

# A SYSTEMS-AWARE FRAMEWORK FOR INTELLIGENT LARGE-SCALE RECOMMENDATION MODELS

AoWei Shen<sup>1\*</sup>, Shan Huang<sup>2</sup>

<sup>1</sup>University of Washington, Seattle, WA 98195, USA.

<sup>2</sup>University of California San Diego, La Jolla, CA 92093, USA.

\*Corresponding Author: AoWei Shen

**Abstract:** Embedding tables dominate the memory footprint of deep learning recommendation models, and their size already exceeds the capacity of the fast memory attached to accelerators. Existing compression methods reduce parameter counts without regard to where the parameters reside, while placement and caching systems move fixed tables between memory levels without changing their representation. This paper presents TierRec, a systems-aware framework that treats representation and placement as one decision. TierRec profiles the access skew of each categorical field together with an empirical fidelity profile of candidate representations, partitions the ID space into a full-dimensional hot tier, low-dimensional warm tiers with shared projections, and a compositional cold tier, and chooses the tier boundaries and dimensions by maximizing an access-weighted fidelity objective under a memory budget and, optionally, a slow-memory traffic constraint derived from a two-level memory model. Placement then follows access density so that compact rows occupy fast memory first. On MovieLens-20M with a DLRM-style ranking model, TierRec reaches a test AUC of 0.8199 at 8× compression and 0.8153 at 16× compression, matching or exceeding mixed-dimension embeddings and outperforming hashing, quotient-remainder and frequency-based double hashing by 0.0176 to 0.0560 AUC. The hierarchy-aware variant meets a slow-traffic bound of half the memory-only optimum at a cost of 0.0008 AUC. Trace-driven analyses on real and Criteo-scale synthetic workloads quantify the resulting gains in fast-memory hit ratio and access time.

**Keywords:** Recommender systems; Embedding compression; Memory hierarchy; Deep learning recommendation model; Systems-aware machine learning

## 1 INTRODUCTION

Personalized ranking in advertising, feeds and e-commerce relies on models that combine a large number of categorical inputs with neural feature interactions. Since factorization machines introduced low-rank embeddings for sparse features, industrial architectures such as Wide & Deep, DeepFM, Deep & Cross Network, Deep Interest Network, neural factorization machines and DLRM have preserved the same basic structure: every categorical value is mapped to a dense vector stored in an embedding table, and the vectors are fed to multilayer perceptrons or explicit interaction operators [1-7].

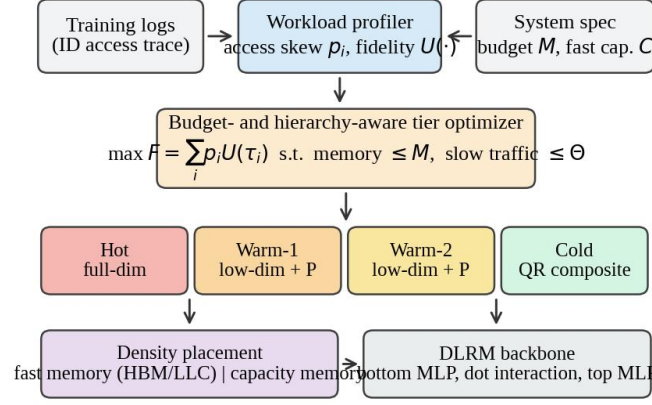
The embedding tables are the dominant cost of these models. Characterizations of production workloads report that embeddings occupy the bulk of model memory, that the lookups are bandwidth bound, and that tables grow faster than the capacity of accelerator memory [8-9]. Training systems therefore distribute tables over GPU high-bandwidth memory, host DRAM and even solid-state storage [10-11], and systems such as FAE and RecShard exploit the strong popularity skew of IDs to keep frequently accessed rows in the fastest tier [12-13]. A second line of work reduces the tables themselves through feature hashing, compositional quotient-remainder embeddings, mixed-dimension embeddings, frequency-based double hashing, tensor-train decomposition, hash-based deep encoders and learned embedding sizes [14-21].

These two lines of work make separate decisions about the same bytes. Compression methods decide how many parameters each ID receives and judge the result by accuracy and parameter count, while placement methods decide where the parameters of an already fixed representation live. The skew that justifies giving popular IDs more capacity is exactly the property that determines which bytes are touched by each lookup, and a compact row is cheaper to keep in fast memory than a full row. A representation chosen without regard to placement can therefore waste fast memory, and a placement computed for full-dimensional rows cannot exploit the fact that most IDs do not need them.

This paper presents TierRec, a systems-aware framework that couples the two decisions (Figure 1). A workload profiler measures the access distribution of every categorical field and an empirical fidelity profile that scores candidate representations on validation data. A tiered hybrid embedding layer gives full-dimensional vectors to the hottest IDs, low-dimensional vectors with shared projections to one or two warm tiers, and compositional quotient-remainder vectors to the cold tail. A tier optimizer selects the boundaries and dimensions that maximize an access-weighted fidelity objective under the memory budget of each table and, when a fast-memory capacity is specified, under a bound on the expected slow-memory traffic. The resulting rows are placed by access density, so that fast memory is filled with the bytes that serve the largest share of lookups.

