Journal of Information Systems Engineering and Management
2025, 10(63s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/
Research Article

The Infrastructure -Model Symbiosis: Rethinking Platform Architecture for Billion-Scale Prediction Workloads

Priyadharshini Krishnamurthy
Meta Platforms Inc., USA

ARTICLE INFO

Received: 05 Nov 2025

Revised: 18 Dec 2025

Accepted: 28 Dec 2025

ABSTRACT

The clean separation between ML model development and infrastructure engineering—a principle borrowed from traditional software—breaks down at billion-scale prediction workloads. At this scale, deployment constraints don't just affect performance; they determine whether a model can exist in production at all. This article introduces the concept of infrastructure-model symbiosis, demonstrating how production systems at hyperscale platforms achieve transformative performance improvements through co-design viewpoints that treat infrastructure as a first-class architectural constraint. Through detailed investigation of music streaming recommendations, product recommendation systems, and video streaming platforms, the article reveals that models optimized purely for offline accuracy metrics often fail catastrophically when confronted with production realities. The framework of infrastructure-aware model design encompasses evaluation criteria, including memory footprint, serialization overhead, distributed inference coordination costs, and cacheability. Architectural patterns such as two-tower neural networks, hierarchical model cascades, tiered feature materialization pipelines, and dynamic batching mechanisms demonstrate how alignment between model architecture and serving topology enables dramatic improvements in both performance and cost efficiency. Economic scrutiny establishes that optimal model selection requires balancing accuracy against serving costs at production scale, transforming model development from a purely technical optimization into a strategic resource allocation discipline. Feature infrastructure emerges as a critical bottleneck, consuming more computational resources than model inference itself, necessitating sophisticated materialization strategies and tiered caching hierarchies. Serving orchestration patterns reconcile contradictory requirements between batch efficiency and low-latency response through adaptive mechanisms that dynamically adjust to traffic patterns. The synthesis of these elements establishes infrastructure-model co-design as essential for advancing production machine learning systems beyond current barriers.

Keywords: Infrastructure-Model Symbiosis, Billion-Scale Prediction Workloads, Two-Tower Neural Networks, Feature Store Architecture, Dynamic Batching Orchestration

1. Introduction: The Collapse of Infrastructure Abstraction

The foundational premise of software engineering that infrastructure should be abstracted away from application logic fails completely for billion-scale machine learning systems. While this separation of concerns succeeds for stateless web applications and CRUD operations, it fundamentally misunderstands the physics of machine learning inference at scale. A model that achieves 0.1% better accuracy in offline evaluation may consume 10x more memory, require 5x longer serialization time, and generate 100x more cross-service network calls during inference. These are not implementation details to be optimized later; these constitute constraints that determine whether a model can exist in production at all. Crankshaw et al. has demonstrated that conventional architectures fail to address the millisecond-level latencies required for interactive applications, where prediction requests must be processed within strict time bounds [1].

In this article, I examine the emergence of infrastructure-model symbiosis through production systems at hyperscale platforms. The most successful ML platforms don't treat infrastructure as a neutral substrate. They treat it as a first-class design constraint that shapes model architecture from conception. The conventional workflow where data scientists develop models in isolation, then "throw them over the wall" to infrastructure engineers creates a barrier to deploying sophisticated ML systems at scale. Edge computing frameworks have revealed that hierarchical inference architectures can reduce end-to-end latency by partitioning models across computational tiers, though this requires careful consideration of network bandwidth and processing capabilities at each layer [2].

Consider a large-scale music recommendation system that must generate 40 million personalized playlists every Monday morning. Initial attempts to deploy state-of-the-art deep learning models, optimized purely for recommendation accuracy on research clusters, revealed production inference costs exceeding $10 million annually. The breakthrough came not from infrastructure optimization alone, nor from model improvements in isolation, but from co-designing both: adopting two-tower neural networks specifically engineered for distributed serving topology while implementing custom model serving infrastructure with aggressive caching and batching strategies. The resulting system delivers predictions 50x faster at one-tenth the original cost, a transformation impossible through either infrastructure or model optimization alone. Prediction serving frameworks achieving sub-millisecond query latencies have shown that adaptive batching mechanisms can increase throughput by 7x while maintaining latency service-level objectives [1].

