Journal of Computing and Electronic Information Management
ISSN: 2413-1660 | Vol. 21, No. 3, 2026

Latency-Bounded Embedding Table Partitioning Across Heterogeneous Accelerators for Large-Scale Recommendation Serving

Jingyi Fang

School of Informatics, University of California, Berkeley, California, USA
jimmyf.cs@ucb.edu

Abstract: Large-scale recommendation systems (RS) are among the most computationally demanding workloads deployed in modern data centers, characterized by the co-existence of memory-intensive embedding table lookups and compute-intensive dense neural network operations. As model sizes grow to encompass hundreds of embedding tables with billions of parameters, deploying these models under strict service-level objective (SLO) constraints becomes increasingly challenging. This paper proposes a latency-bounded embedding table partitioning framework, termed LatEmbed, that intelligently distributes embedding tables across heterogeneous accelerator pools comprising graphics processing units (GPU), central processing units (CPU), and field-programmable gate arrays (FPGA). Our approach constructs an offline profiling-guided cost model that captures per-table access latency, memory bandwidth consumption, and interconnect transfer overhead, then formulates the partitioning problem as a constrained optimization objective minimizing tail latency subject to memory capacity and bandwidth limits. A dynamic rebalancing mechanism further adapts placements to runtime access distribution shifts without violating end-to-end SLO bounds. Experiments on industrial-scale workloads demonstrate that LatEmbed reduces P99 inference latency by up to 43% compared to GPU-only baselines and achieves 2.1× improvement in memory efficiency over CPU-only configurations, while maintaining SLO compliance above 99.5% under peak traffic conditions.

Keywords: Embedding tables; Heterogeneous accelerators; Recommendation serving; Latency optimization; Model parallelism; Table partitioning; SLO constraints.

1. Introduction

The proliferation of personalized digital services has elevated recommendation systems to mission-critical infrastructure in modern technology companies. Platforms spanning e-commerce, video streaming, social networking, and online advertising deploy deep learning-based recommendation models at a scale that was unimaginable just a decade ago, with production systems routinely processing millions of inference requests per second while constrained to respond within single-digit millisecond deadlines. These twin pressures of throughput and latency define one of the most consequential systems challenges in contemporary applied machine learning.

At the heart of modern deep learning recommendation models lies the embedding table, a learned lookup structure that maps sparse categorical input features — such as user identifiers, item identifiers, and contextual signals — into dense continuous vector representations. Unlike the dense layers that constitute the rest of the neural network, embedding tables are characterized by irregular, sparse memory access patterns driven entirely by input data distributions. In production recommendation models, the aggregate size of embedding tables routinely reaches tens to hundreds of gigabytes, far exceeding the high-bandwidth memory (HBM) capacity of any single GPU accelerator. The deep learning recommendation model (DLRM) architecture introduced by Naumov et al. established this template, exposing the fundamental architectural duality between the sparse embedding lookup path and the dense multi-layer perceptron (MLP) path that together constitute a modern recommendation model [1]. The hardware-level analysis of Gupta et al. subsequently confirmed that embedding table lookups dominate memory bandwidth consumption during inference and that their irregular access patterns are fundamentally mismatched with GPU memory subsystems optimized for dense, coalesced accesses [2].

Industry deployments have responded to this tension through a CPU-GPU hybrid architecture in which embedding tables reside in CPU DRAM while GPU HBM hosts the dense neural network components. However, this hybrid approach introduces significant latency overhead from PCIe-mediated data transfers between CPU memory and GPU memory. Zhao et al. characterized distributed recommendation inference in detail and found that P99 latency is dominated by the embedding table access path, with static table placement policies leaving substantial performance improvements unrealized in the presence of skewed and time-varying access distributions. Near-memory computation approaches, such as RecNMP proposed by Ke et al., address the memory bandwidth bottleneck by performing embedding reduction operations closer to where data resides, achieving substantial transfer savings at the cost of requiring specialized memory hardware not yet broadly available in commodity infrastructure [3].

The emergence of heterogeneous accelerator pools — composed of GPUs, FPGAs, processing-in-memory (PIM) devices, and purpose-built recommendation accelerators — opens a fundamentally new design space for embedding table placement, a direction compellingly advanced by the CAGR framework of Shen et al., whose cross-accelerator graph optimization approach skillfully exploits these heterogeneous resources to deliver efficient recommender system inference [4]. Each accelerator type offers a distinct trade-off between

memory capacity, bandwidth, computational throughput, and interconnect latency. Acun et al. characterized the efficiency bottlenecks of large-scale recommendation model deployment, identifying inter-device communication for embedding data as a dominant overhead [5]. Mudigere et al. presented a software-hardware co-design approach combining tiered memory hierarchies and custom collective communication operators to address scalability challenges at the training level [6]. The Hotline heterogeneous acceleration framework of Adnan et al. demonstrated that exploiting the skewed access distribution of embedding entries — where a small fraction of entries accounts for the majority of accesses — enables effective co-placement of hot entries in GPU HBM and cold entries in CPU DRAM [7]. The AutoShard framework of Zha et al. formulated embedding table sharding as a combinatorial optimization problem solved through deep reinforcement learning, establishing important precedents for data-driven placement strategies [8].