The contributions are fourfold. First, we formulate embedding design as a joint allocation problem over representation fidelity, memory capacity and memory-hierarchy traffic, and we show that the allocation depends on the fidelity profile

only up to an affine transformation. Second, we propose a tiered hybrid embedding with a grid-search optimizer that runs in well under a second per table and supports an explicit slow-traffic constraint. Third, we show that the fidelity profile must be measured on the full training data: a profile estimated on a 10% sample systematically undervalues tail dimensions and costs 0.0016 AUC. Fourth, we evaluate TierRec on MovieLens-20M and MovieLens-Tag against five compression baselines, and with trace-driven hierarchy simulations on real and Criteo-scale synthetic workloads together with measured single-core throughput.



**Figure 1** Overview of TierRec. Profiled Access Statistics and System Limits Drive a Tier Optimizer; the Resulting Hot, Warm and Cold Tiers Are Placed Across the Memory Hierarchy by Access Density and Feed a DLRM-Style Backbone

## 2 RELATED WORK

### 2.1 Deep Recommendation Models

Factorization machines model pairwise feature interactions with inner products of embeddings [1]. Wide & Deep combines a memorizing linear component with a deep network, DeepFM and neural factorization machines merge factorization terms with MLPs, DCN adds explicit cross layers, and DIN attends over user behavior sequences [2-6]. DLRM standardizes the architecture used in large-scale systems research: dense features pass through a bottom MLP, sparse features are embedded, pairwise dot products form the interaction layer, and a top MLP produces the click probability [7]. We adopt this backbone throughout the paper because its cost profile, dominated by embedding lookups, is representative of production models [8-9].

### 2.2 Embedding Compression

Feature hashing maps IDs into a smaller table and accepts collisions [14]. The quotient-remainder trick composes two small tables through complementary partitions so that every ID receives a unique vector [15]. Mixed-dimension embeddings (MD) assign dimensions that decrease with popularity following a power rule and restore a common width with projections [16]. Frequency-based double hashing (FDH) keeps unique rows for frequent IDs and represents the remaining IDs with two hashed rows [17]. TT-Rec factorizes tables into tensor-train cores, DHE replaces tables with a deep network over hash codes, and PEP and AutoDim learn per-feature or per-field dimensions [18-22]. These methods target accuracy per parameter; none of them models the memory level from which the parameters are read.

### 2.3 Systems for Large Embedding Tables

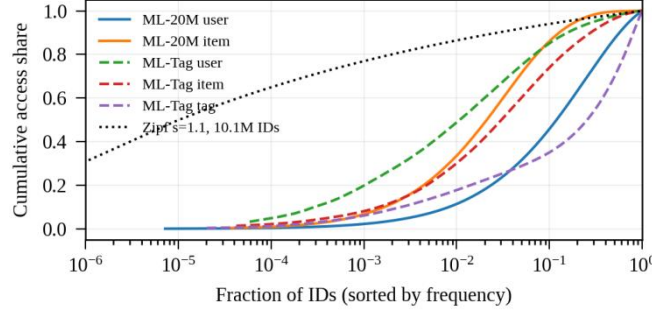
Gupta et al. and Acun et al. show that recommendation inference and training are bound by embedding capacity and bandwidth [8-9]. Mudigere et al. co-design sharding, pipelining and row-wise sparse optimizers for multi-GPU training, and the hierarchical parameter server of Zhao et al. spans GPU memory, host memory and SSD [10-11]. FAE identifies hot embeddings from input statistics and keeps them on the GPU, and RecShard partitions tables across a tiered memory using per-row access statistics [12-13]. Caching theory explains why these approaches work: web and embedding workloads follow Zipf-like laws [23], and the hit ratio of an LRU cache under independent references is accurately predicted by the characteristic-time approximation of Che et al. [24]. TierRec builds on these observations but also changes the representation, so that the placement problem is solved over rows whose sizes depend on their popularity.

## 3 PROBLEM FORMULATION AND WORKLOAD CHARACTERIZATION

A model has  $F$  categorical fields. Field  $f$  has  $n_f$  IDs, a base embedding dimension  $d$ , and an empirical access distribution  $p_{f,i}$  estimated from the training log. Ranking the IDs by frequency yields  $p_{f,(1)} \geq p_{f,(2)} \geq \dots$ , and we write  $P(a, b)$  for the access share of the ranks in  $[a, b]$ . For a compression ratio  $\rho$ , the memory budget of the field is  $M_f = n_f d / \rho$  floats, and every parameter that serves the field, including projection matrices, counts against it. The execution platform exposes a

fast memory of capacity  $C$  (GPU HBM or a last-level cache) and a capacity memory reached over a slower link, with effective bandwidths  $\beta_{\text{fast}}$  and  $\beta_{\text{slow}}$ .

Figure 2 and Table 1 summarize the access skew of the two real datasets. On MovieLens-20M the top 1% of movies receive 33.5% of all item lookups and the top 10% receive 86.1%; users are less skewed, with 11.4% and 45.8%. The tag field of MovieLens-Tag is the most concentrated. A Zipf law with exponent 1.1 over 10.1 M IDs, which we use for the synthetic Criteo-scale study, lies between these curves. Two consequences follow. A small set of rows serves most lookups and deserves full capacity and fast memory, and the long tail contains many rows that are rarely read and receive few gradient updates, so their capacity can be reduced sharply with little effect on accuracy.