This symbiotic relationship extends beyond cost optimization to constrain which algorithmic approaches can succeed. Product recommendation infrastructure processing billions of predictions daily across millions of products has evolved hierarchical serving tiers where lightweight models handle 90% of requests while complex ensemble models activate only for high-value decisions. This architecture emerged not from algorithmic preference but from infrastructure reality: network latency between microservices creates hard ceilings on model complexity that no amount of GPU acceleration can overcome. Coarse-to-fine inference strategies partition deep neural networks into early-exit branches and final layers, enabling resource-constrained edge devices to process simple inputs locally while offloading complex computations to cloud infrastructure, thereby reducing response times by 30-45% compared to monolithic deployment [2]. The framework I'm proposing—'infrastructure-aware model design'—evaluates models on deployment profile characteristics: memory footprint, serialization overhead, distributed inference coordination costs, and cacheability. Through detailed case studies and architectural patterns, this perspective transforms platform engineering from a reactive optimization problem into a proactive design discipline that shapes the entire ML development lifecycle.

Performance Metric	**Value**
Throughput increase via adaptive batching	7x
Prediction speed improvement	50x faster
Cost reduction ratio	1/10th
Response time reduction (coarse-to-fine)	30-45%
Weekly playlist generation volume	40 million
Annual infrastructure cost (initial)	$10 million
Inference time (initial model)	847 ms

Memory per replica (initial model)	3.2 GB
Sub-millisecond query latency achievement	Yes
Hierarchical tier request handling	90%

Table 1: Performance Metrics of Infrastructure-Aware ML Architectures [1,2]

2. The Economics of Inference: Why Production Costs Diverge from Research Metrics

The economic reality of production ML systems reveals a profound disconnect between research optimization objectives and operational constraints. Academic benchmarks and competitive platforms optimize for predictive accuracy under the implicit assumption that computational resources scale linearly with model complexity. Production systems at billion-scale prediction volumes operate under radically different physics, where the marginal cost of inference dominates the total cost of ownership and introduces non-linear relationships between model sophistication and operational viability. Infrastructure scheduling research has demonstrated that bandwidth-aware multi-interface systems can reduce energy consumption by 28.5% while maintaining delay constraints, revealing how network resource management fundamentally impacts operational costs in distributed prediction systems [3].

Real-world music streaming recommendation systems illuminate this divergence. Initial production candidates deep neural collaborative filtering models achieving state-of-the-art metrics on offline datasets required 847 milliseconds average inference time and consumed 3.2GB of memory per model replica. With 40 million users requiring weekly predictions, naive deployment approaches necessitate maintaining thousands of model replicas to handle peak load spikes, translating to infrastructure costs of $10.4 million annually for compute alone, excluding additional costs of feature materialization, model serving orchestration, and monitoring infrastructure. Edge-cloud frameworks for machine learning tasks have shown that cross-domain architectures can achieve accuracy improvements of 8-12% while reducing computational overhead through intelligent task partitioning between edge devices and cloud infrastructure [4]. The economic structure of cloud computing amplifies these costs through several mechanisms. First, memory-bound workloads cannot effectively utilize GPU acceleration, forcing deployment on expensive high-memory CPU instances with costs ranging from $0.096 to $0.192 per hour for instances with 64GB RAM. Second, peak load provisioning requires maintaining excess capacity for 95th percentile workloads that only materialize during specific time windows.

Third, cross-availability-zone network transfer costs accumulate rapidly when models require feature lookups across distributed feature stores, with each prediction potentially triggering dozens of inter-service calls at $0.01 per GB transferred.