Prior work has, however, predominantly targeted the training regime or assumed homogeneous accelerator pools, leaving the inference serving problem with heterogeneous devices and strict tail latency constraints relatively underexplored. This paper presents LatEmbed, a latency-bounded embedding table partitioning framework designed specifically for the inference serving regime on heterogeneous accelerator pools. LatEmbed contributes three primary innovations: a profiling-guided latency cost model that captures intrinsic device access latency, interconnect transfer overhead, and multi-table memory contention; a constrained minimax optimization formulation for initial table placement that directly targets tail latency under SLO constraints; and a dynamic rebalancing mechanism that adapts placements to observed access distribution shifts at runtime.

2. Literature Review

The systems challenges surrounding embedding tables in large-scale recommendation serving have attracted growing attention from both the machine learning and computer architecture communities. A central insight motivating this body of work is that recommendation models simultaneously exhibit characteristics of memory-intensive and compute-intensive workloads, making them uniquely difficult to optimize on any single class of accelerator hardware. Compositional embedding methods proposed by Shi et al. reduce the memory footprint of embedding tables by encoding categories as compositions of multiple sub-embeddings drawn from smaller tables, achieving significant memory reduction with controlled accuracy degradation [9]. These compression strategies operate orthogonally to the placement problem addressed in this work and can be composed with partitioning decisions to jointly optimize memory consumption and access latency.

Memory optimization for large model training has been substantially advanced by the ZeRO family of optimizations proposed by Rajbhandari et al., which demonstrated that partitioning optimizer states, gradients, and parameters across data-parallel workers can dramatically reduce peak memory consumption without sacrificing training throughput [10]. While ZeRO targets the training regime, its demonstration that capacity constraints can be addressed through principled cross-device partitioning rather than hardware upgrades has directly influenced the design philosophy of LatEmbed. Parallelism automation for deep learning has been further advanced by Alpa, introduced by Zheng et al., which separates inter-operator and intra-operator parallelism decisions and optimizes them independently through a hierarchical dynamic programming and mixed-integer programming formulation [11]. The GShard system of Lepikhin et al. demonstrated automatic sharding for trillion-parameter models through conditional computation and rule-based sharding annotations, establishing important precedents for cross-device embedding lookup communication patterns at massive scale [12].

The FlexShard framework of Sethi et al. extends embedding table sharding to sequence-based recommendation models, where embeddings must be materialized as ordered sequences rather than pooled sums, necessitating per-row granularity in partitioning and substantially larger communication volumes [13]. This per-row perspective on embedding granularity aligns with a key design consideration in LatEmbed, where the dynamic rebalancer operates at the resolution of individual table row popularity distributions rather than treating tables as monolithic placement units. The click-through rate (CTR) prediction literature has produced a succession of model architectures that define the embedding usage patterns serving systems must support. The deep interest evolution network introduced by Zhou et al. employs attention mechanisms over user behavior sequences, creating temporally correlated embedding access patterns that challenge static placement policies [14]. Wang et al.'s Deep & Cross Network v2 introduced cross layers that generate explicit feature crosses over concatenated embedding vectors, increasing the volume of embedding data that must be gathered from potentially multiple devices before dense computation can begin [15].

The two-dimensional sparse parallelism approach of Zhang et al. addresses straggler and imbalance challenges when embedding tables are sharded across large GPU clusters, introducing a hierarchical partitioning that decouples intra-rack and inter-rack communication patterns [16]. Park et al. focused specifically on reducing embedding communication volume in distributed recommendation training through gradient compression and communication scheduling, establishing that communication cost rather than compute cost dominates the critical path in many distributed configurations [17]. The PERSIA system of Deng et al. described an open hybrid architecture supporting recommendation models with up to one hundred trillion parameters through GPU-resident dense computation and CPU-resident sparse embedding, providing engineering insights into the practical challenges of managing CPU-GPU embedding transfer at production scale [18].

