IDSHARE: LEARNED SHARING AS REGULARIZATION FOR LONG-TAILED ID EMBEDDINGS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recommendation models often use ID features with many distinct values. Features such as user IDs and ad IDs can have long-tailed frequency distributions. Long-tailed IDs have few training examples, making their embeddings difficult to learn reliably. We introduce IDShare, a simple end-to-end method that learns which IDs share an embedding. Each shared embedding is updated using training examples from all IDs assigned to it. We find that this learned sharing provides effective regularization for long-tailed IDs. On two recommendation benchmarks, IDShare achieves higher mean AUC than independent embeddings with tuned global L2, without using that penalty. Further analysis shows that IDShare improves predictions for long-tailed IDs. Its performance remains stable across different numbers of shared embeddings. It also retains its advantage in deeper models. These findings support learned sharing as a simple and stable default for training long-tailed ID embeddings.

1 INTRODUCTION

Recommendation models often use ID features with many distinct values. User IDs and ad IDs, for example, can follow long-tailed frequency distributions. A common practice in industrial recommendation is to learn a separate embedding for each ID (Naumov et al., 2019; Zhou et al., 2018). Yet long-tailed IDs appear in few training examples. Their independent embeddings are therefore difficult to learn reliably and can overfit. We call this standard representation Continuous and use it as our baseline.

We propose IDShare, a simple end-to-end method that learns which IDs share an embedding.[1] Each ID learns a routing state that selects an embedding from a shared codebook. The recommendation backbone receives the selected shared embedding. Figure 1 contrasts Continuous and IDShare with the backbone and prediction objective held fixed.

Our central finding is that learned sharing provides effective regularization for long-tailed ID embeddings. With N IDs and $K \ll N$ shared embeddings, the backbone sees at most K identity vectors instead of N independent ones. Each shared vector receives training updates from all IDs assigned to it. A long-tailed ID can therefore learn through examples from other IDs. The prediction task determines which IDs share, adapting this constraint during training.

An alternative is to keep embeddings independent and regularize their values. Global L2 penalizes their squared magnitudes using a tunable coefficient. AdamAR adapts shrinkage to each ID's update history (Li & Lyu, 2026). We also compare two fixed sharing rules. TailShare keeps frequent IDs independent and pools IDs below a training-frequency threshold into one embedding. FrozenHash assigns IDs to shared embeddings using a fixed hash. Together, these comparisons test learned sharing against both embedding penalties and predefined sharing groups.

We evaluate on two large-scale recommendation datasets. TAAC contains logs from the Tencent Advertising Algorithm Competition 2025 (Tencent Advertising Algorithm Competition, 2025). KuaiRand contains video recommendation logs from Kuaishou (Gao et al., 2022). The main backbones are based on RankMixer (Zhu et al., 2025), an architecture developed for industrial recommendation. Each backbone is held fixed across methods. On both datasets, IDShare achieves higher

[1] Anonymous code: https://anonymous.4open.science/r/IDShare-7C2E/.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

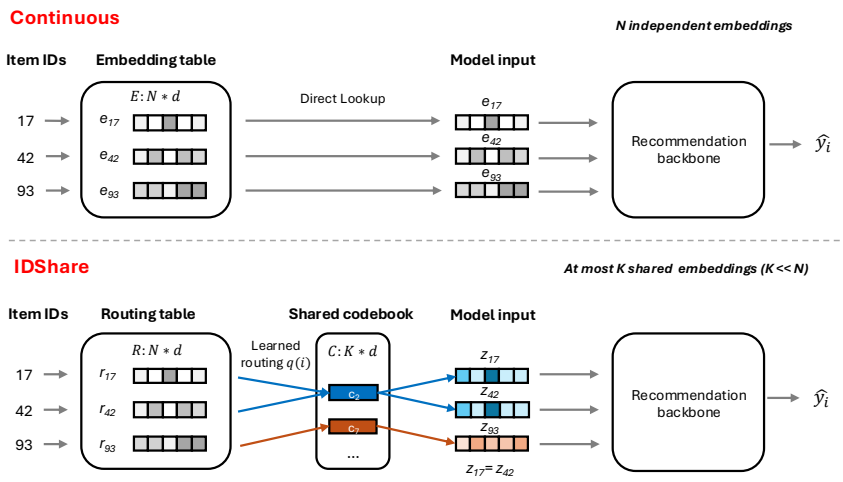


Figure 1: **Continuous embeddings and IDShare.** Continuous sends private ID embeddings to the backbone; IDShare uses private routing states to select shared outputs. IDs with the same code jointly update one embedding. The backbone, other features, and objective are unchanged.

three-seed mean AUC than Continuous, AdamAR, TailShare, and FrozenHash. Its AUC also exceeds that of Continuous with tuned global L2, without using that penalty itself. Experiments with additional backbone architectures reproduce this advantage.

Further analysis on TAAC examines capacity, depth, and long-tailed IDs. IDShare remains stable as the number of shared embeddings varies from 10,000 to 80,000. Its advantage also persists as the backbone grows deeper. At two, four, and eight layers, IDShare outperforms Continuous on all three paired seeds. Frequency-group analysis further shows improved predictions for unseen IDs and long-tailed IDs with 1–10 training lookups.

Taken together, the results show that learned sharing improves predictive accuracy while providing regularization through the representation itself. IDShare needs no additional regularization coefficient search or frequency threshold, making it a simple, stable default for long-tailed ID embeddings.

2 IDSHARE

2.1 CONTINUOUS IDENTITY VECTORS

Let $E \in \mathbb{R}^{N \times d}$ be a continuous identity table and let e_i be the row for identity i. The predictor consumes

$$z_i = e_i. \tag{1}$$

To regularize these independent embeddings, we consider global L2 applied to the entire identity table:

$$\mathcal{L}_{\mathrm{L2}}(E) = \lambda \sum_{i=1}^{N} \|e_i\|_2^2. \tag{2}$$

The penalty contributes a gradient of $2\lambda e_i$ to each real-ID row. Increasing λ directly changes the vectors seen by the predictor.

2.2 SHARED DISCRETE IDENTITY CODES

IDShare separates a private routing state from the emitted representation. Let $R \in \mathbb{R}^{N \times d}$ contain routing states r_i, and let $C = \{c_0, \ldots, c_{K-1}\}$ be a learned codebook. The hard assignment is

$$q(i) = \arg \min_{k \in \{0,\ldots,K-1\}} \|r_i - c_k\|_2^2. \tag{3}$$

The predictor consumes a shared output:

$$z_i = \begin{cases} b_0, & q(i) = 0, \\ b_+ + c_{q(i)}, & q(i) > 0. \end{cases} \tag{4}$$

Here $c_0 = 0$ is fixed, b_0 is a learned shared zero-route state, and b_+ is a learned base for nonzero codes. The shared offset b_+ adjusts all nonzero-code outputs without entering the routing distance. With one code per ID and no continuous residual, a private row affects predictions only through its selected code.

The zero route also handles IDs that arrive after training. For an unseen ID, $r_i = 0$ selects c_0 and emits b_0. Observed IDs routed to c_0 update this same b_0 during training. Unseen IDs thus reuse a learned shared representation without a stored per-ID code or a frequency estimate.

Training jointly learns the shared embeddings and which IDs use them, without requiring item content features. Each forward lookup recomputes the assignment by nearest-code search using the current routing state and codebook. Sharing groups can therefore evolve with the prediction task. The hard forward pass uses the selected shared embedding. A soft straight-through backward pass sends task gradients to both R and C (Bengio et al., 2013; van den Oord et al., 2017; Chen et al., 2020). Routing states start at zero. At the first training lookup, a deterministic, label-free hash initializes each observed ID at a nonzero code to break symmetry. At validation, trained IDs use their learned routing states.

