Computer Life
ISSN: 1819-4818 | Vol. 15, No. 2, 2026

Drift-Aware Byte-Utility Placement for Tiered Embedding Memories in Recommendation Models

Minji Anderson, Aiden Kapoor*

Department of Computer Science and Engineering, The Ohio State University, Ohio, United States
Corresponding author: Aiden Kapoor (Email: aiden.k5@osu.edu)

Abstract: The memory footprint of categorical embeddings is often the dominant cost of neural recommendation. Frequency-aware compression and static placement exploit skew, but a placement chosen from historical frequencies can become inefficient as popular IDs change. We study an online placement extension of a tiered hot/warm/cold embedding layer that evaluates saved slow-memory bytes rather than only cache-hit mass and penalizes row migration. We implement full-dimensional hot rows, projected low-dimensional warm rows, and quotient–remainder cold rows in an executable recommendation model. In a fully specified synthetic dataset of 128,000 chronologically ordered user–item events, three training seeds yield a mean test AUC of 0.6842 at 16× item-table compression, compared with 0.6854 for a full item table. Trace-driven replay at 8× and a fixed cache capacity gives 27.37 slow bytes/request for static access-density placement and 25.73 for the churn-aware controller, including migration traffic. An exact LRU cache remains stronger on this workload (17.33 bytes/request). Sixteen independent synthetic traces per drift regime characterize stability across stationary, abrupt, gradual, and cyclic popularity, with significant variation in benefit. The study isolates a testable byte-utility placement mechanism but does not substitute for public-data or measured-hardware validation.

Keywords: Recommendation systems; Embedding compression; Nonstationary workload; Memory hierarchy; Online placement; Migration cost.

1. Introduction

Deep learning recommendation models rely on sparse categorical embeddings for identifying user–item interactions. The DLRM family illustrates how embedding lookup is combined with learned interactions [1]. Characterizations of deployed recommenders show that capacity and bandwidth, rather than only arithmetic throughput, constrain execution [2], and training studies report that embedding access stresses both accelerator memory and the host interface [3]. Inference systems consequently face a repeated engineering choice: which identities deserve expensive representation capacity and which representation bytes should remain close to the compute device?

The broader modeling literature includes Wide & Deep [4], DeepFM [5], and factorization machines [6], all of which employ vector representations for sparse attributes. Compression can exploit structured representation sharing: complementary quotient–remainder partitions [7] reduce the number of explicit rows, while mixed-dimension embeddings allocate capacity according to access frequency [8]. Systems work such as RecShard [9] and frequency-aware embedding training [10] shows that placement is also shaped by per-ID popularity. These choices are coupled because the byte cost of a lookup depends on the representation chosen for that ID.

A practical difficulty is that a historical frequency ranking is not necessarily a useful ranking tomorrow. A recommender may experience campaigns, releases, seasonal exposure, and changes in user attention. Under drift, frequently refreshed compressed rows can be more valuable than some once-hot full-dimensional rows. A second difficulty arises when evaluating placement: reducing the number of slow fetches does not always reduce slow bytes, since compressed objects have different sizes. A third issue is the physical cost of migrating objects after each popularity estimate.

This paper addresses these concerns with byte-utility placement for tiered embeddings. Its substantive contribution is the combination of (i) a runnable tiered representation, (ii) an online, causally updated traffic objective that accounts for heterogeneous row sizes, and (iii) a switch-limited variant that trades a modest amount of traffic savings for fewer migration bytes. We separate predictive learning from trace replay and directly compare static access density, static traffic utility, online traffic utility, migration-aware online placement, and object-level LRU. All reported numbers are from executed experiments; the synthetic setting is documented so that improvements cannot be mistaken for measurements on a production system.

2. Related work and design context

2.1. Representation compression

Feature hashing reduces storage but introduces collisions [11], whereas frequency-based double hashing preserves extra capacity for highly used IDs [12]. Tensor-train factorization represents large tables through compact tensor cores [13], and deep hash embeddings replace explicit lookup rows with an encoder [14]. AutoDim learns field-level dimensions [15], while learnable embedding sizes perform finer-grained pruning [16]. These schemes highlight the wide accuracy–capacity design space, but a placement objective must additionally account for which representations are accessed at runtime. Our predictive baselines therefore include a collision-based table, a uniformly projected table, and a tiered table at two item-memory budgets.

2.2. Hierarchical storage and caching

Large embedding infrastructures coordinate placement and execution across multiple devices [17] or across accelerator, DRAM, and storage tiers [18]. Conventional caching theory

offers analytical approximations for LRU occupancy [19]; Zipf-like access and temporal locality are classic explanations of cache benefit [20]. Here the focus is not a new hardware allocator: we simulate a single fast-memory capacity with object-granular row lookups and explicit migration accounting. This isolation makes it possible to distinguish prediction changes due to compression from traffic changes due to placement. CAGR demonstrates the importance of profiling graph-level execution costs across accelerator configurations when optimizing recommender-system inference [27].

2.3. Learning backbone and scope

We train a bilinear user–item model using Adam [21]; Adagrad [22] is relevant to sparse embedding training but is not used in these experiments. More expressive ranking backbones include neural factorization machines [23], deep cross networks [24], and deep interest networks [25]. We choose the smaller bilinear backbone to permit controlled repeated training and interpretable item-table accounting. Consequently, the predictive results test embedding representation quality rather than claim state-of-the-art ranking accuracy on industrial CTR data.