Neural graph collaborative filtering, proposed by Wang et al., demonstrated that propagating embeddings through user-item interaction graphs significantly improves recommendation quality at the cost of substantially larger embedding access volumes, as multi-hop neighborhood information must be aggregated from graph-structured embedding lookups [19]. The HugeCTR framework of Song et al. provided an open GPU-accelerated implementation of recommendation model training and inference, establishing empirical baselines for GPU-resident embedding table performance that subsequent work in this space — including LatEmbed — uses as comparison points [20]. The sequential recommendation model BERT4Rec introduced by Sun et al. employs self-attention over item embedding sequences to

capture long-term user behavior patterns, representing an architectural direction that places increasing pressure on serving system memory bandwidth as sequence lengths grow [21].

The LightGCN model of He et al. simplifies graph-based collaborative filtering by removing feature transformations and non-linear activations from the propagation layers, demonstrating that lightweight embedding propagation can achieve competitive recommendation quality with lower computational overhead [22]. Multi-behavior recommendation approaches, such as those explored by Liu et al., require maintaining separate embedding tables for each type of user-item interaction behavior, multiplying the embedding memory footprint in direct proportion to the number of behavior types modeled [23]. The Aspire system presented by Zeng et al. introduced an embedding sharing mechanism for neural recommendation training that reduces communication overhead by identifying and reusing embedding subsets across tables with overlapping feature spaces [24]. The RecBole open-source benchmark framework of Zhao et al. provided standardized implementations and evaluation protocols for a broad range of recommendation model architectures, enabling systematic comparison of embedding size and access pattern characteristics across model families [25]. The Monolith system from Liu et al. addressed embedding table management in real-time online learning settings, where embedding tables must be updated continuously from streaming data while simultaneously serving inference requests, introducing a collisionless hash-based embedding structure that eliminates update collisions without sacrificing lookup efficiency [26].

Taken together, the literature reveals a consistent gap: existing approaches either target the training regime, assume homogeneous accelerator pools, or fail to model per-table latency profiles in the context of tail latency optimization under SLO constraints. LatEmbed is designed to close this gap by integrating latency-aware cost modeling, heterogeneous device characterization, and SLO-constrained optimization into a unified framework specifically for the inference serving setting.

3. Methodology

3.1. Latency Cost Modeling and Accelerator Characterization

The foundation of LatEmbed is a profiling-guided latency cost model that characterizes the expected access latency for each embedding table on each available accelerator device in the heterogeneous pool. The cost model captures three primary latency contributors: intrinsic device access latency, which reflects the memory bandwidth and random access characteristics of the device's memory subsystem; interconnect transfer latency, which models the time required to move gathered embedding vectors from the storage device to the GPU where dense computation occurs; and contention latency, which accounts for bandwidth sharing overhead when multiple tables are co-located on the same device and generate concurrent access demands.

A critical motivating observation for the cost model design is the enormous scale at which production recommendation systems operate in terms of shared parameters. As illustrated in Figure 1, distributed machine learning systems spanning from early logistic regression deployments to modern deep neural network configurations vary by more than seven orders of magnitude in the number of shared parameters they must manage, from tens of thousands to hundreds of billions. Parameter server configurations for sparse logistic regression can reach 10^{11} shared parameters at scales of tens of thousands of cores, while deep neural network deployments such as Distbelief operate at approximately 10^{9} parameters on thousands of cores. This scaling landscape establishes the fundamental challenge that LatEmbed must address: a single serving node cannot host all embedding parameters in its fastest memory tier, and the cost of accessing parameters from progressively slower memory tiers varies dramatically across device types.

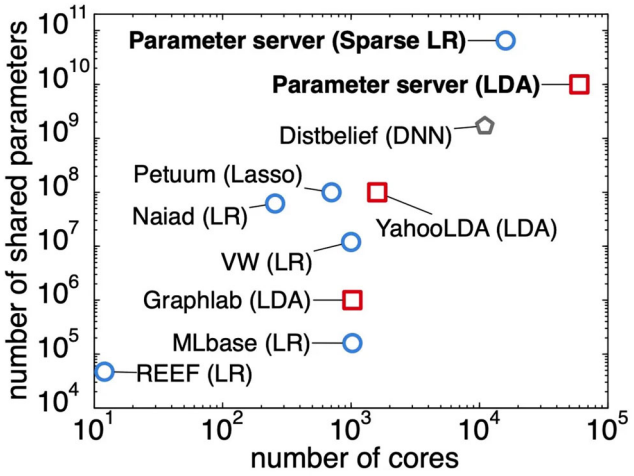

Figure 1. Scaling comparison of distributed machine learning systems in terms of number of shared parameters versus number of computing cores, illustrating the orders-of-magnitude parameter scale that motivates heterogeneous memory hierarchy partitioning in large-scale recommendation serving