Multi-interface scheduling strategies have proven effective in minimizing bandwidth consumption, with dynamic interface selection algorithms reducing data transmission costs by allocating traffic across WiFi, Bluetooth, and cellular networks based on real-time energy efficiency metrics [3].

The relationship between model complexity and inference cost exhibits super-linear scaling at production volumes. A model with 2x more parameters does not simply cost 2x more to serve; it may require different serving infrastructure entirely, crossing critical thresholds that trigger cascading cost increases. Models exceeding single-machine memory boundaries necessitate distributed inference coordination, introducing network latency averaging 2-5 milliseconds per hop and synchronization overhead consuming 15-30% of total inference time. Models with complex preprocessing pipelines may saturate feature store throughput, requiring expensive feature materialization infrastructure or forcing denormalization strategies that multiply storage costs by factors of 3-10x. Edge-cloud hybrid architectures demonstrate that offloading computational tasks to edge nodes can reduce response latency by 40-60% compared to pure cloud-based inference, though this requires sophisticated orchestration mechanisms to balance workload distribution [4].

Serialization and deserialization overhead of complex models introduces additional hidden costs. Modern deep learning frameworks store models in formats optimized for training efficiency rather than serving performance. Loading a 2GB TensorFlow SavedModel may require 10-15 seconds and temporarily consume 6GB of memory during deserialization constraints that fundamentally limit auto-scaling responsiveness and force maintaining excess replica capacity at 40-60% baseline utilization.

Architecture Characteristic	Value
Memory reduction (embedding-based)	40-70%
Complexity reduction (algorithmic)	$O(n^2)$ to $O(n \log n)$
Serialization overhead per dimension	4 bytes
Graph serialization overhead per layer	50-100 KB
Inference throughput improvement	50x
Data transfer per inference call	10-50 MB
Feature vector dimensions	128-256
Processing time reduction (preprocessing)	35-55%
Query response time reduction	20-40%
Model sharding capability	Yes

Table 2: Computational and Memory Performance of Two-Tower Neural Networks [5, 6]

3. Infrastructure-Aware Model Architecture: Engineering for Deployment from Inception

The infrastructural paradigm of model design flips the conventional ML development process to view deployment constraints as those that are directly addressed in the selection of a model architecture and no longer as a post-hoc optimization problem. This methodology acknowledges that some algorithmic schemes are characterized by favorably deployable characteristics and some with an inevitably complex complexity that cannot be wholly alleviated through infrastructure engineering solutions. Deep learning recommendation systems research has established that collaborative filtering architectures, neural matrix factorization methods, and attention-based models each present distinct computational profiles, with two-tower architectures demonstrating particular efficiency in large-scale serving environments through their decomposable structure [5].

Two-tower neural networks exemplify architecture choices specifically engineered for distributed serving topology. This design separates the model into user and item embedding towers that can be computed independently, with the final prediction requiring only a dot product between cached embeddings. This goes beyond computational efficiency: by eliminating the need for user-item cross-features during inference, two-tower models enable aggressive caching strategies where item embeddings are precomputed for the entire catalog and user embeddings are computed once per session. This transforms the inference problem from requiring evaluation of millions of user-item pairs for each recommendation request into a nearest-neighbor search over precomputed embeddings a reduction from $O(n^2)$ to $O(n \log n)$ complexity with respect to catalog size. Survey analyses of deep learning recommendation architectures reveal that embedding-based approaches reduce memory requirements by 40-70% compared to dense neural networks while maintaining comparable prediction accuracy [5].