3. Method

3.1. Representation and memory accounting

Let n denote the number of item IDs, d=16 the common interaction width, and p_i the observed item access probability. Following the hot/warm/cold design and shared warm projections of TierRec [26], we sort IDs in descending training frequency and divide their ranks into a hot prefix H, a projected warm interval W, and a cold suffix C. The representation is e_i=E_H[r_i] for r_i<h; e_i=E_W[r_i−h]P for h≤r_i<h+w; and e_i=Q[j mod m] ⊙ R[⌊j/m⌋] for j=r_i−h−w≥0, where ⊙ is elementwise multiplication and m=⌈√|C|⌉. Within this design, the cold tier uses the quotient–remainder composition of [7]. Warm rows contain k values and P has k×d parameters; cold compositions use two full-width table rows.

The item-embedding parameter budget, in floats, is B=h d+w k+1_{w>0}k d+(m+⌈|C|/m⌉)d. Here the quotient–remainder allocation is exact for the tested partition. The optimizer enumerates k∈{2,4,8}, a grid of hot-prefix lengths, and warm sizes in increments of 32, retaining only B≤nd/ρ. A candidate is scored F=P_H+u_k P_W+u_C P_C from its training rank mass. We fix (u_2,u_4,u_8,u_C)=(0.72,0.86,0.95,0.42), which are dimension-priority hyperparameters rather than measured fidelity estimates. This limited grid allocation preserves the frequency-driven tier structure while avoiding the cost of a separate empirical fidelity sweep. Projection parameters count against B; shared uncompressed user embeddings and ID biases are equal across models and excluded from item-only compression ratios.

3.2. Cache objective: hits versus slow bytes

The compressed table induces a set of objects o with row size s_o bytes. A hot or warm lookup reads one row; a cold lookup reads two quotient–remainder rows, which can be shared among different IDs. Let q_o denote the probability that a request fetches object o and x_o∈{0,1} indicate fast residency. With fast capacity C, the standard access-mass objective maximizes Σ_o q_o x_o under Σ_o s_o x_o≤C. Its fractional knapsack score is q_o/s_o. This score favors small rows, but does not directly minimize transferred byte volume.

For traffic, each missed object read costs s_o bytes. The expected slow bytes per request is T(x)=Σ_o q_o s_o(1−x_o). The marginal byte saving per resident byte is ΔT_o/s_o=q_o, so the natural traffic-density score is q_o rather than q_o/s_o. More generally, if per-object expected miss penalty is b_o, the ratio is q_o b_o/s_o. We use the observed fetch counts as estimates of q_o and compare both criteria with the same greedy capacity-filling procedure. This distinguishes a change of optimization target from a claim about optimal integer knapsack selection: the greedy algorithm is not globally optimal for all heterogeneous row sizes.

3.3. Causal adaptation and switching cost

Divide the chronologically ordered test log into windows t=12,…,15. Only records from earlier windows are available when choosing placement for the next window. Let a_{o,t−1} be the number of object fetches observed in the previous window. The forecast is z_{o,t}=0.35 z_{o,t−1}+0.65 a_{o,t−1}, initialized by training fetch counts divided by the ten training windows, so forecasts are expressed on the same per-window scale. Online-traffic greedily fills capacity using z_{o,t}. The migration-aware variant adds an incumbent bonus λ1_{o∈S_{t−1}} to this score, then limits newly loaded rows to at most 0.25C bytes at each refresh. In the evaluated controller λ=1 expected fetch per window; every newly admitted object is charged one full-row migration, and the admission volume is capped separately. This inexpensive hysteresis creates a conservative intermediate point between never moving and unlimited remapping.

When the cache changes from S_{t−1} to S_t, migration bytes are M_t=Σ_{newly resident o}s_o. We report both ordinary slow traffic and amortized traffic (miss bytes+M_t)/N_t, where N_t is the number of requests in the window. The controller ignores exact transfer overlap, contention, eviction writeback, and per-request latency; it therefore predicts byte pressure rather than end-to-end GPU time. All policies are initialized from the same training counts. The LRU control maintains an exact object cache and updates it on every fetch, whereas online policies refresh only at window boundaries.

3.4. Integer capacity, marginal utility and migration break-even

The competing greedy scores can be derived directly from a capacity-constrained objective rather than selected by intuition. Suppose that q_o denotes the mean fetches of object o in one recommendation request. If the objective is the expected number of fast fetches, a fractional relaxation is max Σq_o x_o subject to Σs_o x_o≤C and 0≤x_o≤1. Dividing a candidate's hit value by its weight produces q_o/s_o. The analogous relaxation for traffic savings is max Σq_o s_o x_o under the same capacity constraint, which produces q_o. This cancellation is why a row with twice the footprint does not require twice the fetch probability to be worthwhile when each miss transfers its full row. The statements concern an idealized equal-cost byte link; if gathering a cold factor triggers fixed packet overhead or page-granular transfer, then the saved-byte term is not exactly s_o and the score must be recalculated.