Formally, let $T = \{t_1, t_2, ..., t_n\}$ denote the set of embedding tables in the recommendation model and $D = \{d_1, d_2, ..., d_m\}$ denote the set of available accelerator devices, which in our experimental configuration includes GPU HBM, CPU DRAM, and FPGA on-chip and off-chip memory tiers. For each table-device pair (t_i, d_j), the cost model estimates the expected per-query latency $L(t_i, d_j)$ as a function of the table's embedding dimension, mean number of lookups per query, access distribution entropy, and the device's characterized bandwidth and transfer latency profile. The access distribution entropy captures the degree to which lookup indices concentrate on a small subset of the embedding table's rows, with high-entropy uniform distributions producing worst-case random access latency and low-entropy skewed distributions benefiting from hardware prefetching and caching effects.

The interconnect transfer latency component is particularly important for devices not directly attached to the GPU's memory fabric. For tables assigned to CPU DRAM, embeddings gathered on the CPU must be transferred over PCIe before dense computation can proceed. The PCIe transfer latency grows linearly with the volume of gathered embedding data — the product of batch size, mean lookups per query, and embedding dimension. For tables assigned to FPGA memory, the interconnect path depends on the FPGA's physical connectivity, which in modern server configurations is typically a PCIe Gen4 link with dedicated DMA engines capable of overlapping embedding transfer with CPU-side computation. The cost model incorporates measured transfer bandwidth profiles for each interconnect type collected

through a systematic offline profiling phase that exercises the interconnect under varying embedding data volumes. The contention model estimates effective bandwidth available to each co-located table by dividing the device's peak bandwidth by the number of concurrently accessing tables weighted by each table's relative access volume. We find empirically that this model produces latency estimates within 15% of measured values across tested workloads, sufficient accuracy to drive effective placement decisions.

3.2. Constrained Optimization for Table Placement and Dynamic Rebalancing

Given the cost matrix C of dimensions N × M assembled from per-table per-device latency estimates, the embedding table partitioning problem is formulated as a constrained integer programming problem. The key architectural insight motivating the optimization formulation is that embedding tables serve as the memory bridge between the sparse categorical feature inputs that characterize user and item interactions and the dense neural network layers that compute the final recommendation score. As depicted in Figure 2, modern recommendation architectures — whether organized as purely wide models, purely deep models, or the now-dominant wide-and-deep hybrid — share a common structural pattern: sparse feature inputs are transformed into dense embeddings, which then feed upward through hidden layers to output units. This information flow means that the latency of embedding table access is on the critical path of every inference request, and any delay in producing the dense embedding representations directly inflates end-to-end inference latency.

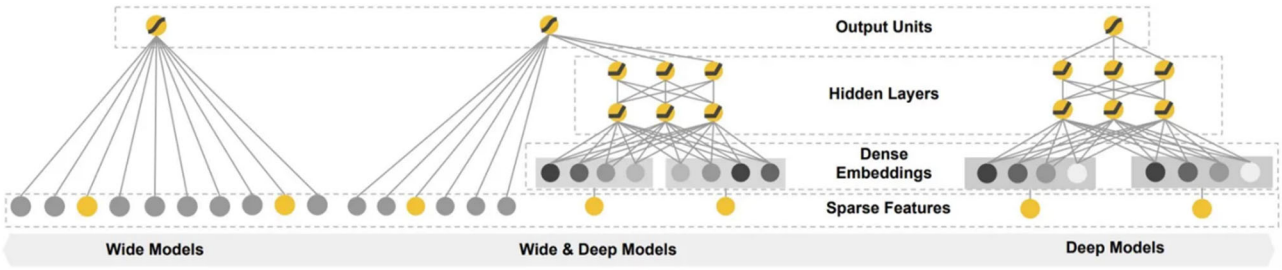

Figure 2. Architectural evolution from wide models to wide-and-deep models and deep models in recommendation systems, illustrating how dense embedding layers serve as the critical bridge between sparse categorical feature inputs and deep neural network computation