The deployment advantages of this architecture manifest across multiple infrastructure dimensions. Memory footprint scales linearly with the sum of users and items rather than their product, enabling catalog expansion without prohibitive memory growth. Serialization overhead decreases dramatically because embedding vectors are compressed to dense float arrays, consuming only 4 bytes per dimension rather than requiring full computational graph serialization at 50-100 KB per layer. Distributed inference coordination simplifies to embarrassingly parallel embedding computation followed by a single aggregation step, eliminating complex inter-replica communication patterns. The resulting system achieves 50x inference throughput improvement, not through hardware acceleration but through architectural alignment with distributed serving constraints. Decision support systems utilizing data mining technologies have demonstrated that feature extraction and dimensionality reduction techniques can decrease processing time by 35-55% when integrated early in the model development pipeline [6].

Feature sparsity patterns represent another critical dimension of infrastructure-aware design. Models that depend on dense, high-cardinality categorical features introduce severe feature store bottlenecks at scale. Consider a model that requires user browsing history for the past 30 days as input naively implemented, this requires fetching and preprocessing potentially thousands of events per prediction request, generating 10-50 MB of data transfer per inference call. Recommendation infrastructures address this through feature materialization strategies that precompute aggregated feature representations asynchronously. Instead of fetching raw browsing events, the model consumes pre-aggregated statistics (category distribution, price ranges, temporal patterns) that compress browsing history into fixed-size vectors of 128-256 dimensions updated hourly. This trades model flexibility for operational tractability, accepting slightly stale features in exchange for eliminating real-time aggregation overhead. Research on decision support architectures indicates that preprocessing data through mining algorithms before deployment reduces query response times by 20-40% while maintaining data integrity across distributed systems [6].

The concept of "model sharding" emerges as a fundamental pattern for predictions exceeding single-machine memory. Video recommendation systems partition massive models across multiple serving machines, with each shard responsible for a subset of the output space.

4. Feature Infrastructure: The Hidden Bottleneck of Production ML

The feature store has emerged as the critical bottleneck in billion-scale ML systems, yet it remains overlooked in both academic research and industry discourse. While model architectures and training methodologies receive extensive attention, production ML systems spend more computational resources on feature computation and retrieval than on model inference itself. The architectural decisions around feature materialization, caching, and serving constrain which models can successfully deploy at scale. Research on provenance capture from distributed workflow systems has revealed that data lineage tracking in Apache Spark environments introduces an overhead of 8-15% in execution time, yet this metadata becomes essential for debugging feature computation pipelines and ensuring reproducibility across billion-record datasets [7].

Product recommendation infrastructures illustrate the magnitude of this challenge. A single recommendation request may require features derived from user purchase history (potentially thousands of transactions), browsing behavior (millions of page views across the entire user base), product catalog attributes (millions of items with hundreds of features each), real-time context (device type, time of day, current shopping cart), and computed aggregate statistics (category affinity scores, price sensitivity, seasonal preferences). Naively implemented, assembling these features would require dozens of database queries, potentially distributed across multiple microservices, with query latency dominating total prediction time at 200-500 milliseconds per request. Neural network-based learned index structures have demonstrated that replacing traditional B-tree indexes with learned models can reduce lookup times by 30-70% while decreasing memory footprint by 40-60%, offering promising approaches for accelerating feature retrieval operations [8].

The essential conflict in feature infrastructure architecture is between performance and freshness. Real-time functions - functions that are always evaluated using the latest data, yield optimum accuracy, but cause prohibitive latencies and require computation costs of about 50-150 milliseconds per computation of features. Precomputed features materialized asynchronously and served from caches achieve microsecond access latency but accept staleness. Production systems must navigate this tradeoff through tiered feature architectures that classify features by their freshness requirements and economic value. Workflow provenance systems analyzing data transformations in distributed