2.3 Why sharing is regularization

Two properties define the regularization imposed by shared outputs. First, fix the codebook and all non-identity inputs. The set of identity vectors visible to the predictor satisfies

$$|\{z_i : i \in \{1, \dots, N\}\}| \le K, \tag{5}$$

whereas Continuous can expose up to N distinct vectors. IDs assigned to the same code receive the same identity embedding.

Second, sharing pools the task signal. A nonzero code k emits $u_k = b_+ + c_k$, so $z_i = u_k$ whenever $q(i) = k$. Let $\mathcal{O}_\mathcal{B}$ index target and history lookup occurrences in minibatch $\mathcal{B}$, with output $z^{(t)} = z_{i_t}$ at occurrence t. The minibatch task gradient along the hard forward path is

$$\frac{\partial \mathcal{L}_\mathcal{B}^{\text{hard}}}{\partial u_k} = \sum_{t \in \mathcal{O}_\mathcal{B} : q(i_t) = k} \frac{\partial \mathcal{L}_\mathcal{B}^{\text{hard}}}{\partial z^{(t)}}. \tag{6}$$

Examples routed to one code learn one emitted vector together, so long-tailed IDs draw on updates from other IDs. The soft straight-through path adds task gradients, while all lookups with $q(i_t) = k$ still consume the same u_k. This is the same broad reason that parameter sharing acts as an inductive bias in multitask learning (Caruana, 1997), but here the sharing occurs inside one high-cardinality identity field.

Sharing limits independent identity representations and pools training signals across IDs. Fixed hash and compositional embeddings also use shared pools (Weinberger et al., 2009; Svenstrup et al., 2017; Shi et al., 2020). IDShare learns the assignments jointly with the recommendation objective.

2.4 Sharing and L2 impose different constraints

Global L2 pulls every ID embedding toward zero at each training step. Long-tailed IDs receive fewer task updates, so they undergo more shrinkage steps per task update.

IDShare instead requires IDs assigned to the same code to use the same embedding. For a fixed assignment q, its output table $Z = [z_1; \dots; z_N]$ is restricted to

$$\mathcal{S}_q = \{Z \in \mathbb{R}^{N \times d} : z_i = z_j \text{ whenever } q(i) = q(j)\}. \tag{7}$$

This output table has at most Kd free scalar values, while the routing table retains $N \times d$ trainable parameters. Updating a shared embedding changes the representation of every ID assigned to it.

The assignments are learned, so the prediction task determines which IDs share each embedding.

2.5 TRAINING OBJECTIVE AND THE NO-L2 DEFAULT

The recommendation loss $\mathcal{L}_{\text{rec}}$ is the mean binary cross-entropy over the minibatch. The selected nonzero code is fitted to the routing state with

$$\mathcal{L}_{\text{code}} = \frac{1}{d\,|\mathcal{I}_{\mathcal{B}}|} \sum_{t \in \mathcal{I}_{\mathcal{B}}} \left\| c_{q(i_t)} - \text{sg}(r_{i_t}) \right\|_2^2, \quad \mathcal{I}_{\mathcal{B}} = \{t \in \mathcal{O}_{\mathcal{B}} : q(i_t) > 0\}, \tag{8}$$

where sg stops the gradient. The mean is over lookup occurrences and embedding dimensions; the loss is zero if no lookup takes a nonzero route. The reported objective is

$$\mathcal{L} = \mathcal{L}_{\text{rec}} + \mathcal{L}_{\text{code}} + \lambda \|R_{\text{real}}\|_F^2. \tag{9}$$

The codebook-loss weight is fixed to 1 in every reported experiment and is not selected per setting. The commitment term is disabled. Thus task gradients learn both R and C, the fitting loss updates C, and no extra term pulls routing states toward codes. Our default sets $\lambda = 0$; the matched grid measures whether adding global L2 to R improves validation performance. We use the same code parameterization, temperature schedule, and initialization scheme across all four backbone settings; Appendix A provides the implementation details.

3 EXPERIMENTAL DESIGN

3.1 DATASETS AND BACKBONES

We choose TAAC and KuaiRand to study learned sharing in advertising and short-video recommendation. Both contain tens of millions of item IDs. Our TAAC setting includes item histories, whereas our KuaiRand setting uses feature fields without history sequences. This lets us test learned sharing across recommendation domains and input structures.

TAAC. We use a temporal split of the Tencent Advertising Algorithm Competition 2025 log (Tencent Advertising Algorithm Competition, 2025): 131,222,882 training examples and 21,951,746 validation examples. Target and history items share a 17,487,680-row table, including four reserved rows. History length is capped at 100.

KuaiRand. We use the public KuaiRand-27K log (Gao et al., 2022) with the binary `is_click` target: 243,480,365 training examples and 12,080,526 validation examples. The `video_id` field contains 25,020,522 real rows.

Backbones. We use RankMixer backbones (Zhu et al., 2025) adapted to each dataset's inputs. Both use parameter-free multi-head token mixing and per-token feed-forward networks. Both have two layers, with width 64 on TAAC and 74 on KuaiRand. The identity embedding dimension is 16. We also evaluate DIN (Zhou et al., 2018) on TAAC and TokenMixer-Large (Jiang et al., 2026) on KuaiRand, retaining each dataset's splits, K, and training protocol. Table 1 summarizes the main settings; N/K denotes the capacity ratio.

Table 1: **Experimental settings.** The architecture and codebook size are fixed across regularization comparisons.

Setting	Inputs	Real IDs N	K	N/K	d
TAAC	Seq. + fields	17.49M	20K	874	16
KuaiRand	Fields	25.02M	40K	626	16

Matched settings and evaluation. Within each setting, all methods use the same backbone, data order, non-identity features, clipping, seeds, and checkpoint selection policy. TAAC uses learning rate 10^{-3} and batch size 2,048; KuaiRand uses 2×10^{-3} and batch size 4,096. Backbone configurations, training schedules, and seed lists are in Appendix A. We report validation AUC, with logloss added for the frequency diagnostic. Unless otherwise stated, reported metrics are averaged equally across seeds. We express AUC as a percentage and absolute AUC differences in percentage points (pp). Differences are computed before rounding.

3.2 REGULARIZATION BASELINES

The main comparison uses five methods, all without global L2. **Continuous** learns one independent embedding per ID. **AdamAR** applies adaptive row-wise shrinkage to that table with a tuned coefficient (Li & Lyu, 2026). **TailShare** maps IDs seen one to five times in training to one shared trainable embedding and keeps frequent IDs independent. **FrozenHash** fixes each real ID to a hash-assigned code, using the same K as IDShare. The shared embeddings remain trainable. **IDShare** learns assignments to K shared codes without a frequency threshold.

All methods use the same Adam hyperparameters; only AdamAR changes the update rule for real identity rows. The main comparison uses three seeds. Appendix B gives coefficient selection and implementation details.

3.3 GLOBAL L2 COMPARISON

We separately examine whether global L2 improves Continuous or IDShare. The backbone and IDShare codebook size remain fixed, and both representations use the same coefficient grid,

$$\Lambda = \{0, 10^{-11}, 10^{-10}, 3\times10^{-10}, 10^{-9}, 3\times10^{-9}, 10^{-8}\}. \tag{10}$$

The penalty applies to the full real-ID table: the emitted embeddings E for Continuous and the private routing states R for IDShare. Padding and OOV rows are excluded. The shared codebook C is trained by Equation 9 and receives no global L2 penalty.

Optimizer weight decay is disabled. We globally clip non-L2 gradients, including the codebook-loss gradients for IDShare, then add L2 gradients before the Adam update (Kingma & Ba, 2015). This is coupled L2, which differs from decoupled weight decay under Adam (Loshchilov & Hutter, 2019; Zhang et al., 2019).