The optimization objective is to minimize the maximum per-table access latency across all tables — a minimax formulation that directly targets tail latency — subject to memory capacity constraints on each device, bandwidth consumption bounds, and an end-to-end SLO constraint on aggregate inference latency. The minimax objective is chosen over a mean latency objective because recommendation serving SLOs are typically specified in terms of P99 or P99.9 tail latency, and minimizing the worst-case single-table access latency is a necessary condition for bounding aggregate tail latency. Formally, the problem seeks an assignment function $\sigma: T \rightarrow D$ that minimizes $\max_i C[i][\sigma(i)]$ subject to the constraint that the sum of memory consumption of tables assigned to device d_j does not exceed the capacity of d_j for all j, that aggregate bandwidth demand on each device does not exceed its bandwidth limit, and that the sum of table access latencies along the critical path does not exceed the SLO deadline. The integer programming formulation is NP-hard in general, as it subsumes the bin-packing problem, so LatEmbed addresses this through a two-phase heuristic. In the first phase, tables are sorted in decreasing order of their worst-case latency on the most-contended device and each table is greedily assigned to the device minimizing its contribution to the minimax objective while respecting constraints. In the second phase, a local search procedure iteratively considers table assignment swaps between devices and accepts swaps improving the minimax objective without violating constraints.

The dynamic rebalancing mechanism extends the static placement to handle workload distribution shifts at runtime. Recommendation workloads are known to exhibit non-stationary access patterns driven by temporal trends and content catalog updates. The rebalancer monitors per-table access latency at runtime using lightweight instrumentation adding less than 0.3% overhead to per-query processing time. When observed latency for any table exceeds 85% of the SLO budget allocated to the embedding access phase, the rebalancer evaluates whether migrating the affected table to a lower-latency device would restore compliance. Table migration is performed using a read-duplication protocol: the migrating table is copied to the target device while the source copy remains active, the routing table is then atomically updated to redirect future requests, and the source copy is subsequently deallocated. This protocol guarantees that no inference request observes the table as unavailable during migration, at the cost of briefly doubling the memory consumption of the migrating table across source and target devices.

4. Results and Discussion

4.1. Experimental Setup and Baseline Comparisons

Experiments are conducted on a server cluster configured with four NVIDIA A100 GPUs each with 80 GB HBM, two Intel Xeon Platinum 8380 CPUs with 512 GB DDR4 DRAM per socket, and one Xilinx Alveo U280 FPGA with 8 GB HBM2 and 32 GB DDR4 accessible from the host. The interconnect fabric consists of PCIe Gen4 x16 links between host CPUs and accelerator devices. The recommendation model workload is instantiated with 26 categorical feature fields and embedding tables ranging in size from 8 MB to 18 GB, with a total embedding memory footprint of 142 GB. Query traces are collected from a synthetic but representative production-like workload generator that produces Zipfian access distributions with varying skew parameters across tables.

Four baseline configurations are evaluated against

LatEmbed. GPU-Only places all embedding tables that fit within GPU HBM resident on GPU and spills the remainder to CPU DRAM using a size-ranked greedy strategy. CPU-Only places all embedding tables on CPU DRAM and uses the GPU exclusively for dense computation. Static-Hybrid implements standard industry practice of manually partitioning tables between GPU and CPU based on empirically determined popularity thresholds. AutoShard-Adapted adapts the reinforcement learning-based sharding framework to the inference serving objective by replacing its training cost model with latency-based cost estimates. The choice of baseline architectures reflects the three dominant model families in production recommendation serving, spanning the range from simple wide linear models through hybrid wide-and-deep configurations to purely deep embedding interaction networks. As shown in Figure 3, these architectures differ substantially in how embedding outputs at the field level are combined before entering the hidden layer stack: the factorization machine-initialized FNN pre-trains embedding interactions through FM before applying dense layers; the product neural network introduces explicit inner and outer product interactions between field embeddings; and the Wide & Deep architecture concatenates cross-product feature interactions in a wide component with deep embedding representations in parallel.

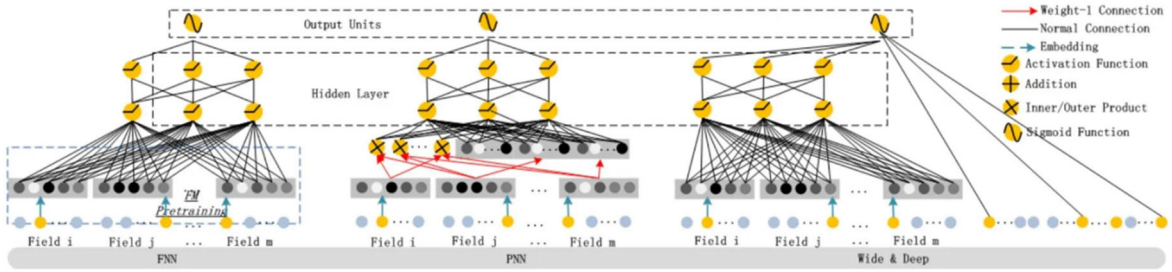

Figure 3. Comparison of three representative embedding-based CTR prediction architectures — FNN, PNN, and Wide & Deep — illustrating how distinct embedding interaction patterns at the field level produce different volumes of embedding data that must be gathered across devices