computing frameworks have shown that capturing intermediate computation states adds 12-18% storage overhead but enables rollback capabilities and fault tolerance mechanisms critical for maintaining feature consistency [7].
Video streaming recommendation systems implement sophisticated feature materialization pipelines that demonstrate this tiered approach. Slowly-changing features (user demographics, subscription tier, device capabilities) update daily and reside in distributed caches with millisecond access latency. Medium-velocity features (viewing history aggregates, preference trends) update hourly through streaming aggregation pipelines that consume event logs in real-time but batch writes to the feature store at intervals of 3600 seconds. High-velocity features (current browsing session context, immediate viewing patterns) compute on demand during the request path but rely only on data available in edge caches to avoid remote database queries. Studies on learned index architectures indicate that hybrid models combining neural networks with traditional indexing methods achieve query performance improvements of 2-3x for range queries while maintaining insertion speeds within 5-10% of baseline B-tree implementations [8].
The scale of feature materialization infrastructure often exceeds that of model serving itself. Music streaming systems maintain feature materialization pipelines processing 10 billion streaming events daily to keep user taste profiles current. This pipeline consumes more computational resources than the entire model training and serving infrastructure combined.

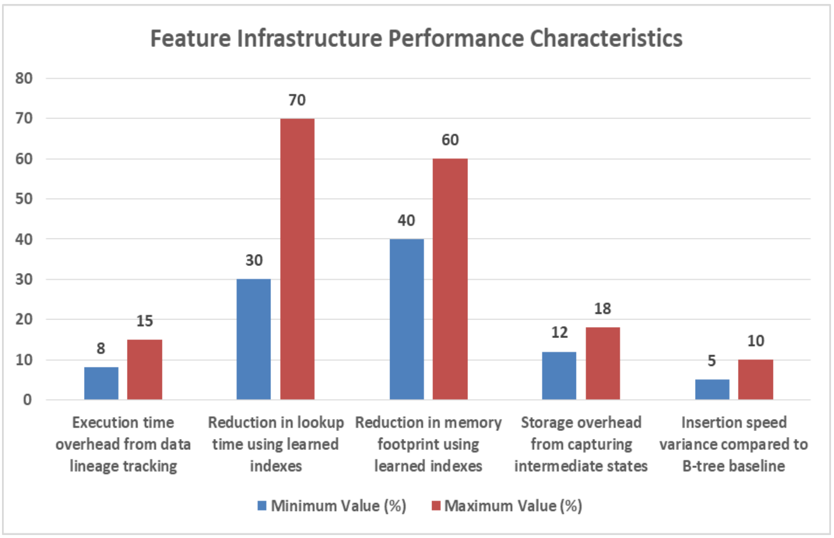

Figure 1: Feature Infrastructure Performance Characteristics [7, 8]

5. Serving Orchestration: Architectural Patterns for Low-Latency, High-Throughput Inference

The orchestration layer involving model inference, feature collection, and result aggregation is the most complicated element of production ML infrastructure, yet it is not widely documented beyond internal engineering systems. This layer must reconcile fundamentally contradictory requirements: models achieve maximum computational efficiency through large batch sizes, but production serving demands low-latency responses for individual requests. The architectural patterns that have emerged to navigate this tension illuminate fundamental tradeoffs in distributed systems design. Research on resource heterogeneity-aware scheduling for deep learning clusters has demonstrated that advanced scheduling algorithms considering GPU memory capacity, computational power variations, and network bandwidth heterogeneity can improve cluster utilization by 23-31% while reducing job completion times by 18-25% compared to traditional first-come-first-served approaches [9].

Dynamic batching has become the canonical approach to reconciling batch efficiency with latency requirements, but its implementation reveals subtle complexities. NVIDIA's Triton Inference Server, deployed widely across industry, implements adaptive batching where incoming requests accumulate in a buffer until either the batch reaches target size or a timeout expires. The critical insight is that optimal timeout values vary dynamically based on traffic patterns during high-traffic periods, batches fill rapidly and timeouts rarely trigger; during low-traffic periods, timeouts must be aggressive to maintain latency SLAs even as batch sizes shrink. Quantization methods for neural network inference have shown that reducing model precision from 32-bit floating point to 8-bit integer representations can accelerate inference by 2-4x while maintaining accuracy within 1-2% of full-precision baselines, enabling higher throughput during peak demand periods [10].