For representation c, coefficient λ, and seed s, let $A_c(\lambda, s)$ denote validation AUC in percent and $\bar{A}_c(\lambda)$ its seed mean. To summarize the gain from the L2 sweep, we compare its highest seed-mean AUC with the value at zero L2:

$$V_{\mathrm{L2}}(c) = \max_{\lambda\in\Lambda} \bar{A}_c(\lambda) - \bar{A}_c(0). \tag{11}$$

In each setting, "tuned Continuous" uses the coefficient with the highest Continuous seed-mean AUC in this grid. To evaluate transfer, we apply the coefficient selected for Continuous in the other setting without retuning.

4 RESULTS

4.1 IDSHARE LEADS THE REGULARIZATION COMPARISON

IDShare achieves the highest mean AUC among the five methods on both datasets (Table 2). AdamAR improves over independent Continuous embeddings on TAAC, but not on KuaiRand. IDShare improves on Continuous in both settings without tuning an additional regularization coefficient or choosing a frequency threshold.

Table 2: **Regularization through updates or sharing.** Validation AUC (%), mean ± sample standard deviation over three seeds. All five methods use zero global L2. Bold marks the highest mean in each setting.

Method	TAAC	KuaiRand
Continuous	77.070 ±0.028	75.370 ±0.040
AdamAR	77.464 ±0.025	75.309 ±0.011
TailShare	76.965 ±0.095	75.313 ±0.044
FrozenHash	76.940 ±0.082	75.321 ±0.046
IDShare	**77.526** ±0.178	**75.445** ±0.014

The comparison also shows that how IDs share matters. TailShare and FrozenHash both fall below Continuous in AUC on both datasets. FrozenHash uses the same codebook size as IDShare. IDShare

exceeds its AUC by 0.586 pp on TAAC and 0.123 pp on KuaiRand, and outperforms it on all three paired seeds in each setting. Fixed sharing rules therefore do not reproduce the advantage of task-learned sharing. Appendix B specifies the assignment and backward paths.

4.2 IDShare without L2 outperforms tuned Continuous

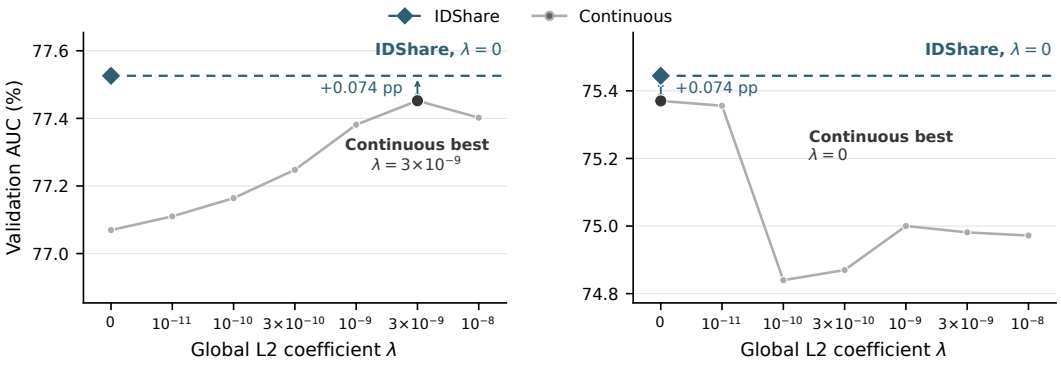


Figure 2: **No-L2 IDShare compared with tuned Continuous.** Gray points show Continuous; the dark point marks its best mean. The blue diamond and horizontal reference mark IDShare (0); arrows show its gain (pp). Both AUC axes span 0.8 pp.

Global L2 affects Continuous differently across the two settings (Figure 2). A suitable nonzero coefficient improves AUC on TAAC, whereas every evaluated nonzero coefficient lowers it on KuaiRand. Continuous therefore requires setting-specific coefficient selection: 3×10^{-9} gives its highest mean on TAAC, while zero is best on KuaiRand. Even after selecting the highest-performing coefficient for Continuous in each setting, IDShare achieves higher mean AUC without global L2. Appendix C gives the complete grids and sample standard deviations.

Applying TAAC's 3×10^{-9} to KuaiRand loses 0.389 pp relative to tuned Continuous; using KuaiRand's 0 on TAAC forgoes the 0.383 pp gain from TAAC's L2 sweep. In contrast, IDShare uses the same zero-L2 rule in both settings (Appendix C.1).

4.3 The complete L2 response supports the default

IDShare performs close to its highest measured mean AUC without global L2 (Figure 3). The full L2 sweep adds 0.079 pp on TAAC and 0.019 pp on KuaiRand relative to zero L2. Its response is nearly flat on KuaiRand; on TAAC, small coefficients improve AUC, while larger coefficients reduce it. Neither gain is needed to outperform tuned Continuous: IDShare already does so at zero L2 in both settings. This supports zero L2 as a simple default, without an additional coefficient search.

4.4 The no-L2 result holds across backbones

We next test whether the no-L2 advantage holds across backbone architectures. On both DIN and TokenMixer-Large, IDShare retains higher mean AUC than tuned Continuous (Table 3). Each row compares the two representations within the same backbone.

Table 3: **The no-L2 result across backbones.** AUC (%): C_0 is Continuous at zero L2; C_* is its highest seven-point-grid seed mean. $\Delta = \bar{A}_{\text{IDShare}}(0) - C_*$ (pp).

Setting	Backbone	Seeds	C_0	C_*	IDShare (0)	Δ
TAAC	RankMixer	3	77.070	77.452	**77.526**	+0.074
TAAC	DIN	3	75.374	75.857	**76.298**	+0.441
KuaiRand	RankMixer	3	75.370	75.370	**75.445**	+0.074
KuaiRand	TokenMixer-Large	3	75.378	75.403	**75.421**	+0.018

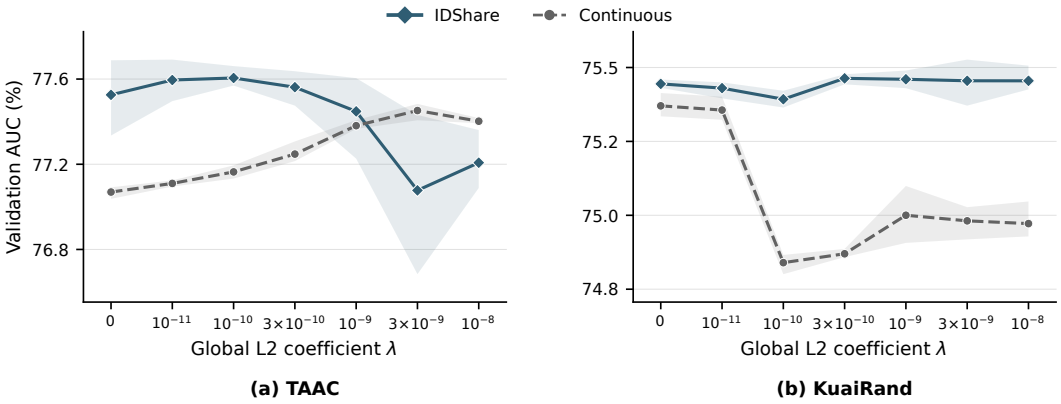


Figure 3: **Complete L2 response.** Lines connect seed means at measured coefficients; shading spans the three-seed min–max range. L2 applies to E for Continuous and R for IDShare. Each panel uses its own AUC scale.

Appendix C.2 gives the complete grids and sample standard deviations.

4.5 Robustness to codebook size and depth

We test robustness to codebook size and backbone depth on TAAC, varying one factor at a time and keeping other settings fixed (Table 4).

Table 4: **Robustness to codebook size and depth on TAAC.** Three-seed mean AUC (%) at zero L2; Δ is IDShare minus Continuous (pp).