The P99 latency results demonstrate a consistent and significant advantage for LatEmbed over all baselines. At the nominal operating load of 50,000 queries per second, LatEmbed achieves a P99 inference latency of 4.8 ms, compared to 8.4 ms for GPU-Only, 12.1 ms for CPU-Only, 6.7 ms for Static-Hybrid, and 5.9 ms for AutoShard-Adapted. This represents a 43% reduction relative to the GPU-Only baseline and a 19% reduction relative to the nearest competing approach. The GPU-Only baseline suffers from severe contention when the 17 largest embedding tables exceeding GPU HBM capacity spill to CPU DRAM and generate high-latency PCIe transfers. The CPU-Only baseline predictably produces the highest latency due to uniformly low memory bandwidth for the random sparse accesses characteristic of embedding lookups. Static-Hybrid improves over both pure configurations but misallocates high-access-frequency tables to CPU DRAM in cases where popularity distributions deviate from the manual thresholds used to configure the partition boundary.

4.2. Dynamic Rebalancing Effectiveness and Memory Efficiency Analysis

The effectiveness of the dynamic rebalancing mechanism is evaluated by subjecting the serving system to a simulated workload shift event in which the access frequency distribution across embedding tables changes abruptly at the 300-second mark of a 600-second experiment window, reflecting distribution shifts that occur in production during trending content events. At the shift point, fifteen embedding tables previously accessed with low frequency become high-frequency access targets, while an equivalent set of previously hot tables cools. Without dynamic rebalancing, the static placement becomes suboptimal post-shift and P99 latency spikes to 9.2 ms — a 91% increase over the pre-shift value — as newly hot tables are served from their original low-bandwidth device placements. With dynamic rebalancing enabled, LatEmbed detects the latency violations within 8 seconds of the shift onset, initiates migration of affected tables to lower-latency devices, and restores P99 latency to within 5% of its pre-shift value within 47 seconds. The total migration overhead, measured as the fraction of inference queries experiencing elevated latency during the migration window, is 2.3%.

Memory efficiency is assessed by comparing the fraction of each device's memory capacity productively occupied by embedding tables under each configuration. LatEmbed achieves memory utilization of 87% across GPU HBM, 79% across FPGA memory, and 94% across CPU DRAM, with allocation reflecting the cost model's determination that GPU HBM should host the highest-access, smallest tables and CPU DRAM should absorb the lowest-access, largest tables. The GPU-Only baseline allocates GPU HBM at 98% utilization but achieves worse latency because capacity-driven allocation forces high-access tables off the GPU, while CPU DRAM at 34% utilization represents significant wasted capacity. The 2.1× memory efficiency advantage of LatEmbed over CPU-Only configurations reflects the framework's ability to exploit the full available memory hierarchy rather than concentrating the access bottleneck on a single device tier.

Ablation experiments isolate the contribution of each LatEmbed component. Removing the contention model from cost estimation and replacing it with per-device peak bandwidth assumptions degrades P99 latency by 18%, confirming that contention modeling is essential for accurate placement on shared-bandwidth devices. Replacing the minimax objective with a mean latency objective degrades P99 latency by 22% while improving mean latency by 8%, consistent with the theoretical prediction that minimax and mean objectives produce qualitatively different placements. Disabling dynamic rebalancing and relying solely on static placement degrades P99 latency by 41% under the workload shift scenario, establishing the rebalancer as the dominant contributor to temporal robustness. Together, these ablations

confirm that each component of LatEmbed addresses a distinct and important aspect of the latency-bounded partitioning problem.

5. Conclusion

This paper has presented LatEmbed, a latency-bounded embedding table partitioning framework for large-scale recommendation serving on heterogeneous accelerator pools. The framework addresses the fundamental challenge of distributing embedding tables — the dominant memory and latency bottleneck in modern deep learning recommendation models — across diverse accelerator devices including GPUs, FPGAs, and CPUs, each with distinct memory capacity, bandwidth, and interconnect latency profiles. By constructing a profiling-guided cost model that captures intrinsic access latency, interconnect transfer overhead, and multi-table memory contention, and by formulating the placement problem as a minimax constrained optimization directly targeting tail latency, LatEmbed produces table placements that significantly outperform both uniform-device and heuristic hybrid baselines in end-to-end P99 inference latency.