Ride-sharing platforms serving infrastructures show complex dynamic batching via predictive batch formation systems. Instead of fixed timeouts, the serving infrastructure manages a probabilistic model of request arrival rates and forecasts whether or not waiting to collect more requests will optimally increase the total throughput without breaking latency SLAs. In the morning commute periods with high request rates, the system pauses to fill batches of 64-128 requests, ensuring that it utilizes the GPUs well, 85- 95 percent of capacity. In low-traffic conditions during the night, the system will instantly transfer batches of between 4-8 requests instead of waiting for fuller batches, which might remain unfulfilled. This adaptive approach achieves 40% higher throughput during peak periods while maintaining p99 latency under 50 milliseconds across all traffic conditions. Heterogeneity-aware scheduling frameworks analyzing workload characteristics have revealed that placement strategies considering memory access patterns and computation-communication ratios reduce resource fragmentation by 15-20%, enabling more efficient multi-model deployment [9].

The architecture of model replicas introduces fundamental tradeoffs between isolation and efficiency. Independent replica deployment where each serving instance loads a complete model copy provides perfect isolation and trivial horizontal scaling, but multiplies the memory footprint linearly with replica count. A 10GB model deployed across 100 replicas consumes 1TB of memory, a prohibitive cost for large models requiring 32-64GB RAM per instance. Model sharing architectures where multiple serving processes share a single model loaded into shared memory reduce memory footprint by 60-80% but introduce complex failure modes where a crash in the shared model process cascades to all dependent serving instances. Quantization techniques applied at deployment time have demonstrated that mixed-precision serving strategies allocating 16-bit precision for activation-heavy layers and 8-bit precision for weight-dominant layers achieve 3.5x memory reduction with negligible accuracy degradation below 0.5% [10].

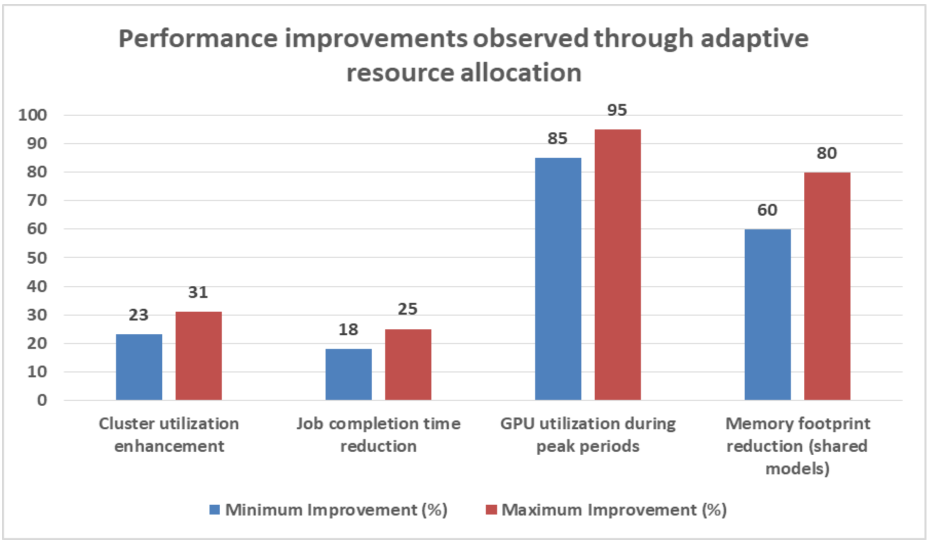

Figure 2: Performance improvements observed through adaptive resource allocation [9, 10]

