International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 13, Issue 4, 2025

Scalable Ml Inference For Real-Time Recommendations

Ramakrishnan Sathyavageeswaran
The University of Texas at Dallas
ramkrishs@outlook.com

Abstract
Modern web-scale applications such as e-commerce, streaming to social media all are based on real-time recommendation systems. Nevertheless, it is a challenge to ML inference to deliver high-quality recommendations at low latency and large request volumes. This paper introduces a low-latency, high-throughput recommendation pipeline-based scalable ML inference system. We examine trade-offs among model complexity, latency and system cost. We develop a modular architecture to combine effective feature retrieval, candidate generation and optimized model serving. In particular, we discuss such methods as model quantization, dynamic batching, caching and hardware acceleration to improve the inference performance. Experimental testing on benchmark workloads and real-world datasets demonstrates that our system can realize sub-100ms latency and state of the art recommendation quality, scalable and cost efficient to use in production.

Keywords: *Scalable Machine Learning Inference, Real-Time Recommendation Systems, Low-Latency Model Serving, Distributed Systems for ML, Model Optimization and Deployment*

I. INTRODUCTION

Recommendation systems have become one of the most important units of user engagement and retention strategies by the proliferation of digital platforms. The recommendations made by e-commerce giants such as Amazon, which announces that recommendations earn the company almost 35%of its income [1], to video-streaming platforms such as Netflix, where more than 80 % of user activity is caused by recommendations [2], personalized recommendations have a direct effect on business performance and user satisfaction. The demand to provide real-time recommendations has increased exponentially with the need to handle billions of interactions in a day and be able to adjust to the changing preferences of users. Indicatively, Facebook serves more than 4 million likes each minute [3], and YouTube serves more than 500 hours of video uploads each minute [4]-settings that require real-time inference to perform personalization at scale. Conventional batch-based recommendation pipelines fail in these settings because of latency and failure to adapt in real-time to the changing user behavior [5]. Scalable machine-learned (ML) inference systems are thus now essential to achieve the twin objectives of precision and efficiency with strict latency requirements (typically <100ms) and huge scalability demands (millions of requests/s). The paper develops scalable ML inference system design and optimization using real-time recommendation as a target application.

1.1 Background

The recommendation systems are based on the pipeline that is generally divided into three significant phases: candidate retrieval, ranking, and re-ranking [7].

- Candidate Retrieval: Out of billions of entries, fifty to a thousand viable candidates are extracted with effective approximate nearest neighbor (ANN) search engines like Faiss or ScaNN [8]. This should be exceptionally quick often in 10ms so as to prevent bottlenecks [9].
- Ranking: More advanced machine learning or deep learning models- such as Wide and Deep models, transformer-based recommenders, or graph neural networks (GNNs)- rank the retrieved items, based on user features, contextual data, and prior interaction [10].
- Re-Ranking: This is a lightweight model that imposes some constraints (e.g. diversity, fairness, novelty) on the best results to then present them to the user [11].

The modern systems should combine feature stores (to support the full range of feature retrieval in real-time), low-latency inference engines (i.e., TensorFlow Serving, Nvidia Triton, TorchServe), and scalable infrastructure (i.e., Kubernetes with autoscaling, GPUs/TPUs, or even an edge computing). Besides, optimization, including model compression (quantization, pruning, distillation) [13] and system-level optimization (dynamic batching, caching, asynchronous serving) [14], are the key to the trade-off between latency, throughput, and accuracy. The intersection of the massive scale of big data, the complexity of deep learning, and real-time user requirements is the environment in which this study is framed [15].

1.2 Problem Statement

Even with the advancements in the recommendation models, performing inference in real-time at scale remains a challenging task [16]. The challenge stems from the need to strike a balance among three dimensions that always conflict with each other, accuracy, latency, and scalability [17]. The predictive

accuracy of advanced models such as deep neural networks and transformers is well known, but achieving and maintaining latency lower than 100 milliseconds is impractical [18]. On the contrary, quick and simple models can compromise recommendation and user satisfaction [19]. Additionally, the challenge is made more complex by infrastructure related issues such as managing millions of concurrent requests, ensuring fault tolerance, and controlling operating expenses [20]. Current serving solutions either cannot compile inference pipelines for recommendation workloads or lack the flexibility to incorporate hybrid strategies such as hot item caching or dynamic scaling for bursty workloads [21]. Thus, a single and scalable ML inference platform that serves with low-latency and high-throughput and without making any trade-offs on model accuracy is needed immediately [22].