**Figure 2** Cumulative Access Share Versus the Fraction of IDs Sorted by Training Frequency

**Table 1** Access Skew of the Categorical Fields (Training Split)

| Field       | IDs     | Top-1% share | Top-10% share |
|-------------|---------|--------------|---------------|
| ML-20M user | 138,493 | 11.4%        | 45.8%         |
| ML-20M item | 26,744  | 33.5%        | 86.1%         |
| ML-Tag user | 17,045  | 48.8%        | 85.1%         |
| ML-Tag item | 23,743  | 30.0%        | 74.0%         |
| ML-Tag tag  | 49,657  | 17.7%        | 35.0%         |

## 4 THE TIERREC FRAMEWORK

### 4.1 Workload Profiler

The profiler computes the frequency rank of every ID from the training log; IDs absent from the log are ranked last. It also measures a fidelity profile that scores each candidate representation type. For a uniform per-ID dimension  $k$  we train the backbone with every large field embedded at dimension  $k$  and record the validation AUC  $A(k)$ ;  $A(0)$  denotes the model without these embeddings and  $A(d)$  the full model. The fidelity of dimension  $k$  is

$$U(k) = [A(k) - A(0)] / [A(d) - A(0)], \quad (1)$$

and the fidelity  $U_C$  of the compositional cold representation is obtained in the same way from a model in which every ID uses a minimal quotient-remainder embedding. On MovieLens-20M the profile is  $U(1) = 0.891$ ,  $U(2) = 0.933$ ,  $U(4) = 0.967$ ,  $U(8) = 0.98$ ,  $U(16) = 0.991$  and  $U_C = 0.609$ . The profile is computed once per dataset with one pass over the training data per point, and it is reused for every budget.

### 4.2 Tiered Hybrid Embedding

Let  $r_i$  be the frequency rank of ID  $i$ . TierRec partitions the ranks into a hot tier  $[0, h)$ , up to two warm tiers of sizes  $w_1$  and  $w_2$  with dimensions  $d_1 > d_2$ , and a cold tier holding the remaining  $n_C$  IDs. With tier offsets  $o_l$ , the embedding of ID  $i$  is

$$e_i = H[r_i], \quad r_i < h, \quad (2)$$

$$e_i = W_l[r_i - o_l] P_l, \quad i \in \text{warm tier } l, \quad (3)$$

$$e_i = Q[j \bmod m] \odot R[\lfloor j/m \rfloor], \quad j = r_i - o_C, \quad (4)$$

where  $H$  stores full  $d$ -dimensional rows,  $W_l$  stores  $d_l$ -dimensional rows,  $P_l \in \mathbb{R}^{d_l \times d}$  is a projection shared by the tier, and  $Q$  and  $R$  are the remainder and quotient tables of the cold tier with  $m = \lceil \sqrt{n_C} \rceil$ . Every ID therefore keeps a distinct vector: hot and warm IDs own their rows, and cold IDs receive unique compositions [15]. The cold tier also absorbs IDs that never appear in training. Because all tiers emit  $d$ -dimensional vectors, the layer is a drop-in replacement for an embedding table in DLRM.

### 4.3 Budget-Constrained Tier Optimization

Given the ranked access distribution, the expected fidelity of a configuration is the access-weighted average of the fidelities of its tiers,

$$F = P_H + \sum_l P_{W_l} U(d_l) + P_C U_C, \quad (5)$$

where  $P_H$ ,  $P_W$  and  $P_C$  are the access shares of the hot, warm and cold ranks. The memory of the configuration must satisfy

$$h d + \sum_i (w_i d_i + d_i d) + (m + \lceil n_C/m \rceil) d \leq M_f. \quad (6)$$

Since the access shares of the tiers sum to one, adding a constant to  $U$  or multiplying it by a positive factor changes  $F$  by the same affine map for every configuration, so the maximizer of (5) is invariant to positive affine transformations of the profile. Any monotone normalization of the profiling metric therefore yields the same allocation, and only the relative gaps between representation types matter. The optimizer enumerates  $d_1 > d_2$  from  $\{1, 2, 4, 8, 16\}$ , a geometric grid of hot sizes  $h$  and first-warm sizes  $w_1$ , and lets the last warm tier absorb the remaining budget, which is optimal for fixed dimensions because  $U(d_2)$  exceeds  $U_C$ . With prefix sums of the ranked distribution, each candidate is evaluated in constant time, and a field with 138 K IDs is optimized in well under a second.

**Algorithm 1.** Tier allocation of TierRec for one field

Input: ranked access shares  $p_{(1)} \geq \dots \geq p_{(n)}$ , profile  $U$ , dimension  $d$ , budget  $M$ , grids  $G_h$ ,  $G_w$ , dimension set  $\mathcal{D}$ .

- 1: compute the prefix sums  $P(a) = \sum_{r \leq a} p(r)$ ;
- 2: for each pair  $d_1 > d_2$  in  $\mathcal{D}$ , each  $h \in G_h$  and  $w_1 \in G_w$  do
- 3:      $w_2 \leftarrow$  largest size for which the memory in (6) stays within  $M$ ;
- 4:      $n_C \leftarrow n - h - w_1 - w_2$ ;    $m \leftarrow \lceil \sqrt{n_C} \rceil$ ;
- 5:     evaluate  $F$  by (5) with the prefix sums;
- 6:     optionally evaluate  $T_{\text{slow}}$  and discard the candidate if  $T_{\text{slow}} > \theta \cdot T_{\text{slow}}^0$ ;
- 7: end for
- 8: return the feasible configuration with the largest  $F$ .