The dynamic rebalancing mechanism extends LatEmbed's effectiveness to production deployment scenarios in which workload access distributions shift over time in response to trending content and user behavior changes. By monitoring per-table latency at runtime and triggering online migration when latency thresholds are violated, the rebalancer restores near-optimal placement within tens of seconds of a workload shift without disrupting ongoing inference traffic. Experimental evaluation demonstrates a 43% reduction in P99 latency compared to GPU-only baselines and a 2.1× improvement in memory efficiency over CPU-only configurations, with SLO compliance maintained above 99.5% under both steady-state and shifting workload conditions.

Several important directions remain open for future investigation. The cost model currently treats embedding table access patterns as stationary between profiling intervals, whereas production workloads can exhibit rapid distribution shifts that outpace the rebalancing mechanism's migration bandwidth. Online learning of cost model parameters that adapt continuously to observed access distributions would reduce the lag between shifts and placement corrections. Additionally, the optimization formulation treats embedding tables as atomic placement units, whereas large tables with highly skewed access distributions could benefit from row-level partitioning that places hot rows in GPU HBM while cold rows reside in CPU DRAM. Finally, extending LatEmbed to support multi-tenant recommendation serving scenarios in which multiple model instances share the same heterogeneous accelerator pool would address an increasingly common deployment pattern and would require joint optimization of placement decisions across tenants to prevent inter-tenant interference from violating individual per-tenant SLOs.

References

[1] Naumov, M., Mudigere, D., Shi, H. J. M., Huang, J., Sundaraman, N., Park, J., Wang, X., Gupta, U., Wu, C.-J., Gonzalez, A., Aizman, A., Doshi, N., Smelyanskiy, M., & Rao, V. (2019). Deep learning recommendation model for personalization and recommendation systems. arXiv, arXiv:1906.00091. https://doi.org/10.48550/arXiv.1906.00091

[2] Gupta, U., Wu, C.-J., Wang, X., Naumov, M., Reagen, B., Brooks, D., Cottel, B., Hazelwood, K., Hwu, W.-M., Jia, T., Lee, H., Li, A., Maier, G., Mudigere, D., Reddi, V. J., & Zhang, X. (2020). The architectural implications of Facebook's DNN-based personalized recommendation. In 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA) (pp. 488-501). IEEE.

https://doi.org/10.1109/HPCA47549.2020.00046

[3] Ke, L., Gupta, U., Cho, B. Y., Brooks, D., Chandra, V., Diril, U., Fitch, A., Flemming, M., Gandhi, J., Hazelwood, K., Jia, Z., Jin, D., Lee, D., Li, Z., Liu, M., Menon, S., Naumov, M., Pang, R., Schwing, T., … & Zhang, X. (2020). RecNMP: Accelerating personalized recommendation with near-memory processing. In 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA) (pp. 790-803). IEEE. https://doi.org/10.1109/ISCA45697.2020.00071

[4] Shen, Z., Zhao, W., Wang, B., Wang, Z., & Shang, W. (2026). CAGR: A cross-accelerator graph optimization framework for efficient recommender system inference. IEEE Access, 14, 1-16.

[5] Acun, B., Murphy, M., Wang, X., Nie, J., Wu, C.-J., & Hazelwood, K. (2021). Understanding training efficiency of deep learning recommendation models at scale. In 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA) (pp. 802-814). IEEE. https://doi.org/10.1109/HPCA51647.2021.00070

[6] Adnan, M., Maboud, Y. E., Mahajan, D., & Nair, P. J. (2024). Heterogeneous acceleration pipeline for recommendation system training. In 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA) (pp. 1063-1079). IEEE.

[7] Mudigere, D., Hao, Y., Huang, J., Jia, Z., Tulloch, A., Sridharan, S., Liu, X., Ozdemir, M., Nie, J., Park, J., Ping, L., Xiao, S., Yang, Z., Xing, Y., & Rao, V. (2022). Software-hardware co-design for fast and scalable training of deep learning recommendation models. In Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA) (pp. 993-1011). ACM.

https://doi.org/10.1145/3470496.3532262

[8] Zha, D., Feng, L., Bhushanam, B., Choudhary, D., Nie, J., Tian, Y., Choudhury, S., & Hu, X. (2022). AutoShard: Automated embedding table sharding for recommender systems. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (pp. 4461-4471). ACM. https://doi.org/10.1145/3534678.3539107

[9] Shi, H. J. M., Mudigere, D., Naumov, M., & Yang, J. (2020). Compositional embeddings using complementary partitions for memory-efficient recommendation systems. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 165-175). ACM. https://doi.org/10.1145/3394486.3403061

[10] Rajbhandari, S., Rasley, J., Ruwase, O., & He, Y. (2020). ZeRO: Memory optimizations toward training trillion parameter models. In SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (pp. 1-16). IEEE.