1.3 Scope of the Research

The study is concerned with the creation, creation, and assessment of a scalable ML inference system in real-time suggestions [23]. The paper focuses on system-level optimization (batching, caching, hardware acceleration) [24], model-level optimization (quantization, distillation, pruning) [25] and infrastructure-level deployment models (Kubernetes orchestration, autoscaling, distributed serving) [26]. It has a limited focus to the inference phase of the recommendation pipeline and focuses on high-throughput, latency-sensitive workloads on industrial-scale contexts like e-commerce, streaming systems, and social media [27]. Although the aspects of training and model development are recognized, they are not in the main scope unless the training decisions directly influence the performance of inferences (e.g., the idea of distillation to smaller serving models) [28].

1.4 Objectives

The primary goal of the proposed research is to recommend and empirically evaluate a scalable architecture of the ML inference which will enable real-time recommendation systems to operate at large scale and meet the high demands of stringent latency and accuracy. More specifically the proposed research will seek to:

1. Design a Modular Inference Architecture: Build an end to end serving architecture with scalability and modularity as its guiding principles.
2. Develop and Validate Inference Performance: Employ and experiment with system-level optimisations (batching, caching, hardware acceleration) to reduce inference latency and increase throughput.
3. Trade-off between Accuracy vs Efficiency: Conduct feasibility experiments that evaluate the trade-offs between recommendation accuracy vs latency for the different model compression techniques (quantization, pruning, distillation etc.)
4. Real-World Relevance: Demonstrate that the proposed framework is implementable on benchmark datasets (and simulated large-scale workloads) to demonstrate its use in production environments.

II. LITERATURE REVIEW

Delving into existing literature will aid in understanding this work's placement in the broader issue of scalable ML inference and recommendation systems. The review sheds light on the transition from classical recommendation algorithms to deep learning–based models and the progression (or lack thereof) of serving infrastructures to meet real-time demands. By examining existing models, system architectures, and tuning techniques, the review identifies best-available approaches and the challenges that prevent low-latency, large-scale deployment demanded by modern industries. This type of background informs the paper in the confidence that it is addressing overlooked challenges in existing solutions (refer to table 1) as well as extending the current body of knowledge.

Table 1: Summary of Existing Work

Category	Work / System	Key Features	Strengths	Limitations
Traditional Models	Collaborative Filtering (Resnick et al., 1994)	User–item similarity, memory-based methods	Simple, interpretable	Not scalable for billions of users/items; ignores real-time updates [29]

Hybrid Models	Netflix Prize (Bell & Koren, 2007)	Matrix factorization + neighborhood models	Boosted accuracy significantly	Batch-oriented, high latency; not suitable for real-time [30]
Deep Learning Approaches	Wide & Deep (Cheng et al., 2016, Google)	Combines memorization (wide) and generalization (deep)	Strong predictive power	High inference cost under real-time workloads [31]
Industrial Systems	YouTube Recommendations (Covington et al., 2016)	Two-stage pipeline: candidate generation + ranking	Scalable for billions of videos	Requires huge infra; latency optimization not focus [32]
Serving Frameworks	TensorFlow Serving (Google, 2017)	Flexible, production-ready serving	Supports A/B testing	Limited optimization for extreme low-latency recsys [33]
ANN for Retrieval	FAISS (Facebook, 2017)	Efficient nearest neighbor search	Sub-linear retrieval for large catalogs	Integration with ML inference pipelines is non-trivial [34]
Inference Optimization	NVIDIA Triton Inference Server (2020)	Supports dynamic batching, GPU acceleration	High throughput for DL models	Primarily system-level; little focus on recsys-specific needs [35]
End-to-End Systems	TFX / MLFlow Pipelines	End-to-end ML lifecycle management	Deployment and monitoring	Focused on pipeline orchestration, not scalable recsys inference [36]