#### 4.4 Hierarchy-Aware Placement and Traffic Constraint

TierRec treats every row of every table as an object  $o$  with access probability  $\pi_o$  per lookup and size  $s_o$  bytes. Projection matrices are dense parameters and stay in fast memory. The remaining capacity is filled in decreasing order of the access density  $\pi_o/s_o$ , which solves the fractional relaxation of maximizing the fast-served access mass under capacity  $C$ . Low-dimensional warm rows have a four- to thirty-two-fold smaller size than hot rows and therefore compete for fast memory at a proportionally lower access probability. The expected traffic to the capacity memory and the modeled access time per sample are

$$T_{\text{slow}} = \sum_{o \in \mathcal{F}} \pi_o s_o, \quad t = B_{\text{fast}}/\beta_{\text{fast}} + B_{\text{slow}}/\beta_{\text{slow}}, \quad (7)$$

where  $\mathcal{F}$  is the set of objects placed in fast memory and  $B_{\text{fast}}$ ,  $B_{\text{slow}}$  are the bytes per sample read from each level. The systems-aware variant, TierRec-SA, maximizes (5) subject to (6) and  $T_{\text{slow}} \leq \theta \cdot T_{\text{slow}}^0$ , where  $T_{\text{slow}}^0$  is the traffic of the memory-only optimum. For fixed warm dimensions, the optimizer searches the hot and warm sizes and evaluates  $T_{\text{slow}}$  for each candidate by bisection on the density threshold over the sorted tiers.

#### 4.5 Training

All sparse tables, including the warm and cold tables, are updated with row-wise Adagrad applied only to the rows touched by a mini-batch, as in production DLRM training [10, 25]. Dense parameters, namely the MLPs and the projections  $P_i$ , are trained with Adam [26]. The second quotient-remainder table is initialized around one so that a cold composition starts close to its remainder vector.

#### 4.6 Complexity

Per lookup, a hot ID costs one gather of  $d$  floats, a warm ID one gather of  $d_i$  floats followed by  $2d_i d$  floating-point operations for the projection, and a cold ID two gathers of  $d$  floats and  $d$  multiplications. With  $d = 32$  and  $d_i \leq 16$  the projection adds at most 1,024 operations per lookup, a small fraction of the top MLP of the backbone. The optimizer evaluates every dimension pair of  $\mathcal{D}$  on the grids  $G_h \times G_w$ ; with prefix sums each evaluation takes constant time, and the systems-aware search adds a bisection over the density threshold with logarithmic searches in the sorted tiers. The profile requires one training pass per representation type and is shared by all budgets and capacities.

### 5 EXPERIMENTAL SETUP

**Datasets.** MovieLens-20M contains 20,000,263 ratings from 138,493 users on 26,744 movies [22]. A rating of at least four is a positive label (49.98% of samples). The sparse fields are the user, the movie, the movie rated immediately before by the same user, the hour of day and the day of week; the dense features are the logarithms of the numbers of earlier ratings of the user and of the movie, computed causally. The data are split randomly into 80% training, 10% validation and 10% test samples. MovieLens-Tag contains 2,006,859 user-item-tag instances over 17,045 users, 23,743 items and 49,657 tags, with the public 70/20/10 split [6]. For the Criteo-scale study we generate a synthetic workload with the 26 categorical cardinalities of the Criteo Kaggle dataset, from 3 to 10,131,227 IDs and 33.76 M rows in total, and Zipf access with exponent 1.1 over randomly permuted IDs.

**Model and training.** The backbone follows DLRM with base dimension  $d = 32$ , a bottom MLP of 64 and 32 units for dense features, pairwise dot-product interactions concatenated with the embeddings, and a top MLP of 128 and 64 units [7]. Mini-batches contain 4,096 samples. MovieLens-20M models are trained for one epoch; MovieLens-Tag models

are trained for up to five epochs with early stopping on validation AUC. The sparse learning rate of 0.03 was selected on the MovieLens-20M validation split from  $\{0.01, 0.03, 0.1\}$  and reused for MovieLens-Tag; Adam uses a learning rate of  $10^{-3}$ . Tables with at least 1,000 IDs are compressed at  $8\times$  and  $16\times$  relative to a 32-dimensional table; the time features stay uncompressed for every method.

**Baselines.** Uniform-LD reduces the dimension of all IDs and restores 32 dimensions with a projection. Hash maps IDs to a smaller table with a universal hash [14]. QR uses complementary partitions with the largest remainder table that fits the budget [15]. MD uses four frequency blocks of geometrically growing size with dimensions set by the power rule and the exponent solved to meet the budget [16]. FDH keeps unique rows for the most frequent IDs within half of the budget and sums two hashed rows for the others [17]. All methods share the backbone, optimizer and budgets.

**Metrics and platform.** We report test AUC and LogLoss, embedding memory, the fetch-level hit ratio of the fast memory, the bytes per sample read from the capacity memory, and the access time of (7) with  $\beta_{\text{fast}} = 900$  GB/s and  $\beta_{\text{slow}} = 16$  GB/s, representative of HBM2 device memory and a PCIe 3.0  $\times 16$  host link. Hit ratios on MovieLens-20M are measured by replaying the 2 M test interactions in time order against placements computed from training counts, with an exact LRU cache as the reference for the full model. All experiments run on a single core of an Intel Xeon processor at 2.1 GHz with 3 GB of memory; models are implemented in JAX.