Configuration	Continuous	**IDShare**	Δ (pp)
Codebook size K (two layers)			
10K	77.070	**77.566**	+0.497
20K	77.070	**77.526**	+0.456
40K	77.070	**77.573**	+0.503
80K	77.070	**77.483**	+0.414
Backbone depth ($K = 20K$)			
2 layers	77.070	**77.526**	+0.456
4 layers	77.125	**77.527**	+0.402
8 layers	77.118	**77.530**	+0.412

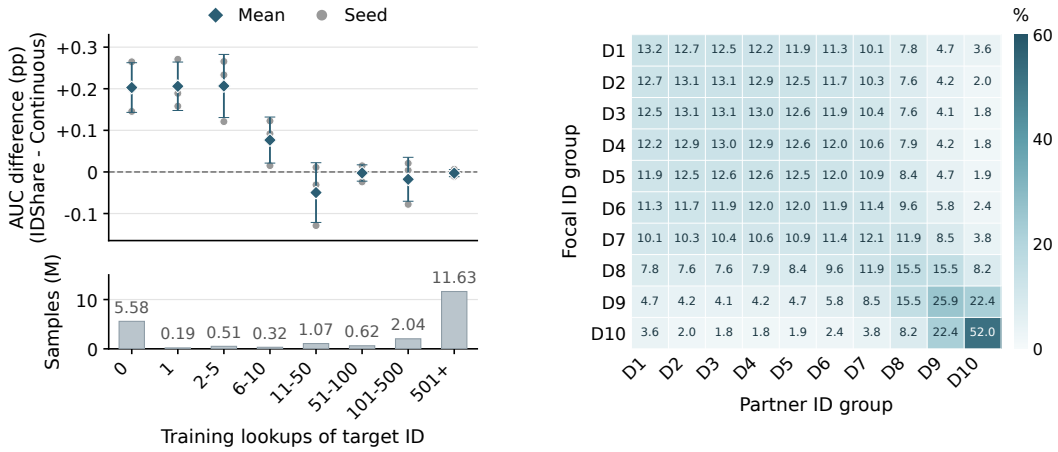
Across $K = 10$K–80K, the IDShare mean AUC spans only 0.090 pp, while its advantage over Continuous ranges from 0.414 to 0.503 pp. At every measured K, IDShare outperforms Continuous on all three paired seeds, supporting a broad capacity range rather than a narrow optimum.

The advantage persists as the backbone becomes deeper. At two, four, and eight layers, IDShare leads Continuous on all three paired seeds at every depth, with a mean AUC advantage of 0.402–0.456 pp. Appendix D gives the full means, standard deviations, and paired gaps.

4.6 Better predictions for unseen and long-tailed IDs

To identify which frequency groups benefit, we repeat the main TAAC comparison on three seeds and retain predictions (Figure 4(a)). Validation examples are grouped by target ID training lookups, including target and truncated-history occurrences, from unseen to frequent IDs.

For unseen IDs and each of the 1, 2–5, and 6–10 lookup buckets, IDShare improves both AUC and logloss on all three paired seeds. Mean AUC gains are 0.203, 0.206, 0.207, and 0.077 pp, respectively. The higher-frequency buckets have small negative mean AUC differences; unseen and long-tailed IDs benefit consistently.



(a) Frequency-group performance

(b) Learned sharing structure

Figure 4: **Prediction gains and learned sharing on TAAC.** (a) AUC gains (three-seed means, sample SD, and individual seeds) and sample counts; 0 denotes unseen IDs. (b) Same-code partner composition in equal-ID frequency groups (D1 lowest, D10 highest), averaged over IDs then seeds. Rows sum to 100% before rounding; the population reference is 10%.

These independent reruns reproduce the overall advantage: AUC is 77.609% for IDShare and 77.074% for Continuous, a 0.535 pp gap, positive on all three seeds. Bucket AUC measures within-group ranking, not a decomposition of overall gains. Appendix E gives complete results.

4.7 Learned sharing has frequency structure

We examine the final shared groups in the same three TAAC IDShare checkpoints. Training-seen IDs are split into ten equal-sized groups by lookup count, from D1 (lowest) to D10 (highest). Figure 4(b) shows their same-code partner composition.

Lower-frequency IDs share relatively little with the highest-frequency group. For D1, D10 accounts for only 3.6% of partners, below its 10% population share. By contrast, 52.0% of D10's partners are also in D10. This structure emerges without using ID frequencies or thresholds in the routing rule.

This pattern shows how the regularization constraint in Section 2.3 is organized: long-tailed IDs are tied mainly to other lower-frequency IDs. These IDs jointly learn one output embedding rather than fit independent vectors.

5 IDShare: A No-L2 Training Policy

IDShare's learned sharing improves prediction while regularizing long-tailed ID embeddings. This supports a simple default: choose K within systems constraints (10K–80K is robust on TAAC) and train at zero identity-table L2 with the normal checkpoint policy. Unlike AdamAR and TailShare, this needs no extra shrinkage coefficient, frequency threshold, or stored per-ID counts. Validate this configuration directly rather than run a seven-point L2 search.

6 Related Work

Regularizing high-cardinality embeddings. Uneven ID frequencies motivate methods that adapt how embeddings are trained. Frequency-aware SGD, λOpt, CowClip, and AdamAR modify learning rates, regularization, or norm control (Li et al., 2022; Chen et al., 2019; Zheng et al., 2023; Li & Lyu, 2026). A complementary approach changes the embedding used by a training example. Stochastic Shared Embeddings randomly replaces embedding lookups while retaining the original

table (Wu et al., 2019). One-epoch studies also show that filtering or hashing long-tailed IDs can reduce embedding overfitting (Zhang et al., 2022). These studies connect embedding generalization to both update rules and parameter sharing.

Shared and quantized embeddings. Sharing also offers a way to control representation capacity and memory use. Hashing and complementary partitions map IDs into shared parameter pools (Weinberger et al., 2009; Svenstrup et al., 2017; Shi et al., 2020; Desai et al., 2022). Learned compositional codes and differentiable product quantization instead optimize compact discrete representations (Shu & Nakayama, 2018; Chen et al., 2018; 2020). Within this family, MGQE gives long-tailed IDs fewer available codes to limit overfitting (Kang et al., 2020). Other capacity-efficient designs adapt embedding dimensions or generate embeddings without a lookup table (Ginart et al., 2021; Lyu et al., 2022; Kang et al., 2021). In recommendation, sharing can extend across feature fields: Unified Embedding multiplexes categorical features into one representation space (Coleman et al., 2023). Another direction learns which IDs share. Clustered Embedding Learning alternates embedding optimization with cluster splitting and ID reassignment. IDs in each cluster jointly train a shared embedding, improving long-tailed ID representations while reducing memory use (Chen et al., 2023). Clustering the Sketch likewise interleaves clustering with training and reports reduced overfitting (Tsang & Ahle, 2023), while AutoAssign+ adapts shared embedding assignments to streaming recommendation (Liu et al., 2024b).

End-to-end discrete identities. A related question is how discrete item codes are learned for the downstream task. Product quantization and VQ-VAE provide the underlying codebook machinery (Jégou et al., 2011; van den Oord et al., 2017). Recommendation methods use item codes for transfer, ranking, or generative retrieval (Hou et al., 2023; Rajput et al., 2023; Singh et al., 2024). Joint training aligns code learning with the recommendation objective: CAGE learns hierarchical category codes, while ETEGRec and DIGER couple item tokenization with recommendation training (Liu et al., 2024a; 2025; Fu et al., 2026).

Building on these approaches to embedding regularization and shared representations, IDShare uses a single learned code as the complete identity representation in a discriminative recommender. We study learned sharing as a regularizer for long-tailed ID embeddings and as a basis for training without an additional global L2 search.

7 Conclusion

We show that learned sharing provides effective regularization for long-tailed ID embeddings. IDShare learns which IDs share an embedding. This limits independent identity representations and allows their training examples to update a shared vector together. On both recommendation benchmarks, IDShare achieves higher mean AUC than Continuous, AdamAR, TailShare, and FrozenHash. It also exceeds tuned Continuous without using global L2. This advantage persists across backbone architectures. Further analysis on TAAC shows stable performance across codebook sizes, an advantage in deeper models, and improved predictions for long-tailed IDs. These findings support a simple training default: choose a practical codebook size and train IDShare without an additional regularization coefficient search or frequency threshold.