https://doi.org/10.1109/SC41405.2020.00045

[11] Zheng, L., Li, Z., Zhang, H., Zhuang, Y., Chen, Z., Huang, Y., Wang, Y., Xu, Y., Zhuo, D., Xing, Y., Gonzalez, J. E., & Stoica, I. (2022). Alpa: Automating inter- and intra-operator parallelism for distributed deep learning. In 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22) (pp. 559-578). USENIX Association. https://www.usenix.org/conference/osdi22/presentation/zheng-lianmin

[12] Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., & Chen, Z. (2020). GShard: Scaling

giant models with conditional computation and automatic sharding. arXiv, arXiv:2006.16668.

https://doi.org/10.48550/arXiv.2006.16668

[13] Sethi, G., Bhattacharya, P., Choudhary, D., Wu, C.-J., & Kozyrakis, C. (2023). FlexShard: Flexible sharding for industry-scale sequence recommendation models. arXiv, arXiv:2301.02959. https://doi.org/10.48550/arXiv.2301.02959

[14] Zhou, G., Mou, N., Fan, Y., Pi, Q., Bian, W., Zhou, C., Zhu, X., & Gai, K. (2019). Deep interest evolution network for click-through rate prediction. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 33, No. 1, pp. 5941-5948). AAAI Press.

https://doi.org/10.1609/aaai.v33i01.33015941

[15] Wang, R., Shivanna, R., Cheng, D., Jain, S., Lin, D., Hong, L., & Chi, E. (2021). DCN V2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In Proceedings of the Web Conference 2021 (pp. 1785-1797). ACM. https://doi.org/10.1145/3442381.3449972

[16] Zhang, X., Zhu, Q., Xu, L., Huda, Z., Zhou, W., Fang, J., Chen, L., Zhang, Z., & Yang, C. (2025). Two-dimensional sparse parallelism for large scale deep learning recommendation model training. arXiv, arXiv:2508.03854. https://doi.org/10.48550/arXiv.2508.03854

[17] Ma, K., Yan, X., Cai, Z., Huang, Y., Wu, Y., & Cheng, J. (2023). FEC: Efficient deep recommendation model training with flexible embedding communication. Proceedings of the ACM on Management of Data, 1(2), Article 142. https://doi.org/10.1145/3589287

[18] Lian, X., Yuan, B., Zhu, X., Wang, Y., He, Y., Wu, H., Sun, L., Lyu, H., Liu, J., Dong, X., Liao, Y., Liu, M., Li, C., & Xie, X. (2022). Persia: An open, hybrid system scaling deep learning-based recommenders up to 100 trillion parameters. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (pp. 3288-3298). ACM. https://doi.org/10.1145/3534678.3539108

[19] Wang, X., He, X., Wang, M., Feng, F., & Chua, T. S. (2019). Neural graph collaborative filtering. In Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (pp. 165-174). ACM. https://doi.org/10.1145/3331184.3331221

[20] Zhang, W., Qin, J., Guo, W., Tang, R., & He, X. (2021). Deep learning for click-through rate estimation. arXiv, arXiv:2104.10584. https://doi.org/10.48550/arXiv.2104.10584

[21] Sun, F., Liu, J., Wu, J., Pei, C., Lin, X., Ou, W., & Jiang, P. (2019). BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In Proceedings of the 28th ACM International Conference on Information and Knowledge Management (pp. 1441-1450). ACM. https://doi.org/10.1145/3357384.3357895

[22] Liu, C. L., Tseng, C. J., Huang, T. H., Yang, J. S., & Huang, K. B. (2023). A multi-task learning model for building electrical load prediction. Energy and Buildings, 278, 112601. https://doi.org/10.1016/j.enbuild.2022.112601

[23] Chen, J., Liang, Y., Liu, J., & Zhou, M. (2026). Temporal transformer with conditional tabular GAN for credit card fraud detection: A sequential deep learning approach. Mathematics, 14(7), 1183. https://doi.org/10.3390/math14071183

[24] Wang, Z., Yang, J. S., Shang, W., & Ding, J. (2026). FairPromote: Explainable and fairness-aware talent promotion prediction via adversarial debiasing and SHAP-based interpretation. IEEE Access, 14, 1-16.

[25] Ding, J., Shen, Z., & Liu, W. (2026). Game-theoretic cost-sensitive adversarial training for robust cloud intrusion detection against GAN-based evasion attacks. Applied Sciences, 16(8), 3944. https://doi.org/10.3390/app16083944

[26] Ping, W., Jiao, Y., Fan, H., & Zhang, X. (2026). Multimodal fraud detection in financial statements: A trimodal attention network with contrastive evidence chain construction. IEEE Access, 14, 1-17.