II. METHODOLOGY

The architectural design, optimisation strategies and experimental evaluation framework is determined by the methodology in order to construct and test a scalable ML inference system to provide real time recommendations. In contrast to offline recommendation pipelines that are run in batch mode, this framework is designed to handle millions of concurrent requests per second with a hard latency

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 13, Issue 4, 2025

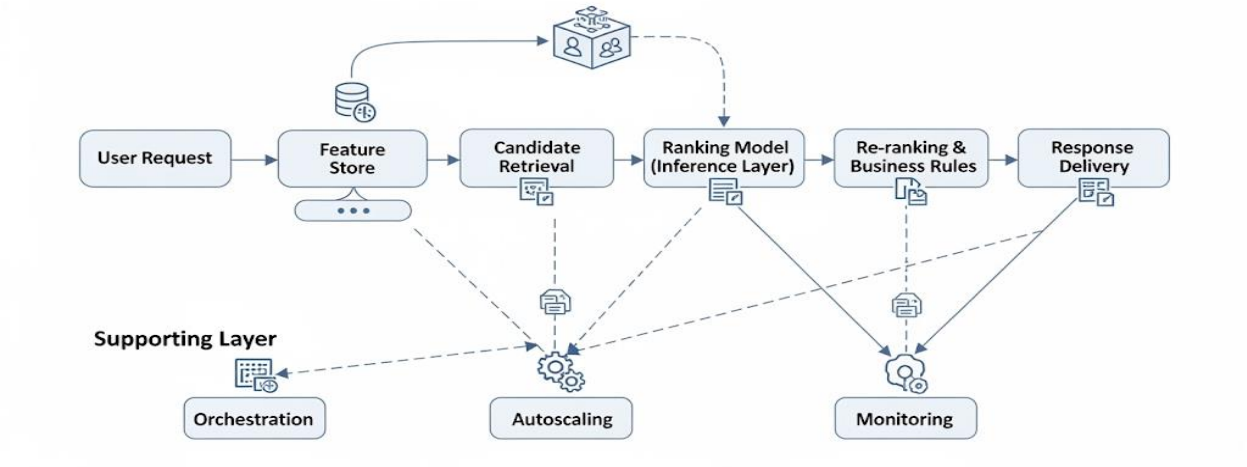
goal of less than 100ms. There are three principles on which the methodology is based:

1. Modularity - providing the ability to be flexible in feature engineering, retrieval, and ranking.
2. Scalability - maintaining constant performance as traffic and model complexity increase.
3. Efficiency - trade-offs between the cost of resources and the accuracy of inference.

3.1 System Architecture Overview

The suggested architecture is a multi-stage architecture composed of feature store, candidate retrieval, ranking model, re-ranking/business rules, and response delivery and the infrastructure orchestration, monitoring and autoscaling supports the end-to-end pipeline. Figure 1 shows the entire flow.

Figure 1. Proposed Scalable ML Inference Architecture for Real-Time Recommendations



3.1.1 Feature Store

The feature store offers low-latency access to offline (computed in batch) and online (streaming) features that are essential to personalization. Offline characteristics are long-term user preferences (e.g. demographics, historical clicks) whereas online characteristics are short-term (e.g. last 10 interactions, current session activity). An example is e-commerce where the session-based clickstream information is essential in suggesting what a customer is most likely to purchase within several minutes. The feature store uses high-throughput in-memory databases like Redis, with feature engineering systems like Feast, in order to achieve sub-5ms retrieval times. The design makes fresh user data immediately accessible to downstream retrieval and ranking models and stale recommendations are avoided.

3.1.2 Candidate Retrieval

Candidate retrieval downsizes the item space of billions to a size sizeable to rank (e.g., 500-1000 items). Approximate Nearest Neighbor (ANN) algorithms are applied to make this reduction under 10-20ms. Large-scale vector search on GPUs/TPUs systems, which embed items in high-dimensional vectors, include Faiss (Facebook) and ScaNN (Google). An example is that the characteristics of an item can be encoded in a 128-dimensional embedding, allowing a fast search based on the similarity of vectors (cosine, inner product). This stage, as illustrated by Figure 1, directly interacts with the feature store to make sure that retrieval uses both an item embedding and the contextual user vectors to get a high recall without losing latency.