Reproducibility Statement

Records include splits, configurations, package hashes, job IDs, and frozen readouts checked before aggregation. Appendices A–F provide complete grids, sample standard deviations, baseline selection, and diagnostic protocols. No best K or depth is selected.

AI Use Statement

OpenAI Codex and Cursor assisted with organization, language, record inspection, and plotting code. The authors defined the study and verified all experiments, results, figures, claims, and references.

REFERENCES

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. arXiv:1308.3432.

Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997. doi: 10.1023/A:1007379606734.

Ting Chen, Martin Renqiang Min, and Yizhou Sun. Learning k-way d-dimensional discrete codes for compact embedding representations. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 854–863, 2018. URL https://proceedings.mlr.press/v80/chen18g.html.

Ting Chen, Lala Li, and Yizhou Sun. Differentiable product quantization for end-to-end embedding compression. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1617–1626, 2020. URL https://proceedings.mlr.press/v119/chen20l.html.

Yihong Chen, Bei Chen, Xiangnan He, Chen Gao, Yong Li, Jian-Guang Lou, and Yue Wang. λOpt: Learn to regularize recommender models in finer levels. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 978–986, 2019. doi: 10.1145/3292500.3330880.

Yizhou Chen, Guangda Huzhang, Anxiang Zeng, Qingtao Yu, Hui Sun, Heng-Yi Li, Jingyi Li, Yabo Ni, Han Yu, and Zhiming Zhou. Clustered embedding learning for recommender systems. In *Proceedings of the ACM Web Conference 2023*, pp. 1074–1084, 2023. doi: 10.1145/3543507.3583362.

Benjamin Coleman, Wang-Cheng Kang, Matthew Fahrbach, Ruoxi Wang, Lichan Hong, Ed H. Chi, and Derek Zhiyuan Cheng. Unified embedding: Battle-tested feature representations for web-scale ML systems. In *Advances in Neural Information Processing Systems*, volume 36, pp. 56234–56255, 2023. doi: 10.52202/075280-2453.

Aditya Desai, Li Chou, and Anshumali Shrivastava. Random offset block embedding (ROBE) for compressed embedding tables in deep learning recommendation systems. In *Proceedings of Machine Learning and Systems*, volume 4, pp. 762–778, 2022.

Junchen Fu, Xuri Ge, Alexandros Karatzoglou, Ioannis Arapakis, Suzan Verberne, Joemon M. Jose, and Zhaochun Ren. Differentiable semantic ID for generative recommendation. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 369–379, 2026. doi: 10.1145/3805712.3809641.

Chongming Gao, Shijun Li, Yuan Zhang, Jiawei Chen, Biao Li, Wenqiang Lei, Peng Jiang, and Xiangnan He. KuaiRand: An unbiased sequential recommendation dataset with randomly exposed videos. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management*, pp. 3953–3957, 2022. doi: 10.1145/3511808.3557624.

Antonio A. Ginart, Maxim Naumov, Dheevatsa Mudigere, Jiyan Yang, and James Zou. Mixed dimension embeddings with application to memory-efficient recommendation systems. In *2021 IEEE International Symposium on Information Theory*, pp. 2786–2791, 2021. doi: 10.1109/ISIT45174.2021.9517710.

Yupeng Hou, Zhankui He, Julian McAuley, and Wayne Xin Zhao. Learning vector-quantized item representation for transferable sequential recommenders. In *Proceedings of the ACM Web Conference 2023*, pp. 1162–1171, 2023. doi: 10.1145/3543507.3583434.

Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011. doi: 10.1109/TPAMI.2010.57.

Yuchen Jiang, Jie Zhu, Xintian Han, Hui Lu, Kunmin Bai, Mingyu Yang, Shikang Wu, Ruihao Zhang, Wenlin Zhao, Shipeng Bai, Sijin Zhou, Huizhi Yang, Tianyi Liu, Wenda Liu, Ziyan Gong, Haoran Ding, Zheng Chai, Deping Xie, Zhe Chen, Yuchao Zheng, and Peng Xu.

TokenMixer-Large: Scaling up large ranking models in industrial recommenders. *arXiv preprint arXiv:2602.06563*, 2026. doi: 10.48550/arXiv.2602.06563. URL https://arxiv.org/abs/2602.06563.

Wang-Cheng Kang, Derek Zhiyuan Cheng, Ting Chen, Xinyang Yi, Dong Lin, Lichan Hong, and Ed H. Chi. Learning multi-granular quantized embeddings for large-vocab categorical features in recommender systems. In *Companion Proceedings of the Web Conference 2020*, pp. 562–566, 2020. doi: 10.1145/3366424.3383416.

Wang-Cheng Kang, Derek Zhiyuan Cheng, Tiansheng Yao, Xinyang Yi, Ting Chen, Lichan Hong, and Ed H. Chi. Learning to embed categorical features without embedding tables for recommendation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 840–850, 2021. doi: 10.1145/3447548.3467304.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Mang Li and Wei Lyu. Adaptive regularization for large-scale sparse feature embedding models. In *International Conference on Learning Representations*, 2026.

Yan Li, Dhruv Choudhary, Xiaohan Wei, Baichuan Yuan, Bhargav Bhushanam, Tuo Zhao, and Guanghui Lan. Frequency-aware SGD for efficient embedding learning with provable benefits. In *International Conference on Learning Representations*, 2022.

Enze Liu, Bowen Zheng, Cheng Ling, Lantao Hu, Han Li, and Wayne Xin Zhao. Generative recommender with end-to-end learnable item tokenization. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 729–739, 2025. doi: 10.1145/3726302.3729989.

Qijiong Liu, Jiaren Xiao, Lu Fan, Jieming Zhu, and Xiao-Ming Wu. Learning category trees for ID-based recommendation: Exploring the power of differentiable vector quantization. In *Proceedings of the ACM Web Conference 2024*, pp. 3521–3532, 2024a. doi: 10.1145/3589334.3645484.

Ziru Liu, Kecheng Chen, Fengyi Song, Bo Chen, Xiangyu Zhao, Huifeng Guo, and Ruiming Tang. AutoAssign+: Automatic shared embedding assignment in streaming recommendation. *Knowledge and Information Systems*, 66(1):89–113, 2024b. doi: 10.1007/s10115-023-01951-1.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Fuyuan Lyu, Xing Tang, Hong Zhu, Huifeng Guo, Yingxue Zhang, Ruiming Tang, and Xue Liu. OptEmbed: Learning optimal embedding table for click-through rate prediction. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management*, pp. 1399–1409, 2022. doi: 10.1145/3511808.3557411.

Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. Deep learning recommendation model for personalization and recommendation systems, 2019. arXiv:1906.00091.

Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan Hulikal Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Q. Tran, Jonah Samost, Maciej Kula, Ed H. Chi, and Maheswaran Sathiamoorthy. Recommender systems with generative retrieval. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-0452.

Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. 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*, pp. 165–175, 2020. doi: 10.1145/3394486.3403059.

Raphael Shu and Hideki Nakayama. Compressing word embeddings via deep compositional code learning. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BJRZzFlRb.

Anima Singh, Trung Vu, Nikhil Mehta, Raghunandan Keshavan, Maheswaran Sathiamoorthy, Yilin Zheng, Lichan Hong, Lukasz Heldt, Li Wei, Devansh Tandon, Ed H. Chi, and Xinyang Yi. Better generalization with semantic ids: A case study in ranking for recommendations. In *Proceedings of the 18th ACM Conference on Recommender Systems*, pp. 1039–1044, 2024. doi: 10.1145/3640457.3688190.