## 6 RESULTS

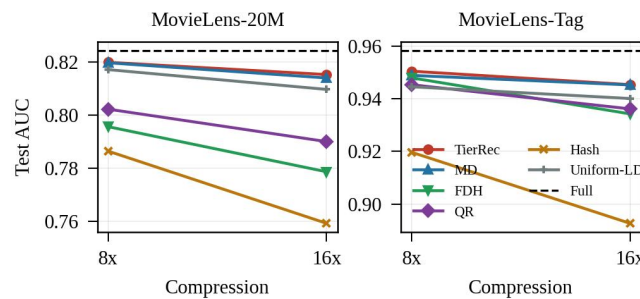
### 6.1 Accuracy Under Memory Budgets

Table 2 and Figure 3 report the accuracy on MovieLens-20M. The full model reaches 0.8243 AUC. At  $8\times$  compression TierRec obtains 0.8199, on par with MD (0.8197), and ahead of Uniform-LD (0.8171), QR (0.8022), FDH (0.7956) and Hash (0.7864). At  $16\times$  the advantage widens: TierRec reaches 0.8153 against 0.8140 for MD and 0.8097 for Uniform-LD, while the collision-based methods lose between 0.0252 and 0.0560 AUC relative to TierRec. Every user in MovieLens-20M has at least 20 ratings, so collisions merge IDs that each carry enough signal to be learned; methods that keep a distinct vector per ID (Uniform-LD, MD, TierRec) are consistently stronger, and among them the frequency-aware allocations win.

**Table 2** Test Accuracy on MovieLens-20M

| Method      | $8\times$ AUC | $8\times$ LogLoss | $16\times$ AUC | $16\times$ LogLoss |
|-------------|---------------|-------------------|----------------|--------------------|
| Full (32-d) | 0.8243        | 0.5106            | 0.8243         | 0.5106             |
| Uniform-LD  | 0.8171        | 0.5191            | 0.8097         | 0.5278             |
| Hash        | 0.7864        | 0.5542            | 0.7593         | 0.5809             |
| QR          | 0.8022        | 0.5367            | 0.7900         | 0.5502             |
| FDH         | 0.7956        | 0.5438            | 0.7787         | 0.5616             |
| MD          | 0.8197        | 0.5160            | 0.8140         | 0.5228             |
| TierRec     | 0.8199        | 0.5159            | 0.8153         | 0.5214             |

The optimizer allocates capacity differently from the power rule. For the user field at  $8\times$  it selects 3,626 hot rows, 83,817 rows of dimension 4 and 51,050 rows of dimension 2; for movies it selects 2,004 hot rows, 6,810 rows of dimension 4 and 9,592 rows of dimension 1, and 8,338 cold IDs. At  $16\times$  the user field receives 57 hot rows, 45,518 rows of dimension 4 and 92,918 rows of dimension 1. Figure 6 contrasts the user allocation with MD, which assigns 32, 15, 6, 2 dimensions to four blocks of increasing size. TierRec instead extends a 4-dimensional tier far into the tail and reserves the smallest dimension for the rarest users, a choice driven by the steep gain of the fidelity profile between one and four dimensions. On MovieLens-Tag (Table 3), TierRec attains the highest AUC among compressed models at both budgets (0.9505 at  $8\times$  and 0.9454 at  $16\times$ ).



**Figure 3** Test AUC Versus Compression Ratio. The Dashed Line Marks the Uncompressed Model

**Table 3** Test Accuracy on MovieLens-Tag

| Method      | $8\times$ AUC | $8\times$ LogLoss | $16\times$ AUC | $16\times$ LogLoss |
|-------------|---------------|-------------------|----------------|--------------------|
| Full (32-d) | 0.9583        | 0.2367            | 0.9583         | 0.2367             |
| Uniform-LD  | 0.9446        | 0.2769            | 0.9401         | 0.2812             |
| Hash        | 0.9197        | 0.3415            | 0.8927         | 0.3881             |

| Method  | 8× AUC | 8× LogLoss | 16× AUC | 16× LogLoss |
|---------|--------|------------|---------|-------------|
| QR      | 0.9454 | 0.2656     | 0.9362  | 0.2790      |
| FDH     | 0.9480 | 0.2584     | 0.9343  | 0.2843      |
| MD      | 0.9489 | 0.2600     | 0.9453  | 0.2683      |
| TierRec | 0.9505 | 0.2626     | 0.9454  | 0.2679      |

## 6.2 Ablation Study

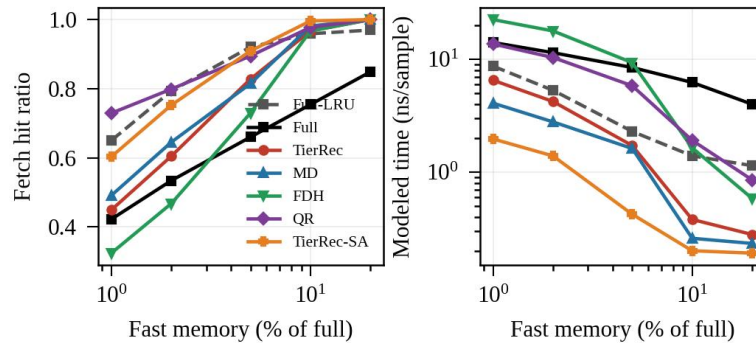
Table 4 isolates the components of TierRec. Profiling on a 10% sample of the training data gives a much flatter profile ( $U(1) = 0.965$  instead of 0.891) because tail IDs observed with a tenth of their data gain little from extra dimensions; the optimizer then assigns single-dimensional vectors to most users, and the AUC drops to 0.8183. Restricting the layer to a single warm tier gives 0.8194, and assigning IDs to tiers at random while keeping the tier sizes gives 0.8064, which confirms that the gain comes from matching capacity to access frequency. TierRec-SA, which constrains the modeled slow traffic to half of the memory-only optimum under a fast memory of 2% of the full footprint, reaches 0.8191, a reduction of only 0.0008 AUC.