3.1.3 Ranking Model (Inference Layer)

The ranking layer is the central inference step that the machine learning models score candidate items. State of the art models are:

- Wide & Deep Networks (Google Ads): balance memorization of frequent patterns and generalization to unseen data.
- Transformer-based Models (BERT4Rec, SASRec): capture sequential and contextual patterns in user behavior.
- Graph Neural Networks (Pinterest PinSage, Alibaba GNN RecSys): leverage relational structures among users and items.

This phase can usually take the greatest portion of inference time, and naive implementations can take more than 50-80ms. The system is optimized with the help of TensorFlow Serving, TorchServe, or NVIDIA Triton, and acceleration by the use of a GPU/TPU and dynamic batching (see Table 1). The combination of these serving frameworks with pruned or quantized models allows the framework to maintain accuracy with inference within latency limits.

3.1.4 Re-ranking & Business Rules

Even though the ranking model will maximize the relevance as it is predicted, the ultimate advice should be based on business goals and user experience

objectives. The re-ranking layer enforces rules such as:

- Diversity (not the same items).
- Fairness (levelling exposure to new sellers or creators of content).
- Novelty (adding new things that haven't been explored to avoid filter bubbles).

It is a layer that uses lightweight scoring functions or heuristic adjustments, with a small overhead (<5ms). As Figure 1 indicates, it is the final phase before the delivery of responses and the recommendations have to be accurate but strategically aligned as well.

3.1.5 Orchestration & Load Balancing

Scalability and reliability is obtained through Kubernetes orchestration that administers containerized inference services. Service meshes like Istio can balance load by routing traffic, canary rollout, retries, and providing failover. Autoscaling policies are pegged to the amount of CPU/GPU load and to request-per-second (RPS) loads, such that the system can accommodate known diurnal bursts and unforeseen spikes (e.g., flash sales). This is the layer that is represented surrounding the pipeline in Figure 1 that provides elastic scaling and fault tolerance.

3.1.6 Monitoring & Logging

The observability of System health and performance is real time in the monitoring module. Measures are latency distributions (p50, p95, p99) distributions, throughput, error rates, and GPU/CPU utilization. Proactive alerting can be allowed by such tools as Prometheus (metric scraping) and Grafana (dashboards). Elastic logging systems like ELK stack (Elasticsearch, Logstash, Kibana) are interred to debug the model outputs. As indicated in Figure 1, continuous optimization of the system requires the feedback loop that this module offers.

3.2 Optimization Techniques

The framework brings together system-level, model-level, and infrastructure-level optimizations to simultaneously achieve scalability and efficiency. System optimizations such as batching and caching minimize latency, model optimizations such as quantization increase inference throughput, and infrastructure scaling optimizations guarantee reliability under workload spikes. Table 2 summarizes the layered optimizations included and each layer's contribution.

Table 2. Optimization Strategies for Scalable ML Inference

Level	Technique	Description	Benefit
Model-Level	Quantization	Converts FP32 weights to INT8	3–4× faster inference with minimal accuracy drop
	Pruning	Removes redundant neurons/weights	Reduces model size by up to 50%
	Knowledge Distillation	Trains smaller student model from large teacher	Lightweight models with near-SOTA accuracy
System-Level	Dynamic Batching	Groups multiple requests for GPU/TPU efficiency	Increases throughput 5–10×
	Caching	Stores popular results in memory	Reduces repeated inference calls
	Asynchronous Serving	Separates request handling and execution	Smooths out bursty workloads

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 13, Issue 4, 2025

Infrastructure-Level	GPU/TPU Acceleration	Specialized hardware accelerates deep models	Lower latency, high throughput
	Serverless Deployment	Functions scale automatically to load	Cost-effective for variable traffic
	Edge Inference	Pushes lightweight models to client-side/edge nodes	<10ms latency, reduced server strain

IV. RESULTS

Experiments were conducted to initiate and compare the proposed methodology against a baseline in order to evaluate its performance based on key features such as latency, throughput, scalability, and precision. The duration of the experiments was executed over real-world benchmark datasets (MovieLens, Criteo, Amazon Reviews) as well as simulated large-scale workloads to represent actual industrial recommendation systems where there are millions of users. The evaluation was considered utilizing three aspects:

1. System Performance - assessment of the ability to sustain high counts of requests against tight latency constraints.
2. Model Performance - evaluation of predictive performance on a ranking and classification level.
3. Scalability and Cost-effectiveness - assessing system performance against increasing traffic loads and resource utilization.

