access.

Additional directions involve extending feature graph support to enable richer dependency mod- eling, incremental refresh semantics, and declar- ative feature–feature relationships, while preserv-ing bounded latency guarantees for online inference workloads.

Conclusion

This paper examined the design, migration, and em- pirical behavior of a production feature serving plat- form, tracingits evolution fromanin-house Feature Mart with explicit NoSQL management to a man- aged Feature Store that integrates analytics-backed ingestion with internally managed NoSQL serving.

Our results show that managed feature stores preserve predictable serving latency and stable throughput while significantly reducing operational complexity.Centralizing ingestion, refresh, and serving coordination simplifies consistency guaran- tees without introducing unbounded latency behav- ior.

Beyond performance, the migration streamlined development and operational workflows by eliminat- ing bespoke data pipelines and manually managed servingin frastructure, shifting efforttoward feature definition, validation, and governance. This tran- sition reduced coordination overhead and improved iteration velocity across feature producersandcon- sumers.

Overall, these findings underscore the value of treating feature platforms as first-class systems com- ponents. Aligning analytics-based ingestion with managed NoSQL serving abstractions enables scal- able, reliable, and operationally efficient machine learning systems at production scale.

References

1. D. Crankshaw et al., "Clipper:A Low-Latency Online Prediction Serving System,"NSDI, 2017.
2. M. Zaharia et al., "Accelerating the Machine Learning Life-cy clewithMLflow,"IEEEDataEngineeringBulletin,2018.
3. J. Dean and L. A. Barroso, "The Tail at Scale,"Communications of the ACM, 2013.
4. S. Kim et al., "Serving DNNs Like Clockwork:Perfor- mance Predictability from the Bottom Up,"OSDI, 2020.

Citation: Perikala. K. (2026) Feature Serving at Scale: Design, Migration, and Empirical Evaluation of a Production Feature Platform. International Journal of Artificial intelligence and Machine Learning, 4(1), 1-7. https://doi.org/10.55124/jaim.v4i1.300

5. J. Jiang et al., "Heterogeneity-Aware DNN Serving," ML-Sys, 2022.
6. P.Barhametal.,"MachineLearningSystemsareStuckin a Rut,"HotOS, 2019.
7. E. Breck et al., "The ML Test Score:A Rubric forML Production Readiness,"IEEE Software, 2017.
8. D. Narayanan et al., "Efficient Large-ScaleLanguage Model Inference,"MLSys, 2023.
9. E. Breck et al., "The ML Test Score:A Rubric forML Production Readiness,"IEEE Software, 2017.
10. D. Sculley et al., "Hidden Technical Debt in Machine Learning Systems,"NeurIPS, 2015.
11. M. Zaharia et al., "Accelerating the Machine LearningLifecycle with MLflow,"IEEE Data Engineering Bulletin,2018.
12. A. M. Ahmed et al., "Feature Stores:A DataManage- ment Perspective,"IEEE Data Engineering Bulletin, 2022.
13. M.Hellersteinetal.,"Ground:ADataContextService,"
14. CIDR,2017.
15. M. Abadi et al., "TensorFlow: A System for Large-Scale Machine Learning,"OSDI, 2016.
16. E.Brecketal.,"DataValidationforMachineLearning," SysML,2019.
18. M.Boehmetal., "SystemML:DeclarativeMachineLearn- ing on Big Data,"VLDB, 2018.
19. D. Crankshaw et al., "Clipper:A Low-Latency Online Prediction Serving System,"NSDI, 2017.
20. J. Dean and L. A. Barroso, "The Tail at Scale,"Communications of the ACM, 2013.
21. C. Zhang et al., "Deep Feature Synthesis for AutomatedFeature Engineering,"ICML, 2015.
22. A. Kumar et al., "Model Selection and HyperparameterOptimization in Machine Learning," Foundations and TrendsinMachineLearning, 2019.
23. J. Kreps et al., "Kafka: A Distributed Messaging System for Log Processing,"NetDB, 2011.
24. S. Melnik et al., "Dremel:Interactive Analysis of Web- Scale Datasets,"VLDB, 2010.
25. K. Hellerstein et al., "Serverless Computing:OneStep Forward, Two Steps Back,"CIDR, 2019.
26. P. Barham et al., "Machine Learning Systems areStuck in a Rut,"HotOS, 2019.

Citation: Perikala. K. (2026) Feature Serving at Scale: Design, Migration, and Empirical Evaluation of a Production Feature Platform. International Journal of Artificial intelligence and Machine Learning, 4(1), 1-7. https://doi.org/10.55124/jaim.v4i1.300