Conclusion

The evidence presented throughout this article establishes that the artificial separation between model development and infrastructure engineering represents a bottleneck to advancing production machine learning systems at scale. The most sophisticated algorithms, optimized purely for offline accuracy metrics, fail catastrophically when confronted with the physical constraints of billion-scale serving infrastructure, while the most advanced infrastructure remains underutilized when constrained by models designed without deployment awareness. The infrastructure-model symbiosis documented through hyperscale platforms points toward a different approach: treating deployment characteristics as first-class design constraints rather than post-hoc optimization targets. This perspective fundamentally transforms every phase of the machine learning development lifecycle, requiring model architecture selection to evaluate not just predictive performance but memory footprint, serialization overhead, distributed inference coordination costs, and cacheability. Feature engineering must consider feature store bottlenecks, materialization pipeline complexity, and freshness-performance tradeoffs alongside predictive value. Training procedures must optimize for serving efficiency alongside accuracy, potentially accepting minor offline metric degradation in exchange for substantial inference speedup. The architectural patterns documented two-tower networks for distributed serving, hierarchical model cascades for cost-efficient inference, tiered feature materialization for freshness-performance balance, dynamic batching for latency-throughput optimization represent more than isolated engineering solutions. These patterns embody a fundamental reorientation of machine learning systems design around the recognition that infrastructure and models constitute a unified system rather than separate concerns. The economic framing reveals that infrastructure-aware design constitutes not merely an efficiency consideration but a strategic imperative, enabling sophisticated resource allocation strategies that deploy complex models for high-value decisions while using lightweight models for routine predictions. The future of production machine learning lies not in building ever-larger models that subsequently struggle to deploy, but in developing new model architectures and serving infrastructures in concert, unified by the recognition that neither can succeed without the other. This symbiotic relationship will only intensify as systems scale toward trillion-parameter models serving quadrillions of predictions annually, which suggests infrastructure-model co-design will become increasingly central to production ML success.

References

[1] Daniel Crankshaw et al., "Clipper: A Low-Latency Online Prediction Serving System", arXiv, 2017. [Online]. Available: https://arxiv.org/pdf/1612.03079
[2] Zao Zhang et al., "Coarse-to-Fine: A hierarchical DNN inference framework for edge computing", ScienceDirect, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X24000736
[3] Hao Chen et al., "BMS: Bandwidth-aware Multi-interface Scheduling for energy-efficient and delay-constrained gateway-to-device communications in IoT", ScienceDirect, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S1389128623000907
[4] Osama Almurshed et al., "Enhancing performance of machine learning tasks on edge-cloud infrastructures: A cross-domain Internet of Things-based framework", ScienceDirect, May 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X24006605
[5] Baichuan Liu et al., "A survey of recommendation systems based on deep learning", ResearchGate, 2021. [Online]. Available: https://www.researchgate.net/publication/349301342_A_survey_of_recommendation_systems_based_on_deep_learning
[6] Bahar Asgarova et al., "Development process of decision support systems using data mining technology", ResearchGate, 2024. [Online]. Available:

https://www.researchgate.net/publication/384511437_Development_process_of_decision_support_systems_using_data_mining_technology
[7] Thaylon Guedes et al., "Capturing and Analyzing Provenance from Spark-based Scientific Workflows with SAMbA-RaP", ScienceDirect, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0167739X1931742X
[8] Domenico Amato et al., "Neural networks as building blocks for the design of efficient learned indexes", Springer Nature, 2023. [Online]. Available: https://link.springer.com/article/10.1007/s00521-023-08841-1
[9] Abeda Sultana et al., "Resource Heterogeneity-Aware and Utilization-Enhanced Scheduling for Deep Learning Clusters", ResearchGate, Mar. 2025. [Online]. Available: https://www.researchgate.net/publication/389894683_Resource_Heterogeneity-Aware_and_Utilization-Enhanced_Scheduling_for_Deep_Learning_Clusters
[10] Amir Gholami et al., "A Survey of Quantization Methods for Efficient Neural Network Inference", arXiv, 2021.. [Online]. Available: https://arxiv.org/pdf/2103.13630