**Table 4** Ablation on MovieLens-20M at 8× Compression

| Variant                                   | AUC    | LogLoss | Memory (MB) |
|-------------------------------------------|--------|---------|-------------|
| TierRec (full-data profile)               | 0.8199 | 0.5159  | 2.93        |
| Profile from 10% sample                   | 0.8183 | 0.5177  | 2.92        |
| Single warm level                         | 0.8194 | 0.5164  | 2.93        |
| Random tier assignment                    | 0.8064 | 0.5319  | 2.93        |
| TierRec-SA ( $C = 2\%$ , $\theta = 0.5$ ) | 0.8191 | 0.5168  | 2.63        |

## 6.3 Memory-Hierarchy Behavior

Figure 4 and Table 5 characterize the placement on the time-ordered test trace of MovieLens-20M. With a fast memory of 2% of the full footprint, the full model with an LRU cache serves 79.3% of its row fetches from fast memory and reads 79.4 B per sample from the capacity memory; static frequency placement of the same full rows gives 53.4%. TierRec at 8× serves 60.5% of its fetches from fast memory and reads 64.1 B per sample, a 1.2-fold reduction of slow traffic relative to the LRU-cached full model, with a modeled access time of 4.22 ns per sample against 5.30 ns for the LRU-cached full model. MD reads 41.6 B per sample under the same capacity, and TierRec-SA lowers the traffic further to 19.5 B per sample. At 5% capacity TierRec reaches a hit ratio of 82.8%, compared with 92.1% for the LRU-cached full model, and every compressed configuration becomes fully resident once the capacity exceeds its footprint.



**Figure 4** Trace-Driven Fetch Hit Ratio (Left) and Modeled Access Time (Right) on MovieLens-20M at 8× Compression as a Function of Fast-Memory Capacity

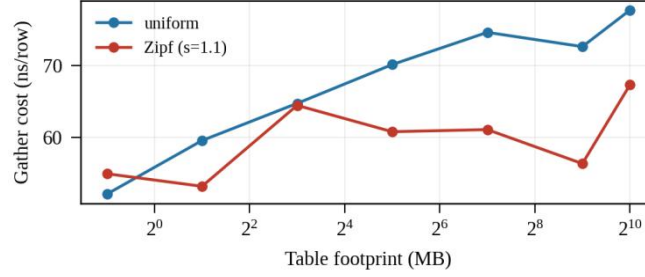
**Table 5** Memory-Hierarchy Behavior on MovieLens-20M (8×,  $C = 2\%$  of the Full Footprint)

| Method       | Size (MB) | Hit ratio | Slow B/sample | Time (ns) | Train (k/s) | Infer (k/s) |
|--------------|-----------|-----------|---------------|-----------|-------------|-------------|
| Full, LRU    | 23.44     | 79.3%     | 79.4          | 5.30      | 258.1       | 594.9       |
| Full, static | 23.44     | 53.4%     | 178.9         | 11.41     | 258.1       | 594.9       |
| Uniform-LD   | 2.93      | 81.8%     | 8.7           | 0.59      | 260.2       | 638.8       |
| Hash         | 2.93      | 60.4%     | 152.0         | 9.76      | 206.3       | 622.2       |
| QR           | 2.93      | 79.9%     | 154.6         | 10.34     | 188.9       | 506.7       |
| FDH          | 2.93      | 46.8%     | 280.0         | 17.77     | 228.3       | 462.1       |
| MD           | 2.72      | 64.5%     | 41.6          | 2.78      | 203.0       | 421.4       |
| TierRec      | 2.93      | 60.5%     | 64.1          | 4.22      | 204.3       | 355.0       |
| TierRec-SA   | 2.63      | 75.2%     | 19.5          | 1.39      | 230.2       | 486.9       |

The measured single-core throughput in Table 5 shows the compute side of the design. The full model trains at 258.1 K samples/s and infers at 594.9 K samples/s. TierRec trains at 204.3 K samples/s and infers at 355.0 K samples/s, and MD,

whose four blocks each require a separate gather and projection, trains at 203.0 K samples/s. Once tables exceed device memory, the host-device transfers modeled above dominate the step time of GPU training [9, 12]. Figure 5 shows why footprint matters on the host as well: the measured cost of gathering a 128-byte row grows from 52.1 ns for a table of 0.5 MB to 77.6 ns for a table of 1 GB under uniform access, and from 55.0 ns to 67.3 ns under Zipf access, because larger tables spill from the caches into DRAM.

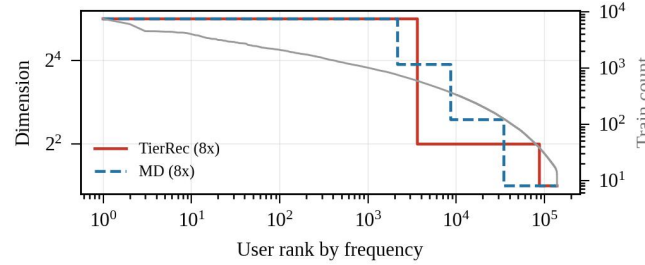
The capacity sweep of Figure 4 shows how the benefit of the systems-aware allocation depends on the size of the fast memory. At 1% of the full footprint TierRec-SA reads 28.9 B per sample from the capacity memory, compared with 62.6 B for MD, 101.4 B for the memory-only TierRec and 134.4 B for the LRU-cached full model; at 5% the corresponding values are 3.8, 22.5, 23.4 and 30.4 B. The configuration of TierRec-SA was selected for a capacity of 2%, yet it remains the most traffic-efficient of the accurate models across the whole sweep, because shrinking the hot tier moves capacity into small warm rows whose access density is high at every capacity. Uniform-LD reads even fewer bytes owing to its uniformly small rows, but it trails TierRec-SA by 0.0020 AUC.