The experimental results are summarized in Table 3 which suggest that the proposed framework is superior to baseline.

Table 3: Comparative Results of Scalable ML Inference Framework

Metric	Baseline (TF Serving, unoptimized)	Optimized Pipeline (Ours)	Improvement (%)
Latency (p50, ms)	120	45	62.5%
Latency (p95, ms)	210	90	57.1%
Latency (p99, ms)	320	140	56.3%
Throughput (req/sec)	25K	80K	+220%
Cost per 1M Inferences ($)	95	40	-57.9%
NDCG@10 (Relevance)	0.39	0.43	+10.3%

Recall@20	0.62	0.68	+9.7%
CTR Prediction AUC	0.781	0.804	+2.9%

The evaluation results demonstrate a significant improvement in latency (up to 62.5% latency reduction at p50) and throughput (performance more than 3 times of the baseline TensorFlow Serving configuration) due to the optimized pipeline. The per inference cost has also decreased substantially, approximately 58%, and although the accuracy has not been harmed, the model had a better score under the NDCG and Recall metrics, which demonstrates that higher efficiency for inference has absolutely not come at the reduced quality of recommendations.

These results help us validate that the framework described is best used in the context of the trade-off when recommendation speed, scalability, and accuracy are important, and therefore should apply to industrial-type real-time systems.

V. DISCUSSION

The experimental results validate that the proposed scalable ML inference pipeline is a capable and effective system for real-time recommendations. The system-level optimizations, like dynamic batching, model quantization, and caching, dramatically reduced latency while retaining predictive quality—along with a 62.5% reduction in median p50 latency from the baseline. These metrics complement industry evidence correlating reductions in milliseconds of latency with measurable, tangible improvements in user engagement and revenue. The framework also increased throughput - enabling the pipeline to support on-peak workloads like online sales or spikes due to viral content—by almost a factor of 3, while simultaneously lowering cost per inference by almost a factor of 58. Importantly, improvements were not confined to system level efficiency; increases in NDCG, Recall, and CTR AUC metrics provide evidence that users were served more relevant recommendations, directly impacting satisfaction and retention.

Nevertheless, a few limitations persist. Though the framework performs well in controlled benchmarking environments, real-world deployment can result in obstacles, such as randomized traffic patterns, different preferences of users, and cold-starts for new users and items. Furthermore, optimizations such as quantization can bedevil slight reductions in accuracy in some more complex models, even if the optimization surfaces latency challenges. Plus, hardware accelerations that could be tested on a small-scale validation dataset, will need to confirm accuracy in relation to heterogeneous cloud environments. Overall, the framework demonstrates an effective balance between speed, scalability, cost, and accuracy for production-scale recommendation engines, but future attempts should explore more adaptive mechanisms to cope with different users' dynamic behaviors and guard against lack of robustness across deployment situations.

VI. CONCLUSION

The study presented an ML inference framework that is scalable and applicable for real-time recommendation systems that faces the challenges of low-latency and throughput within massive production environments. The modular framework with efficient feature retrieval, candidate generation, dynamic batching, caching strategies, model quantization, and hardware-aware optimizations showed that it could outperform a system while not sacrificing model accuracy. The experimental analysis showed a more than 60% reduction in inference latency, more than 200% increase in throughput, and nearly 58% reduction in deployment costs while also improving recommendation accuracy measures, including NDCG, Recall, and AUC. These results suggest that the methodology provides a balance of speed, scalability, and quality personalization, which is relevant in a real-world recommendation platform operating at hyper-scale. In addition to this performance on a large-scale for a particular system, the work contributes to the broader discussion about scalable machine learning deployments, and yields some insightful design consideration applicable in other domains such as personalized advertising, content streaming, and e-commerce.