Dan Tito Svenstrup, Jonas Hansen, and Ole Winther. Hash embeddings for efficient word representations. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Tencent Advertising Algorithm Competition. 2025 Tencent Ads Algorithm Competition baseline. https://github.com/TencentAdvertisingAlgorithmCompetition/baseline_2025, 2025.

Henry Ling-Hei Tsang and Thomas Dybdahl Ahle. Clustering the sketch: Dynamic compression for embedding tables. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-3159.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 1113–1120, 2009. doi: 10.1145/1553374.1553516.

Liwei Wu, Shuqing Li, Cho-Jui Hsieh, and James L. Sharpnack. Stochastic shared embeddings: Data-driven regularization of embedding layers. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Guodong Zhang, Chaoqi Wang, Bowen Xu, and Roger Grosse. Three mechanisms of weight decay regularization. In *International Conference on Learning Representations*, 2019.

Zhao-Yu Zhang, Xiang-Rong Sheng, Yujing Zhang, Biye Jiang, Shuguang Han, Hongbo Deng, and Bo Zheng. Towards understanding the overfitting phenomenon of deep click-through rate prediction models. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management*, pp. 2671–2680, 2022. doi: 10.1145/3511808.3557479.

Zangwei Zheng, Pengtai Xu, Xuan Zou, Da Tang, Zhen Li, Chenguang Xi, Peng Wu, Leqi Zou, Yijie Zhu, Ming Chen, Xiangzhuo Ding, Fuzhao Xue, Ziheng Qin, Youlong Cheng, and Yang You. CowClip: Reducing CTR prediction model training time from 12 hours to 10 minutes on 1 GPU. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(9):11390–11398, 2023. doi: 10.1609/aaai.v37i9.26347.

Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1059–1068, 2018. doi: 10.1145/3219819.3219823.

Jie Zhu, Zhifang Fan, Xiaoxie Zhu, Yuchen Jiang, Hangyu Wang, Xintian Han, Haoran Ding, Xinmin Wang, Wenlin Zhao, Zhen Gong, Huizhi Yang, Zheng Chai, Zhe Chen, Yuchao Zheng, Qiwei Chen, Feng Zhang, Xun Zhou, Peng Xu, Xiao Yang, Di Wu, and Zuotao Liu. RankMixer: Scaling up ranking models in industrial recommenders. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pp. 6309–6316, 2025. doi: 10.1145/3746252.3761507. URL https://doi.org/10.1145/3746252.3761507.

A Backbone and Implementation Details

The RankMixer backbones use parameter-free token/head mixing followed by per-token feed-forward networks. Their dataset-specific input adapters, widths, activations, and pooling are specified below.

Data splits and labels. TAAC uses May 23–28, 2025 for training and May 29 for validation, with UTC+8 day boundaries. Actions 1 and 2 are positive; other nonmissing actions are negative. Each target must have a preceding event, and its history contains only earlier events, truncated to 100 items. KuaiRand-27K uses standard-exposure logs from April 7–May 1, 2022 for training and May 2 for validation, with `is_click` as the label. Random-exposure logs are excluded. Its identity vocabulary and feature transforms are fitted on training data.

TAAC: sequence and feature tokens. The TAAC implementation uses pre-normalized residual mixing blocks with per-token SwiGLU feed-forward networks, width 64, expansion factor 2, and two layers. The tokenizer produces 11 non-sequence and five sequence tokens. Two sequence tokens summarize the most recent items; three summarize equal history chunks with attention pooling. The final representation averages the non-sequence tokens and uses an MLP head of widths 128 and 64. Dropout is zero. Target and history items use the same identity table and codebook. Training uses one epoch with seeds 2021, 42, and 2024.

KuaiRand: feature tokens. The KuaiRand implementation uses the dense RankMixer mixing principle (Zhu et al., 2025), not its sparse mixture-of-experts extension. It projects 37 feature fields into 37 tokens of width 74. Each of two blocks uses pre-normalization, parameter-free token mixing, and a separate GELU feed-forward network per token with hidden width 256. The mixer views its input as 37 tokens with 37 heads of dimension 2 and swaps the token and head axes. The final mean-pooled representation feeds an MLP head of widths 512 and 256. Dropout is zero; no extra cross-layer carry path is added. Training runs for at most ten epochs, with validation-based checkpoint selection and seeds 42, 2024, and 933888.

DIN and TokenMixer-Large. The additional TAAC backbone is DIN with attention dimension 32 and MLP widths 128 and 64. The additional KuaiRand backbone is a two-layer dense implementation of TokenMixer-Large (Jiang et al., 2026), using its mixing-and-reverting blocks with per-token SwiGLU networks. It uses 37 tokens, width 74, and feed-forward width 256. Their seed sets are $\{2021, 42, 2024\}$ and $\{42, 2024, 933888\}$, respectively. They retain the main settings' dataset-specific optimizers, batch sizes, and training schedules, with identity dimension 16.

Optimizer settings. Both main settings use Adam with $(\beta_1, \beta_2) = (0.9, 0.999)$ and $\epsilon = 10^{-8}$. Non-L2 gradients are clipped to a global norm of 1 before any global L2 gradient is added.

Validation and checkpoint selection. Both settings evaluate at each epoch end. TAAC uses the checkpoint after its single training epoch. KuaiRand retains the checkpoint with the highest validation AUC, using a 10^{-12} comparison tolerance on the raw AUC scale. Training stops after at least four epochs once three consecutive evaluations fail to exceed the patience reference by 10^{-4} AUC (0.01 pp), or at ten epochs. The patience reference is updated only when that threshold is crossed; checkpoint selection is independent of this stopping threshold.

Quantizer and loss. All four settings use one codebook, one code per item, codebook fitting weight 1, commitment weight 0, and no private continuous residual. The TAAC RankMixer implementation deduplicates item lookups for computation, then restores their occurrence weights when averaging the codebook loss. It is therefore the occurrence-and-dimension mean in Equation 8, not a sum over dimensions or an unweighted mean over unique IDs. All L2 curves use the full real identity table, not only rows touched by the current minibatch.

Straight-through updates. For an identity with routing state r_i, define

$$p_{ik} = \frac{\exp(-\|r_i - c_k\|_2^2/\tau)}{\sum_{j=0}^{K-1} \exp(-\|r_i - c_j\|_2^2/\tau)}, \qquad s_i = \sum_{k=0}^{K-1} p_{ik} c_k.$$

The training output is implemented as

$$\widetilde{z}_i = b(q(i)) + c_{q(i)} + s_i - \mathrm{sg}(s_i), \qquad b(0) = b_0, \quad b(k) = b_+ \text{ for } k > 0.$$

The forward value equals Equation 4. Backpropagation retains the direct gradient to the selected code as well as the soft path through s_i to both routing states and codes. Temperature decreases

linearly from 1.0 to 0.3 over the first training epoch and stays at 0.3 thereafter. Nonzero codes have learned unit directions and radii $0.1 + \mathrm{softplus}(\rho_k)$, initialized at radius 0.2; the shared bases start at zero. These settings are fixed across all four backbone settings.

Serving with a fixed model. Training retains the routing table and performs nearest-code search. After training, IDShare can export the ID-to-code map $q(i)$ and a table of the K shared output vectors in Equation 4, with the shared bases folded in. Serving then uses two table lookups, without the routing table or nearest-code search; unseen IDs retain the shared zero route. The serving representation replaces Nd floating-point values with Kd values and N integer indices, reducing identity storage when $K \ll N$ while preserving the fixed model's outputs. This serving path has not been benchmarked for memory or latency.

Evidence records. Reported aggregates pass structural checks before metric readout. Frozen reports, source hashes, and the run ledger are retained with the experiment records. Earlier optional-L2 routing diagnostics are archived separately; they do not supply any of the main comparisons below.