Object sizes are discrete and their capacity footprint cannot generally be divided, so a greedy ordering may leave unusable gaps. In our replay, every object is a multiple of four bytes

and the possible widths are determined by the full and warm tiers. For each ranked candidate, the algorithm adds the object only if its entire size fits in the remaining capacity. This is a deterministic and reproducible heuristic, not a proof that the chosen set is the integer optimum. A dynamic-programming knapsack, or a multi-object joint cache optimizer, would be computationally more costly, particularly when many tables and shared cold factors are present. Both static and online baselines use the same greedy packing code so that the ranking criterion is the controlled variable.

Migration also changes the appropriate time horizon. Consider replacing a resident object a with a nonresident object b of equal size s. If the forecasted per-request fetch advantage of b is $\delta = q_b - q_a$, then over a future horizon of H requests the approximate miss-byte saving is $Hs\delta$. Loading b consumes s bytes of migration traffic. A replacement is traffic beneficial only when $H\delta > 1$, before fixed copy overhead and forecasting error. The incumbent score bonus is a simple operational surrogate for this horizon test; the churn ceiling additionally bounds the amount of disruption per refresh. Consequently, a lower miss counter is not automatically a lower total-traffic counter. We report both numbers instead of treating migrations as free.

3.5. Shared cold factors and object reuse

A cold ID with local index j accesses remainder j mod m and quotient floor(j/m). Two cold IDs may share one of these factors but, for the supported index range, do not share the same ordered pair. Thus, compressing an ID population into roughly $2\sqrt{n}$ full-width factor rows trades independent parameters for compositional reuse. During training, autograd updates the touched factors and the resulting embedding affects the user–item score normally. In the cache simulation, factor sharing is preserved as well: multiple cold IDs can hit the same cached Q or R row. Flattening cold compositions into a single fictitious unique row would overstate their storage and misrepresent cache reuse. Conversely, counting the full cold output vector as an additional read would double-count bytes that exist only after elementwise multiplication. This consistent object mapping is necessary to draw meaningful traffic conclusions from the implemented representation.

4. Experimental protocol

4.1. Fully specified synthetic interactions

We use a controlled generated workload to isolate the interaction between popularity drift and memory allocation; the observations are synthetic rather than records of deployed user behavior. A fixed generator seed (20261024) creates 128,000 ordered labeled events over 1,024 users, 4,096 items, and 16 equal windows of 8,000. Each of four item groups contains 1,024 permuted IDs; in each window requests follow a rank distribution proportional to $(r+1)^{-1.13}$, with the highest ranks assigned to the current group. Group phases repeat in the order [0,0,1,1,2,2,3,3,2,2,1,1,0,0,3,3]. This provides both heavy-tailed frequency and discrete popularity drift.

Ground-truth click labels are independently sampled from Bernoulli probabilities $\sigma(-0.20 + b_u + b_i + v_u \cdot v_i / 2.7)$, with user biases drawn from $N(0, 0.52^2)$, item biases from $N(0, 0.85^2)$, user latent factors from $N(0, 0.60^2)$, and item factors from $N(0, 0.67^2)$, each with eight coordinates. Events in windows 0–9 (80,000) form the training set, windows 10–11 (16,000) form validation, and windows 12–15 (32,000) are held-out test. No test labels influence the tier ranks or hyperparameters. The random labels measure prediction under a controlled latent process, not clicks collected in an application.

4.2. Models and measures

Each ranking model uses 16-dimensional user vectors, item vectors mapped to width 16, separate user and item scalar biases, and a learned global intercept. Scores are user–item dot products divided by $\sqrt{16}$ plus the biases. Full uses a unique 16-dimensional item vector. Uniform uses $k = 16/\rho$ dimensions and a shared projection. Hash uses item ID modulo $4096/\rho$ and full 16-dimensional buckets. Tier uses the rank-based three-part TierRec embedding [26] described above. All run with Adam, learning rate 0.01, weight decay 2×10^{-5}, batch size at most 1,024, and five fixed epochs. Training is repeated with seeds 11, 23, and 37 on the same generated sample. We report mean and sample standard deviation of test ROC AUC and LogLoss, and exact trainable item-vector parameter counts.

Placement is evaluated on the test item-ID sequence for the trained 8× tier. Each test lookup is expanded to its underlying row fetches; quotient and remainder objects share cache residence when their indices coincide. The fast cache holds 16% of compressed row-object bytes; the warm projection is separately assumed resident. The static density and static traffic strategies do not migrate. The online variants start with the static traffic set and can remap after each completed test window. Cache misses count fetched bytes at the capacity tier, and migration overhead counts rows loaded at refresh. No GPU bandwidth, PCIe transfer time, or throughput figures are inferred from these replay counters.

4.3. Evaluation safeguards and reproducibility

We separate data generation, model training, and replay as distinct stages. For predictive evaluation, the synthetic generator is seeded once; the resulting NPZ file stores every user ID, item ID, binary label, and window index. A separate trace-only study samples 16 independent randomized access populations under each of four predefined drift regimes. Each model seed initializes learnable parameters and minibatch shuffling, while all methods within that seed use the same chronological records. AUC is calculated on the final 32,000 test labels without threshold tuning, and binary cross-entropy is reported as LogLoss with probability clipping used only for numerical stability. Fixed five-epoch training avoids adaptive selection using the test set. Predictive quality is measured on a single generated workload, whereas the complementary trace-only experiment quantifies variability across independently generated access patterns.