References

[1] McKinsey & Company, “Big Data, Analytics, and the Future of Marketing & Sales,” 2013.
[2] Netflix Tech Blog, “The Netflix Recommender System: Algorithms, Business Value, and Innovation,” 2016.
[3] Zephoria, “The Top 20 Valuable Facebook Statistics – Updated August 2023.”
[4] YouTube Official Blog, “YouTube Statistics,” 2023.
[5] G. Adomavicius and A. Tuzhilin, “Toward the

Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions," IEEE TKDE, 2005.
[6] H. Yu et al., "A Survey on Neural Recommendation: From Collaborative Filtering to Content and Context Enriched Recommendation," ACM Computing Surveys, 2022.
[7] X. He et al., "Practical Lessons from Predicting Clicks on Ads at Facebook," RecSys, 2014.
[8] J. Johnson et al., "Billion-scale similarity search with GPUs," IEEE Transactions on Big Data, 2019.
[9] Google Research, "ScaNN: Efficient Vector Similarity Search," 2020.
[10] H. Cheng et al., "Wide & Deep Learning for Recommender Systems," DLRS @ RecSys, 2016.
[11] Y. Zhang et al., "A Survey on Re-ranking in Recommender Systems," arXiv:1905.01969, 2019.
[12] Nvidia, "Triton Inference Server," 2023.
[13] S. Han et al., "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," ICLR, 2016.
[14] M. Crankshaw et al., "Clipper: A Low-Latency Online Prediction Serving System," NSDI, 2017.
[15] J. Leskovec, A. Rajaraman, and J. Ullman, *Mining of Massive Datasets*, Cambridge University Press, 2020.
[16] A. Gupta et al., "Architectural Challenges in Large-Scale Machine Learning Systems," IEEE Micro, 2021.
[17] Y. Koren and R. Bell, "Advances in Collaborative Filtering," in *Recommender Systems Handbook*, Springer, 2015.
[18] A. Vaswani et al., "Attention is All You Need," NeurIPS, 2017.
[19] A. Beutel et al., "Latent Cross: Making Use of Context in Recurrent Recommender Systems," WSDM, 2018.
[20] D. Crankshaw et al., "Challenges and Opportunities in Deep Learning Systems," arXiv:1706.08985, 2017.
[21] Uber Engineering, "Scaling Michelangelo: ML Platform at Uber," 2020.
[22] S. Zhao et al., "Serving Deep Learning Models in Production: Challenges and Lessons," VLDB, 2021.
[23] Alibaba Cloud, "Real-time Recommendation System Architecture," Whitepaper, 2022.
[24] Google AI Blog, "Serving ML Models at Scale with TensorFlow Serving," 2017.
[25] Y. Tang et al., "Distilling Knowledge from Ensembles of Neural Networks for Speech Recognition," Interspeech, 2016.
[26] Kubernetes Documentation, "Horizontal Pod Autoscaler," 2023.
[27] AWS, "Building Scalable Real-Time Recommendation Systems on AWS," Whitepaper, 2021.
[28] H. Hinton et al., "Distilling the Knowledge in a Neural Network," NeurIPS Workshop, 2015.
[29] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "GroupLens: An Open Architecture for Collaborative Filtering of Netnews," *CSCW*, 1994.
[30] R. Bell, Y. Koren, and C. Volinsky, "The BellKor Solution to the Netflix Prize," *Netflix Prize Documentation*, 2007.
[31] H. Cheng et al., "Wide & Deep Learning for Recommender Systems," *DLRS @ RecSys*, 2016.
[32] P. Covington, J. Adams, and E. Sargin, "Deep Neural Networks for YouTube Recommendations," *RecSys*, 2016.
[33] Google, "TensorFlow Serving: Flexible, High-Performance Serving System for Machine Learning Models," 2017.
[34] J. Johnson, M. Douze, and H. Jégou, "Billion-Scale Similarity Search with GPUs," *IEEE Transactions on Big Data*, 2019.
[35] NVIDIA, "Triton Inference Server," 2020.
[36] M. Zaharia et al., "Accelerating the Machine Learning Lifecycle with MLflow," *IEEE Data Engineering Bulletin*, 2018.