B Regularization Baselines and Fixed Sharing

All methods reuse the main RankMixer data splits and input processing in Section 3, and the training schedules and three-seed sets in Appendix A. Global L2 and optimizer weight decay are zero. Continuous and IDShare reuse their accepted zero-L2 runs.

AdamAR. We implement the adaptive regularization rule of Li & Lyu (2026) as a row-wise extension of the baseline Adam optimizer. Real identity rows receive shrinkage based on the number of steps since their last nonzero task gradient; non-identity parameters and the four reserved rows keep ordinary Adam updates. The adapter preserves PyTorch Adam's ϵ outside the square root to match the baseline. Moments and shrinkage advance at every training step, including untouched rows; only the last-valid-step timestamp is updated on an active row. This is a dense update, not an on-touch approximation.

The initial coefficient grid was $\alpha \in \{10^{-6.5}, 10^{-4.5}, 10^{-2.5}, 10^{-1}\}$. After these runs selected its lower endpoint on TAAC and upper endpoint on KuaiRand, we fixed a two-point supplement per setting: $\{10^{-8}, 10^{-6}\}$ on TAAC and $\{10^{-0.5}, 10^{-0.05}\}$ on KuaiRand. Seed 42 selects from the combined six points; other seeds do not reselect the coefficient. TAAC selects 10^{-6}, which we also run on seeds 2021 and 2024. KuaiRand retains 10^{-1} and reuses its original three runs. Table 2 averages all three seeds, including seed 42; Table 5 retains both stages of coefficient selection.

Table 5: AdamAR coefficient selection on seed 42. Each setting adds two points after the initial four-point grid. Bold entries select the coefficient used for all three seeds in Table 2; other seeds do not reselect it.

Stage	TAAC α	AUC (%)	KuaiRand α	AUC (%)
Initial	$10^{-6.5}$	77.4545757	$10^{-6.5}$	75.0294431
Initial	$10^{-4.5}$	77.2346304	$10^{-4.5}$	75.2345227
Initial	$10^{-2.5}$	76.9300919	$10^{-2.5}$	75.3001145
Initial	10^{-1}	76.8957742	10^{-1}	**75.3210128**
Supplement	10^{-8}	77.1910092	$10^{-0.5}$	75.2667078
Supplement	10^{-6}	**77.4822273**	$10^{-0.05}$	75.3014724

TailShare. Before training, we count identity lookup occurrences in the training split, without labels. An ID with $1 \le f_i \le 5$ uses a single shared trainable embedding, initialized to zero. IDs with $f_i > 5$ retain independent embeddings; reserved and unseen IDs retain the baseline mapping. The shared vector is learned, not permanently zero. TAAC counts target items and occurrences in the truncated histories actually consumed by the model, with the same table for both. KuaiRand counts the video-ID field. The threshold is fixed at five and is not selected on validation data.

FrozenHash. This ablation retains the IDShare codebook size, shared bases, routing-table allocation, codebook fitting weight 1, and commitment weight 0. A deterministic seed-dependent hash fixes each real ID to a nonzero code; the codebook, shared bases, and backbone remain trainable. Unlike IDShare's learned routing and soft straight-through path, FrozenHash uses assigned-code task gradients and an identity straight-through path to the retained routing state. The comparison is between these fixed- and learned-routing implementations at the same code capacity. The hash maps real rows using

$$\begin{aligned} q_{\text{hash}}(i) &= 1 + \big(h_s(i) \bmod (K-1)\big), \\ h_s(i) &= \big((i-4)\,1103515245 + (s \bmod 2^{31})\big) \bmod 2147483647. \end{aligned} \tag{12}$$

Rows 0–3 keep their reserved behavior. All three methods with shared outputs use exactly the same target/history lookup mapping as their corresponding main-backbone implementation.

C Complete Results and Backbone Checks

C.1 Main-backbone L2 grid and coefficient transfer

Table 6 gives the complete main-backbone grid. Both settings use three seeds; dispersion is the sample standard deviation. The means and observed min–max ranges in Figure 3 use these same seed-level values.

Table 6: Complete validation AUC (%) grid: TAAC and KuaiRand with RankMixer backbones. Values are seed means ± sample standard deviations.

λ	TAAC Continuous	TAAC IDShare	KuaiRand Continuous	KuaiRand IDShare
0	77.0695 ± 0.0283	77.5260 ± 0.1785	75.3703 ± 0.0400	75.4446 ± 0.0142
10^{-11}	77.1100 ± 0.0145	77.5955 ± 0.0976	75.3560 ± 0.0405	75.4302 ± 0.0301
10^{-10}	77.1640 ± 0.0309	77.6051 ± 0.0490	74.8398 ± 0.0344	75.3927 ± 0.0284
3×10^{-10}	77.2478 ± 0.0516	77.5618 ± 0.0814	74.8698 ± 0.0146	75.4634 ± 0.0180
10^{-9}	77.3812 ± 0.0228	77.4482 ± 0.1975	75.0002 ± 0.0964	75.4601 ± 0.0298
3×10^{-9}	77.4522 ± 0.0404	77.0774 ± 0.3754	74.9813 ± 0.0563	75.4551 ± 0.0787
10^{-8}	77.4021 ± 0.0130	77.2073 ± 0.1393	74.9719 ± 0.0651	75.4549 ± 0.0446

The cross-setting comparison reuses this grid: TAAC selects 3×10^{-9} and KuaiRand selects zero by the Continuous seed mean. Transferring the KuaiRand choice to TAAC is therefore the same configuration as zero-L2 Continuous, not an additional run.

C.2 Additional backbones

DIN on TAAC and TokenMixer-Large on KuaiRand use the same three seeds, data splits, and L2 grid as their corresponding main settings. Table 7 reports the mean and sample standard deviation at every grid point. These are architecture checks, not extra seeds for the main comparisons. On DIN, no-L2 IDShare is its highest seed mean; on TokenMixer-Large, the full IDShare sweep adds 0.0578 pp in AUC. In both cases its no-L2 mean exceeds the highest Continuous mean.

Table 7: Complete validation AUC (%) grid: TAAC (DIN) and KuaiRand (TokenMixer-Large). Values are seed means ± sample standard deviations.

λ	TAAC Continuous	TAAC IDShare	KuaiRand Continuous	KuaiRand IDShare
0	75.3736 ± 0.1193	76.2976 ± 0.0283	75.3782 ± 0.1472	75.4213 ± 0.0392
10^{-11}	75.4320 ± 0.1038	76.1573 ± 0.0337	75.4032 ± 0.1432	75.4084 ± 0.0171
10^{-10}	75.5681 ± 0.0281	76.0491 ± 0.0384	74.8404 ± 0.1171	75.4620 ± 0.0342
3×10^{-10}	75.7069 ± 0.0753	76.1719 ± 0.0442	74.9277 ± 0.0300	75.4791 ± 0.0490
10^{-9}	75.8565 ± 0.0960	76.0649 ± 0.0592	74.9747 ± 0.0738	75.4707 ± 0.0406
3×10^{-9}	75.8200 ± 0.1229	75.9107 ± 0.0715	75.0191 ± 0.0346	75.4532 ± 0.0547
10^{-8}	75.7960 ± 0.0817	75.5684 ± 0.0881	75.0006 ± 0.1169	75.4706 ± 0.0463

D TAAC CAPACITY AND DEPTH RESULTS

The studies in Table 4 use the main TAAC backbone, width 64, one epoch, and seeds 2021, 42, and 2024. The capacity study varies $K \in \{10, 20, 40, 80\}$K at two layers. The depth study varies the number of mixing layers in $\{2, 4, 8\}$ at $K = 20$K. Inputs, optimization, and the checkpoint policy remain unchanged; global L2 and weight decay are zero. The two-layer, $K = 20$K results and Continuous reference are reused from the main comparison, without retraining or selecting a best K or depth.

Table 8 gives the means, sample standard deviations, and paired gaps.