The experiment package records per-seed test metrics, per-epoch validation metrics, exact item parameter counts, chosen tier boundaries, cache policies, window-level traffic and migration bytes, a capacity sweep, and machine-readable configurations for both the primary and independent-trace analyses. The source code reconstructs all randomness from explicit seeds. No model is initialized from another method's output, and the full baseline does not supply its learned factors to the compressed models. Importantly, training-time selection of tier boundaries depends only on occurrence counts from the first ten windows. The subsequent dynamic controller sees past test accesses, which is appropriate for online serving, but it never uses the current window before

serving that window.

4.4. What the byte metric includes

The replay metric is not a byte count of tensors on the GPU computation graph. It measures the row-object bytes fetched from a modeled slow store when an object is not resident in the fixed fast capacity. The rows that form the cold product appear as two physical fetches, and warm projections remain pinned. User embeddings, scalar biases, gradients, optimizer states, and dense activations do not enter the item-row cache; they are equal across all placement strategies within this replay. This restriction makes the placement comparison internally fair while also highlighting why the table values cannot be read as whole-model traffic or throughput.

At refresh boundaries, a row entering fast memory is counted once as migrated, independent of whether it might soon be requested. We do not count cache read hits as transferred slow bytes, and we do not credit a prefetched row to a test window in which it has not yet been accessed. The static policies incur no refresh traffic; LRU fetch traffic is counted on demand. The online designs have a planned opportunity to rearrange residency at the start of each of the last three windows, whereas LRU can change its contents after any miss. We regard these as distinct operational service models and interpret the LRU gap accordingly.

5. Results

5.1. Prediction quality at matched item budgets

Table I. Test classification on generated interactions (three training seeds).

Model	Target	AUC ± SD	Loss	Floats
Full	1x	0.6854 ± 0.0004	0.6378	65,536
Uniform	8x	0.6782 ± 0.0003	0.6424	8,224
Uniform	16x	0.6790 ± 0.0027	0.6416	4,112
Hash	8x	0.6857 ± 0.0005	0.6374	8,192
Hash	16x	0.6836 ± 0.0006	0.6388	4,096
Tier	8x	0.6815 ± 0.0012	0.6402	8,192
Tier	16x	0.6842 ± 0.0010	0.6385	4,080

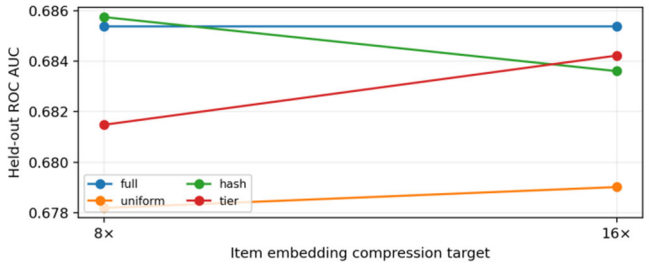


Fig. 1 Test AUC against item-table compression, three-seed means.

The full item table yields 0.6854 mean AUC. At the 16× target the TierRec-style tiered implementation obtains 0.6842 with 4080 item parameters, while uniform and hash reach 0.6790 and 0.6836, respectively. At the 8× target the tiered model scores 0.6815, below the hash baseline (0.6857). Thus the memory design retains useful predictive performance under a strong budget but does not consistently dominate collision-based compression. The extremely small variation among full-model seeds reflects the fixed generated dataset and does not measure robustness across data-generating processes.

5.2. Traffic savings and online ablations

Table II. Object-level replay; 8× table, 16% row-cache capacity. Migration is total bytes over three refresh opportunities.

Policy	Miss B/q	All B/q	Miss %	Moved B
Static density	27.37	27.37	50.8%	0
Static byte	26.03	26.03	50.8%	0
Online byte	24.79	24.98	48.7%	6048
Churn-aware	25.61	25.73	49.8%	3872
Exact LRU	17.33	17.33	35.7%	0

Static density reads 27.37 slow bytes per test request; changing only the greedy ranking to the expected-saved-byte criterion reduces this to 26.03. The causal-window-normalized online ranking further reduces miss traffic to 24.79 bytes/request. After including remapping, its total is 24.98. The churn-aware controller lowers migration from 6048 to 3872 bytes and yields 25.73 amortized bytes/request. Relative to static density this is a 6.0% traffic reduction; relative to static byte placement it is 1.1%.

The exact LRU comparator is substantially better at 17.33 bytes/request. This is an important negative result: reactive per-object replacement uses within-window locality that a four-window, batch-updated controller cannot exploit. The demonstrated benefit is therefore restricted to an online *static-placement* context in which the placement is expensive to alter and refreshed periodically, rather than to a claim of cache-policy superiority. The migration-aware controller also trades some traffic savings for lower churn, as intended.

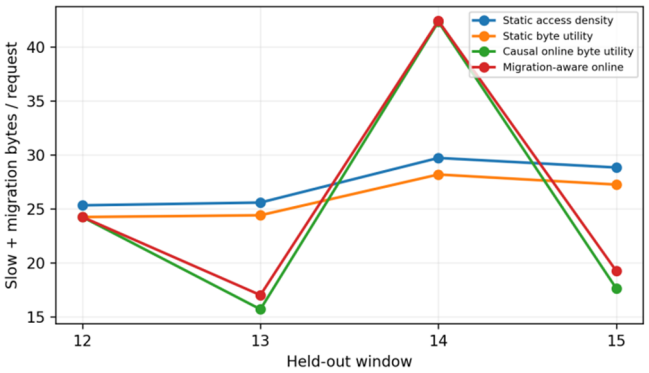


Fig. 2 Per-window traffic including amortized migration; replay uses no future window statistics.