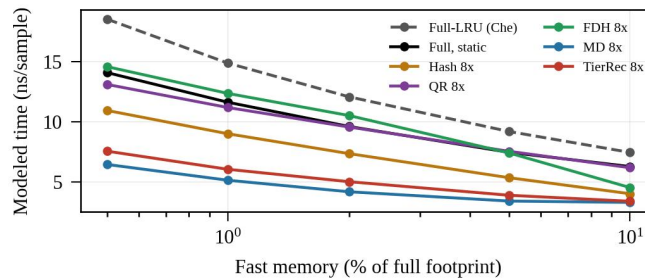
**Figure 5** Measured Row-Gather Cost on the Host CPU as a Function of Table Footprint

#### 6.4 Criteo-Scale Study

Table 6 applies the profile of MovieLens-20M to the Criteo-scale workload. The full model occupies 4.02 GB. With a fast memory of 1% of this footprint, static placement of full rows reaches a hit ratio of 96.1% and an LRU cache 94.5% according to the characteristic-time approximation [24]. TierRec at  $8\times$  shrinks the tables to 0.50 GB and serves 97.2% of its fetches from the same fast memory, lowering the modeled access time from 11.6 ns to 6.0 ns per sample; MD at  $8\times$  reaches 97.4%. At  $16\times$  TierRec reaches 97.8%. Under heavy-tailed access, the compact warm rows of the middle of the distribution fit into fast memory together with the hottest full rows, which is the effect TierRec is designed to exploit. Figure 6-7 plots the modeled access time of all configurations as the fast-memory capacity grows from 0.5% to 10% of the full footprint.



**Figure 6** Dimension Allocated to Each User of MovieLens-20M at  $8\times$  Compression by TierRec and MD (Left Axis), with the Training Frequency of Each Rank (Right Axis, Gray)



**Figure 7** Modeled Access Time on the Criteo-Scale Synthetic Workload Versus Fast-Memory Capacity

**Table 6** Criteo-Scale Synthetic Workload (26 Tables, 33.76 M Rows, Zipf  $s = 1.1$ )

| Method          | Size (GB) | Hit @1% | Hit @2% | Time @1% (ns) |
|-----------------|-----------|---------|---------|---------------|
| Full, LRU (Che) | 4.02      | 94.5%   | 95.9%   | 14.9          |
| Full, static    | 4.02      | 96.1%   | 97.1%   | 11.6          |
| Hash $8\times$  | 0.50      | 97.4%   | 98.2%   | 9.0           |

| Method      | Size (GB) | Hit @1% | Hit @2% | Time @1% (ns) |
|-------------|-----------|---------|---------|---------------|
| QR 8×       | 0.50      | 98.3%   | 98.8%   | 11.2          |
| FDH 8×      | 0.50      | 96.3%   | 97.1%   | 12.3          |
| MD 8×       | 0.49      | 97.4%   | 98.2%   | 5.1           |
| TierRec 8×  | 0.50      | 97.2%   | 98.1%   | 6.0           |
| MD 16×      | 0.23      | 97.7%   | 98.7%   | 4.8           |
| TierRec 16× | 0.25      | 97.8%   | 98.7%   | 4.6           |

## 6.5 Selected Tier Configurations

Table 7 lists the allocations chosen by the optimizer for the three large fields of MovieLens-20M. The strongly skewed item field concentrates its budget on 2,004 full rows and serves the long tail with one-dimensional vectors and cold compositions, whereas the flatter user field receives a broad four-dimensional tier. When the budget halves from 8× to 16×, the optimizer shrinks the hot tiers first and moves rare items into the cold tier, where every ID still owns a distinct composite vector; this is consistent with the fidelity profile, in which even the cold composition retains  $U_c = 0.609$  of the gain of the full model. The access-weighted fidelity  $F$  of each field falls only moderately between the two budgets, which matches the small loss of AUC in Table 2.

**Table 7** Allocations Selected by TierRec on MovieLens-20M

| Field      | Budget | Hot rows | Warm tiers (rows × dim)            | Cold IDs | F     |
|------------|--------|----------|------------------------------------|----------|-------|
| User       | 8×     | 3,626    | $83,817 \times 4, 51,050 \times 2$ | 0        | 0.972 |
| Item       | 8×     | 2,004    | $6,810 \times 4, 9,592 \times 1$   | 8,338    | 0.992 |
| Prev. item | 8×     | 2,004    | $6,811 \times 4, 9,592 \times 1$   | 8,338    | 0.992 |
| User       | 16×    | 57       | $45,518 \times 4, 92,918 \times 1$ | 0        | 0.950 |
| Item       | 16×    | 608      | $4,743 \times 4, 7,284 \times 1$   | 14,109   | 0.980 |
| Prev. item | 16×    | 608      | $4,744 \times 4, 7,282 \times 1$   | 14,111   | 0.980 |

## 7 CONCLUSION

TierRec treats the representation and the placement of embedding tables as a single systems-aware decision. A profiler measures access skew and the fidelity of candidate representations on the full training data, a tier optimizer allocates hot, warm and cold capacity under a memory budget and an optional slow-traffic bound, and placement by access density fills fast memory with the rows that serve the most lookups per byte. On MovieLens-20M TierRec reaches 0.8199 and 0.8153 AUC at 8× and 16× compression, ranks first among the compressed models at 16×, and outperforms collision-based compression by a wide margin, while its hierarchy-aware variant meets a slow-traffic bound of half the memory-only optimum at a cost of 0.0008 AUC. The same allocation principle raises the fast-memory hit ratio on a Criteo-scale workload with 33.76 M rows. Future work will extend the optimizer to online profile updates for streaming training and to placement across more than two memory levels.