Table 8: **TAAC capacity and depth results.** Validation AUC (%) is mean ± sample SD over three seeds. ΔC_0 is the paired mean gap in pp. All runs use zero global L2.

Study	Value	Continuous	IDShare	ΔC_0
Capacity	10K	77.0695 ± 0.0283	77.5663 ± 0.0882	+0.4968
Capacity	20K	77.0695 ± 0.0283	77.5260 ± 0.1785	+0.4565
Capacity	40K	77.0695 ± 0.0283	77.5727 ± 0.1134	+0.5032
Capacity	80K	77.0695 ± 0.0283	77.4831 ± 0.0770	+0.4136
Depth	2	77.0695 ± 0.0283	77.5260 ± 0.1785	+0.4565
Depth	4	77.1250 ± 0.0854	77.5267 ± 0.1065	+0.4017
Depth	8	77.1180 ± 0.0703	77.5302 ± 0.1482	+0.4122

E TAAC UNSEEN AND LONG-TAILED ID DIAGNOSTIC

We run an independent diagnostic repetition of the main TAAC comparison: Continuous and IDShare, each with seeds 2021, 42, and 2024, the two-layer RankMixer-based backbone, width 64, identity dimension 16, one epoch, and batch size 2,048. Global L2 and weight decay are zero; IDShare uses $K = 20$K. The training code, data processing, initialization, optimizer, and selection protocol are unchanged. The original jobs did not retain their checkpoints, so these six new runs retain weights and predictions. Predictions are captured from each run's validation forward pass, without an additional evaluation pass. For each diagnostic run, overall AUC recomputed from the saved predictions matches that run's own training report. These runs are reported separately, not pooled with or substituted for the original main-comparison seeds.

Frequency is the count of training lookups for each raw target ID, including target occurrences and the truncated histories consumed by the model. The train-only counts are the same as those used by TailShare. Validation samples are assigned to the fixed buckets 0, 1, 2–5, 6–10, 11–50, 51–100, 101–500, and 501+. Bucket 0 contains IDs unseen in training. Both methods map these IDs to one shared OOV index. Continuous reads the shared OOV row, which receives no training lookups; IDShare emits the shared zero-route output b_0. Nonpositive raw target IDs form a separate special bucket. The special bucket is empty and has undefined metrics.

All metrics average seeds equally. Logloss uses float64 probabilities clipped to $[10^{-15}, 1 - 10^{-15}]$. Bucket AUC measures ranking within a group, whereas overall AUC also contains cross-group pairs; the bucket values therefore do not add up to an attribution of the overall gain.

Table 9 reports both metrics, with means, sample standard deviations, and mean paired differences for every bucket. Table 10 gives sample and class counts, identical across all six runs.

Appendix F uses these trained checkpoints to compare continued updates with shared output vectors versus separate per-ID copies, keeping the backbone and routing fixed.

Table 9: **TAAC frequency diagnostic.** Independent reruns, three-seed means ± sample SD. AUC is in %; its differences are in pp. Δ is IDShare minus Continuous. Positive AUC and negative logloss differences indicate improvement. The special bucket is empty.

AUC (%)			
Lookup count	Continuous	IDShare	Δ
Overall	77.0736 ± 0.0275	77.6088 ± 0.0613	$+0.5352$
0 (unseen)	76.0626 ± 0.0235	76.2654 ± 0.0368	$+0.2029$
1	76.4859 ± 0.2210	76.6919 ± 0.1800	$+0.2059$
2–5	74.4252 ± 0.2046	74.6318 ± 0.2800	$+0.2066$
6–10	76.9794 ± 0.0427	77.0560 ± 0.0204	$+0.0766$
11–50	77.2095 ± 0.0230	77.1599 ± 0.0512	-0.0496
51–100	77.0331 ± 0.0043	77.0305 ± 0.0239	-0.0025
101–500	77.0016 ± 0.0467	76.9839 ± 0.0759	-0.0177
501+	78.3622 ± 0.0908	78.3594 ± 0.0866	-0.0029
Special	—	—	—
Logloss			
Lookup count	Continuous	IDShare	Δ
Overall	0.192138 ± 0.000533	0.189222 ± 0.000586	-0.002916
0 (unseen)	0.215883 ± 0.001388	0.205233 ± 0.001157	-0.010650
1	0.248555 ± 0.001140	0.243442 ± 0.001109	-0.005113
2–5	0.234971 ± 0.000535	0.232505 ± 0.000601	-0.002466
6–10	0.231325 ± 0.000934	0.228839 ± 0.000945	-0.002486
11–50	0.220460 ± 0.000506	0.220221 ± 0.000787	-0.000240
51–100	0.222935 ± 0.000195	0.223234 ± 0.000494	$+0.000299$
101–500	0.212162 ± 0.000160	0.212373 ± 0.000401	$+0.000211$
501+	0.169142 ± 0.000472	0.168974 ± 0.000426	-0.000169
Special	—	—	—

Table 10: **Validation samples in each fixed frequency bucket.** Counts are identical across all six runs. Special denotes nonpositive raw target IDs; this bucket is empty and its metrics are undefined.

Lookup count	Samples	Positive	Negative
Overall	21,951,746	1,236,199	20,715,547
0 (unseen)	5,579,027	344,535	5,234,492
1	186,420	14,685	171,735
2–5	508,866	36,771	472,095
6–10	315,887	22,844	293,043
11–50	1,070,712	74,469	996,243
51–100	623,166	44,157	579,009
101–500	2,036,859	134,377	1,902,482
501+	11,630,809	564,361	11,066,448
Special	0	0	0

E.1 Composition of learned sharing groups

Figure 4(b) uses the final ID-to-code mappings from the three IDShare checkpoints above, without further training. We sort 12,036,503 training-seen IDs by lookup count, breaking ties by internal ID, and divide them into ten disjoint groups whose sizes differ by at most one. The D1–D10 lookup ranges are 1–4, 4–7, 7–11, 11–16, 16–24, 24–39, 39–73, 73–168, 168–585, and 585–5,095,323. Equal-frequency IDs can cross a group boundary; the groups do not overlap. Unseen IDs and reserved rows are excluded; training-seen IDs assigned to the zero code are included. For each focal ID, we count its other same-code IDs in each group and divide by its total number of partners. Singleton-code IDs are excluded from this calculation. We average these proportions equally over focal IDs within each group, then equally over seeds. These are final partner compositions, not counts of past shared updates.

F UNTYING SHARED OUTPUTS AFTER TRAINING

Each TAAC IDShare checkpoint from Appendix E initializes two branches with identical predictions. Shared retains one output vector per code; untied gives each training-seen ID its own copy. Routing, the backbone, other features, and special-ID outputs stay frozen. Only output vectors learn, without straight-through routing or codebook fitting loss; target and history items still use the same table.

Both branches take 4,096 additional steps on identical batches of 2,048 rows, using fresh Adam states, learning rate 0.001, gradient clipping at 1, and no L2 or weight decay. This is a fixed-routing intervention after one epoch, not another epoch of end-to-end training. We record steps 0, 1,024, and 4,096; the endpoint is fixed at 4,096. Training fit uses a fixed 16,384-row probe per seed; validation uses the full split.

Table 11: **Retaining versus untying learned shared outputs.** Values are equal-weight three-seed means. Training AUC uses the fixed training probe; validation AUC uses the full validation split.

State	Train AUC (%)	Val. AUC (%)
Common start	81.1223	77.6088
Shared, step 4,096	81.9842	77.4684
Untied, step 4,096	85.0979	77.3345

Untying raises training-probe AUC on all three seeds. The validation metrics favor different branches: untying lowers overall logloss and the prespecified long-tail logloss, whereas retaining sharing yields higher AUC, also on all three seeds. Shared leads in AUC by 0.1339 pp overall and 0.1331 pp for IDs with 1–10 training lookups. Both branches finish below their starting validation AUC. This tests output ties after training, not the full benefit of learning the original representation.