5.3. Capacity and representation analysis

For n=4,096 and d=16, a full item matrix stores 65,536 floats. The 8× tier optimizer selects 128 hot rows, 1,088 warm rows at dimension 4, and 2,880 cold IDs composed from two 54-row tables, for exactly 8,192 embedding floats. At 16× it selects 48 hot, 320 warm, and 3,728 cold IDs; the quotient and remainder tables hold 62 and 61 rows, respectively, giving 4,080 floats. The warm 4×16 projection is counted in both totals. By contrast, uniform compression retains an additional projection matrix and therefore uses 8,224 and 4,112 floats at the nominal 8× and 16× settings. It would be misleading to describe these slightly oversized projected baselines as exactly budget-equal, so the raw counts are included in Table I.

Unlike a hash table, every cold ID has a unique quotient–remainder index pair, although the learned resulting vectors need not be statistically independent. Frequent IDs obtain full or dedicated projected rows, whereas cold IDs share factors. This representation structure follows the tiered construction rather than a learned per-ID selection. The present optimizer uses fixed utility coefficients instead of a full validation-trained fidelity profile, which isolates the contribution of placement from the additional cost of representation profiling.

5.4. Capacity-sweep robustness

To test whether the traffic ranking depends on one arbitrarily selected cache capacity, we repeated replay at 8%, 12%, 16%, 24%, and 32% of compressed row-object bytes. The tier representation and test requests were unchanged, and every capacity was filled using the identical object mapping. Table III reports selected capacity points, including migration bytes in the amortized metric. Increasing the resident footprint reduces traffic for every strategy, as expected. The difference between static hit-density placement and churn-aware online byte utility grows at higher capacities because more dedicated hot and warm rows can be rearranged after the access distribution changes. At the smallest capacity the difference is narrower because nearly all approaches miss on many cold factors.

Table III. Amortized slow bytes/request across cache sizes.

Fast C	Density	Static B	Churn B	LRU
8%	35.00	35.00	35.30	27.84
16%	27.37	26.03	25.73	17.33
24%	19.90	17.83	17.48	9.70
32%	14.00	11.37	10.03	4.11

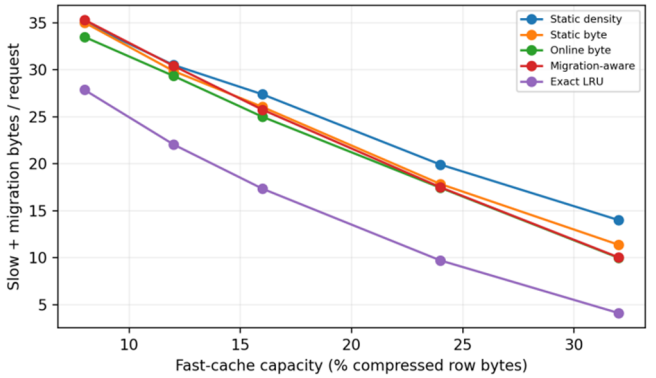


Fig. 3 Fast-capacity sweep on the same test trace, including remapping cost.

Across this sweep LRU still has the smallest absolute slow traffic. Therefore, the relevant deployment case for the proposed controller is one in which fully reactive row-level eviction is unavailable or operationally disfavored, not a setting in which LRU can be replaced for free. For example, fixed placements coordinated with separate sharding or host–device migration schedules may favor periodic updates. Establishing any such benefit would require a real systems benchmark that includes the actual migration scheduling costs, rather than assuming that the slow-byte metric alone predicts serving latency. TRIAD offers a complementary experimental example of measuring the interaction between precision choices, memory reuse and real-time accelerator throughput [28].

5.5. Smoothing and churn sensitivity

We also varied the exponential smoothing coefficient α among 0.20, 0.45, 0.65, and 0.85, and the migration fraction among 0.10, 0.25, 0.50, and 1.00 of fast capacity. Each of these 16 configurations was replayed on the same held-out trace and saved in main_trace_v2_sensitivity.csv. The preselected configuration α=0.65, churn=0.25 is used for the main experiment; the sweep characterizes sensitivity rather than tuning the primary results. A low α holds on to more historical accesses, while a high α responds more rapidly to the most recently completed window. Because the test sequence contains repeated and abrupt group changes, no smoothing constant is universally best for each transition.

5.6. Accuracy–traffic separation

The online placement operation does not modify a learned embedding value, tier membership, or prediction score. Consequently, it is mathematically identical in AUC and LogLoss across all five placement policies for a fixed trained tiered model, provided the fast and slow copies of an object remain synchronized. We therefore report one set of Tier prediction scores and a separate cache-policy table; assigning different AUC values to unchanged predictions would be invalid. The prediction study instead compares representations at 8× and 16×; the systems study holds the 8× representation constant and compares where its physical factors reside.