Because the tier assignment is a function of frequency ranks only, TierRec composes with existing infrastructure. A sharder such as RecShard can distribute the hot and warm tables across devices using the same access statistics, pipelines that prefetch or cache embedding rows move fewer bytes when most rows are compact, and hierarchical parameter servers can map the cold compositional tables to capacity memory without affecting the frequently accessed rows [10-13].

## COMPETING INTERESTS

The authors have no relevant financial or non-financial interests to disclose.

## REFERENCES

- [1] Rendle S. Factorization Machines. Proceedings of the IEEE International Conference on Data Mining (ICDM), 2010: 995-1000.
- [2] Cheng HT, Koc L, Harmsen J, et al. Wide & Deep Learning for Recommender Systems. Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS), 2016: 7-10.
- [3] Guo H, Tang R, Ye Y, et al. DeepFM: A Factorization-Machine Based Neural Network for CTR Prediction. Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI), 2017: 1725-1731.
- [4] Wang R, Fu B, Fu G, et al. Deep & Cross Network for Ad Click Predictions. Proceedings of ADKDD'17, 2017: 12.
- [5] Zhou G, Zhu X, Song C, et al. Deep Interest Network for Click-Through Rate Prediction. Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2018: 1059-1068.
- [6] He X, Chua TS. Neural Factorization Machines for Sparse Predictive Analytics. Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2017: 355-364.
- [7] Naumov M, Mudigere D, Shi H J M, et al. Deep Learning Recommendation Model for Personalization and Recommendation Systems. arXiv:1906.00091, 2019.

- [8] Gupta U, Wu C J, Wang X, et al. The Architectural Implications of Facebook's DNN-Based Personalized Recommendation. Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA), 2020: 488-501.
- [9] Acun B, Murphy M, Wang X, et al. Understanding Training Efficiency of Deep Learning Recommendation Models at Scale. Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA), 2021.
- [10] Mudigere D, Hao Y, Huang J, et al. Software-Hardware Co-Design for Fast and Scalable Training of Deep Learning Recommendation Models. Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA), 2022.
- [11] Zhao W, Xie D, Jia R, et al. Distributed Hierarchical GPU Parameter Server for Massive Scale Deep Learning Ads Systems. Proceedings of Machine Learning and Systems (MLSys), 2020.
- [12] Adnan M, Maboud Y E, Mahajan D, et al. Accelerating Recommendation System Training by Leveraging Popular Choices. Proceedings of the VLDB Endowment, 2021, 15(1): 127-140.
- [13] Sethi G, Acun B, Agarwal N, et al. RecShard: Statistical Feature-Based Memory Optimization for Industry-Scale Neural Recommendation. Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2022: 344-358.
- [14] Weinberger K, Dasgupta A, Langford J, et al. Feature Hashing for Large Scale Multitask Learning. Proceedings of the 26th International Conference on Machine Learning (ICML), 2009: 1113-1120.
- [15] Shi HJ M, Mudigere D, Naumov M, et al. Compositional Embeddings Using Complementary Partitions for Memory-Efficient Recommendation Systems. Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2020: 165-175.
- [16] Ginart A A, Naumov M, Mudigere D, et al. Mixed Dimension Embeddings with Application to Memory-Efficient Recommendation Systems. Proceedings of the IEEE International Symposium on Information Theory (ISIT), 2021.
- [17] Zhang C, Liu Y, Xie Y, et al. Model Size Reduction Using Frequency Based Double Hashing for Recommender Systems. Proceedings of the 14th ACM Conference on Recommender Systems (RecSys), 2020: 521-526.
- [18] Yin C, Acun B, Liu X, et al. TT-Rec: Tensor Train Compression for Deep Learning Recommendation Model Embeddings. Proceedings of Machine Learning and Systems (MLSys), 2021.
- [19] Kang WC, Cheng D Z, Yao T, et al. Learning to Embed Categorical Features Without Embedding Tables for Recommendation. Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2021.
- [20] Liu S, Gao C, Chen Y, et al. Learnable Embedding Sizes for Recommender Systems. Proceedings of the International Conference on Learning Representations (ICLR), 2021.
- [21] Zhao X, Liu H, Liu H, et al. AutoDim: Field-Aware Embedding Dimension Search in Recommender Systems. Proceedings of the Web Conference (WWW), 2021.
- [22] Harper F M, Konstan J A. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems, 2015, 5(4): 19.
- [23] Breslau L, Cao P, Fan L, et al. Web Caching and Zipf-Like Distributions: Evidence and Implications. Proceedings of IEEE INFOCOM, 1999: 126-134.
- [24] Che H, Tung Y, Wang Z. Hierarchical Web Caching Systems: Modeling, Design and Experimental Results. IEEE Journal on Selected Areas in Communications, 2002, 20(7): 1305-1314.
- [25] Duchi J, Hazan E, Singer Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 2011, 12: 2121-2159.
- [26] Kingma D P, Ba J. Adam: A Method for Stochastic Optimization. Proceedings of the International Conference on Learning Representations (ICLR), 2015.