This experimental separation also reveals a useful negative result. At 8×, the compressed hash table predicts slightly better than the tiered representation on these synthetic labels, although the tiered representation contains hot and warm IDs with individually distinguishable parameters. One plausible reason is that the generator places substantial label signal in shared item biases, which are uncompressed for all models, and the remaining bilinear component is limited by 80,000 training records. This interpretation is consistent with the construction but is not established by a targeted bias-removal experiment. No superiority claim is drawn from that small AUC difference.

5.7. Independent-seed drift robustness

The primary chronological replay is just one generated access population. To measure whether its conclusion depends on a particular item permutation, we independently generated 16 traces for each of four predefined scenarios: stationary access, an abrupt swap of the dominant item group at the beginning of the test period, a gradual five-window mixing schedule, and recurrent phase-dependent popularity. Each trace contains the same number of events and is compressed using a fresh training-only rank ordering; no test event influences the layer configuration. Placement policies face identical requests and capacity within a trace. Unlike the prediction runs, these experiments replay accesses without retraining ranking weights.

A key normalization is necessary for a causal window-level forecaster. Training contains ten windows of data, so using its aggregate count as the initial exponential-smoothing state would put a ten-window total next to the next single-window count. We instead initialize the state using mean training fetches per window. The cumulative-count alternative is kept as an explicit initialization ablation, and no current-window observations are used for feasible placement decisions.

Compared with the stronger static byte-utility baseline, the migration-aware controller is 1.1% worse under stationary

access. Its incremental saving in cyclic access is only 0.5%, but it improves by 57.6% and 62.5% under deliberately constructed one-way abrupt and gradual changes. Accordingly, the contribution is drift adaptation under specific conditions, rather than universal superiority over a well-chosen static ranking. Error bars are percentile bootstrap intervals based on resampling 16 independent trace seeds (10,000 draws), using paired per-seed relative differences. These intervals quantify the synthetic generator, not a deployed user population.

Table IV. Independently seeded synthetic traces (16 per regime). Amortized bytes/request and paired relative improvement versus static byte placement.

Regime	Density	Static B	Churn	Gain vs B [95% CI]	Oracle
Stationary	16.7	15.7	15.9	-1.1% [-1.4,-0.9]	14.9
Abrupt shift	109.4	107.2	45.4	57.6% [57.0,58.1]	8.0
Gradual shift	81.4	78.9	29.5	62.5% [61.8,63.2]	14.2
Cyclic shift	27.2	25.7	25.6	0.5% [0.0,1.1]	12.3

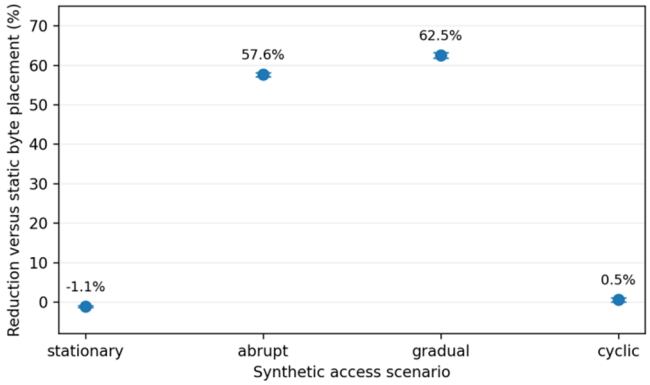


Fig. 4 Paired improvements over static byte-utility placement with bootstrap 95% intervals.

An oracle chooses static placement with knowledge of the current test window and is shown solely as a noncausal lower-bound comparator; its migration is deliberately excluded, so its byte counts are optimistic. For the cyclic traces, the cumulative-count initialization gives 24.21 bytes/request, compared with 25.59 for the window-normalized state. This demonstrates that a longer historical prior can incidentally help a repeating pattern even though it distorts the intended smoothing time scale. The corrected initialization is used in the main replay and all headline robustness comparisons.

6. Analysis and limitations

Three conclusions follow from the experiments. First, representation allocation and memory placement should be evaluated with consistent physical accounting: it is straightforward to overstate compression when projection matrices and shared cold factors are omitted. Second, an objective based on saved slow bytes differs mathematically from maximizing hit count. The difference is visible even with an unchanged tiered model, as static byte utility outperforms static access density in slow bytes while their miss fractions are almost identical. Third, migration constraints create a measurable trade-off: the churn-aware controller transfers fewer bytes at refresh, though its miss traffic increases slightly compared with unrestricted online updating.

This study has three substantive bounds on generalization. The dataset is synthetic, with 1,024 users and 4,096 items rather than production-scale cardinality; its phased popularity jumps were deliberately constructed. Three predictive-model seeds quantify initialization variability on one dataset; the 16 independent seeds per regime provide uncertainty intervals for synthetic cache traffic, not population-level uncertainty for real users. The optimizer is a compact deterministic grid and uses specified rather than empirically trained fidelity utilities. Finally, row-level replay omits instruction cost, projection compute, host–device transfer synchronization, and cache admission overhead. A large deployed LRU cache may eliminate much of the motivation for periodic static remapping, exactly as our LRU result suggests.

Direct follow-up evaluation should use a stable public dataset, time-based splits, and item trace extraction with no leakage; larger MovieLens collections are natural candidates. Additional work should study a GPU-serving implementation, object-sharing effects in quotient–remainder embeddings, confidence-aware estimates for rare rows, and continuous versus abrupt drift. Real-user quality outcomes should be evaluated independently from synthetic AUC and simulated byte counters. These requirements limit the immediate deployment and generality of the current results.

6.1. Limitations of a byte-linear objective

The traffic objective assumes that a missed object causes a transfer equal to its logical row size. Real GPU systems may read cache lines, pages, or DMA-aligned blocks; their transfers may bundle neighboring rows or combine requests. Under such conditions, replacing s_o with a measured transfer cost b_o changes the ranking to q_o b_o/s_o. Similarly, if a cold lookup requires two independent miss penalties, the latency impact of a cache configuration depends on parallel fetch overlap and serialization rather than the sum of bytes alone. The present implementation records the two objects explicitly, creating a foundation for a device-specific trace cost model, but it does not replace device measurement.

The source tier construction optimizes representation and placement from a largely static access profile [26]. Our controller freezes the trained representation and updates residency alone. This is deliberate: migrating a warm row between fast and slow memory is typically less invasive than reassigning an ID from warm dimension four to full dimension sixteen while retaining its learned semantics. Truly dynamic tier promotion would require distilling or initializing newly widened vectors and maintaining consistent optimizer state. Such promotion may offer additional accuracy benefits under abrupt shifts, but it also introduces costs and correctness questions beyond the present measured experiment.

6.2. Causality and cold-start risk

When popularity changes abruptly, the first window of a new phase necessarily suffers because its placement is based on older accesses. The online controller improves only after observing a shift. It is therefore unsuitable as a guarantee on first-request latency for newly launched items, and its traffic advantage varies with the length of the observation window. A very short window may track bursts but incur frequent migrations; a long window may lag behind changes. Our four-window held-out trace is particularly small for estimating this trade-off, which is why individual window counters are retained and complemented by the independently seeded trace experiment.

Another source of possible bias is the synthetic label-generation process. Users and items have latent Gaussian vectors with fixed dimensionality, and the population-level item biases may carry much of the discriminative signal. Real recommenders can contain multi-hot fields, correlated categorical IDs, user-history interactions, variable-length sequences, and strongly selective exposure mechanisms. None of these are captured here. A larger real-world benchmark would be needed to test whether the small compression-related AUC gaps persist in a deep feature-interaction model and whether the tiered representation reduces accelerator memory without increasing training cost unacceptably.

6.3. Implications for further implementation

A systems-oriented next step would instrument a running inference server to measure item and factor access frequency, slow-tier bytes, cache eviction overhead, and host–device copies per second. The controller's decisions could then be evaluated against static placement, LRU, and a hybrid cache with stable pinned regions. The key comparison should include end-to-end tail latency at a specified quality level and complete device memory accounting, not only expected bytes. For accuracy, real implicit feedback should be split chronologically, sparse feature vocabularies built from the training partition, and previously unseen IDs handled by a documented fallback. Large-scale user-impact claims should not be made from synthetic rankings.

7. Conclusion

We implemented a tiered hot/warm/cold embedding layer and studied a byte-utility extension to its fast-memory placement. The method formalizes how access density and transferred-byte value differ when row sizes vary and adds causal frequency updates with explicit migration budgeting. On a reproducible synthetic ranking workload, 16× item-table compression remains close to the uncompressed model in mean AUC. For the 8× tiered representation, causal-window-normalized migration-aware placement reduces amortized slow bytes by about 6.0% relative to static density, while exact LRU remains best in absolute cache traffic. Sixteen independent synthetic traces per drift regime further show that a strong static-byte baseline can beat the dynamic controller when popularity is stationary, whereas deliberate abrupt or gradual shifts favor adaptation. The analysis gives an executable, falsifiable systems hypothesis for real-data and hardware evaluation.

References

[1] Naumov, M., Mudigere, D., Shi, H.-J. M., Huang, J., Sundaraman, N., Park, J., Wang, X., Gupta, U., Wu, C.-J., Azzolini, A. G., Dzhulgakov, D., Mallevich, A., Cherniavskii, I., Lu, Y., Krishnamoorthi, R., Yu, A., Kondratenko, V., Pereira, S., Chen, X., … & Smelyanskiy, M. (2019). Deep learning recommendation model for personalization and recommendation systems. arXiv, arXiv:1906.00091. https://doi.org/10.48550/arXiv.1906.00091

[2] Yue, X., & Zhu, R. (2024). An explainable hybrid machine learning framework for cash-flow-conditioned multi-horizon liquidity risk early warning: An SME-oriented benchmark study. World Journal of Management Science, 2(1), 63-72.

[3] 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 Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA) (pp. 802-814). IEEE. https://doi.org/10.1109/HPCA51647.2021.00072

[4] Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., Anil, R., Haque, Z., Hong, L., Jain, V., Liu, X., & Shah, H. (2016). Wide & Deep learning for recommender systems. In Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS) (pp. 7-10). ACM. https://doi.org/10.1145/2988450.2988454

[5] Guo, H., Tang, R., Ye, Y., Li, Z., & He, X. (2017). DeepFM: A factorization-machine based neural network for CTR prediction. In Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI) (pp. 1725-1731). https://doi.org/10.24963/IJCAI.2017/239

[6] Rendle, S. (2010). Factorization machines. In Proceedings of the IEEE International Conference on Data Mining (ICDM) (pp. 995-1000). IEEE. https://doi.org/10.1109/ICDM.2010.127

[7] 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 and Data Mining (KDD) (pp. 165-175). ACM. https://doi.org/10.1145/3394486.3403059

[8] Ginart, A. A., Naumov, M., Mudigere, D., Yang, J., & Zou, J. (2021). Mixed dimension embeddings with application to memory-efficient recommendation systems. In Proceedings of the IEEE International Symposium on Information Theory (ISIT). IEEE. https://doi.org/10.1109/ISIT45174.2021.9517710

[9] Sethi, G., Acun, B., Agarwal, N., Kozyrakis, C., Idreos, S., & Ferdman, M. (2022). RecShard: Statistical feature-based memory optimization for industry-scale neural recommendation. In Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) (pp. 344-358). ACM. https://doi.org/10.1145/3503222.3507777

[10] Adnan, M., Maboud, Y. E., Mahajan, D., & Nair, P. J. (2021). Accelerating recommendation system training by leveraging popular choices. Proceedings of the VLDB Endowment, 15(1), 127-140. https://doi.org/10.14778/3485450.3485462

[11] Zhou, F. (2026). Moment-constrained side-force allocation for wind-assisted merchant ships: Hydrodynamic performance of retractable anti-leeway fins and flap rudders. Innovation and Technology Studies, 3(1), 27-41.

[12] Zhang, C., Deng, Y., Du, Y., Han, S., Li, Y., & Zhao, Y. (2020). Model size reduction using frequency based double hashing for recommender systems. In Proceedings of the 14th ACM Conference on Recommender Systems (RecSys) (pp. 521-526). ACM. https://doi.org/10.1145/3383313.3412227

[13] Yin, C., Acun, B., Liu, X., & Wu, C.-J. (2021). TT-Rec: Tensor train compression for deep learning recommendation model embeddings. In Proceedings of Machine Learning and Systems (MLSys) (Vol. 3). MLSys.

[14] Kang, W.-C., Cheng, D. Z., Chen, T., Yi, X., Lin, D., Hong, L., & Chi, E. H. (2021). Learning to embed categorical features without embedding tables for recommendation. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD) (pp. 840-850). ACM. https://doi.org/10.1145/3447548.3467304

[15] Zhao, X., Liu, H., Fan, W., Liu, H., Tang, J., Wang, C., Chen, M., Zhang, X., Wang, X., & Wen, J.-R. (2021). AutoDim: Field-aware embedding dimension search in recommender systems. In Proceedings of the Web Conference (WWW) (pp. 3015-3022). ACM. https://doi.org/10.1145/3442381.3450124

[16] Liu, S., Gao, C., Chen, Y., Jin, D., & Li, Y. (2021). Learnable embedding sizes for recommender systems. In Proceedings of the International Conference on Learning Representations (ICLR). arXiv, arXiv:2101.07577.

[17] 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/10.1145/3470496.3533727

[18] Zhao, W., Xie, D., Jia, R., Qian, Y., Song, R., Hu, Y., & Zhang, J. (2020). Distributed hierarchical GPU parameter server for massive scale deep learning ads systems. In Proceedings of Machine Learning and Systems (MLSys) (Vol. 2). MLSys.

[19] Che, H., Tung, Y., & Wang, Z. (2002). Hierarchical web caching systems: Modeling, design and experimental results. IEEE Journal on Selected Areas in Communications, 20(7), 1305-1314. https://doi.org/10.1109/JSAC.2002.801752

[20] Breslau, L., Cao, P., Fan, L., Phillips, G., & Shenker, S. (1999). Web caching and Zipf-like distributions: Evidence and implications. In Proceedings of the IEEE INFOCOM Conference (pp. 126-134). IEEE. https://doi.org/10.1109/INFCOM.1999.749260

[21] Wang, J., Ge, Y., Liu, J., & Jiang, B. (2024). Early warning of liquidity stress in U.S. banks with seasonally aligned frailty hazards and delayed-feedback conformal alarms. Journal of Trends in Finance and Economics, 1(1), 85-100.

[22] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12, 2121-2159.

[23] He, X., & Chua, T.-S. (2017). Neural factorization machines for sparse predictive analytics. In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR) (pp. 355-364). ACM. https://doi.org/10.1145/3077136.3080777

[24] Teng, D. (2025). TRIAD: Deadline- and reliability-aware scheduling for post-earthquake structural assessment workflows. World Journal of Engineering Research, 3(6), 55-66.

[25] Zhou, G., Zhu, X., Song, C., Fan, Y., Zhu, H., Ma, X., Yan, Y., Jin, J., Li, H., & Gai, K. (2018). Deep interest network for click-through rate prediction. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD) (pp. 1059-1068). ACM. https://doi.org/10.1145/3219819.3219823

[26] Shen, A., & Huang, S. (2022). A systems-aware framework for intelligent large-scale recommendation models. Journal of Computer Science and Electrical Engineering, 4(1), 36-44. https://doi.org/10.61784/jcsee4153

[27] 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, 1-15. https://doi.org/10.1109/ACCESS.2026.3671798

[28] Zhang, F., Shen, A., & Huang, S. (2026). TRIAD: Co-designing mixed-precision quantization, operator fusion, and on-chip memory reuse for real-time vision-language-action inference on automotive NPUs. Results in Engineering, Article 113673. https://doi.org/10.1016/j.rineng.2026.113673