ENHANCING CROSS-CATEGORY LEARNING IN RECOMMENDATION SYSTEMS WITH MULTI-LAYER EMBEDDING TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern DNN-based recommendation systems rely on training-derived real-valued embeddings of sparse categorical features. Input sparsity makes obtaining high-quality embeddings for rarely-occurring categories harder as their representations are updated infrequently. We demonstrate an effective overparameterization technique for enhancing embeddings training by enabling useful cross-category learning. Our scheme trains embeddings using training-time forced factorization of the embedding (linear) layer, with an inner dimension higher than the target embedding dimension.

We show that factorization breaks update sparsity via non-homogeneneous weighting of dense base embedding matrices. Such weighting controls the magnitude of weight updates in each embedding direction, and is adaptive to training-time embedding singular values. The dynamics of singular values further explains the puzzling importance of factorization inner dimension on learning enhancements.

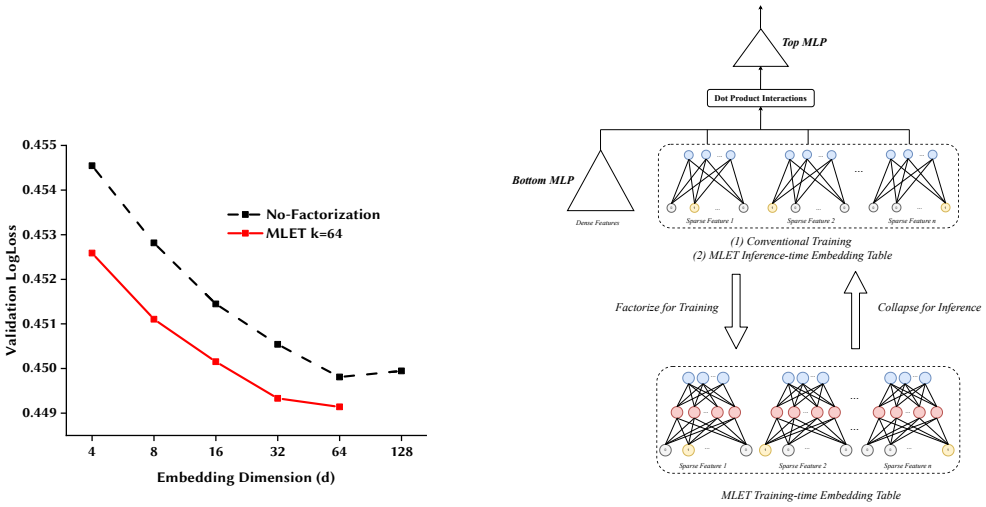
We call the scheme multi-layer embeddings training (MLET). For deployment efficiency, MLET converts the trained two-layer embedding into a single-layer one at the conclusion of training, avoiding inference-time model size increase. MLET consistently produces better models when tested on multiple recommendation models for click-through rate (CTR) prediction. At constant model quality, MLET allows embedding dimension reduction by up to 16x, and 5.8x on average, across the models. MLET retains its benefits in combination with other table-reduction methods (hashing and quantization).

1 INTRODUCTION

Recommendation models (RMs) underlie a large number of applications and improving their performance is increasingly important. The click-through rate (CTR) prediction task is a special case of general recommendation that seeks to predict the probability of a user clicking on a specific item. User reactions to earlier-encountered instances are used in training a CTR model and are described by multiple features that capture user information (e.g., age and gender) and item information (e.g., movie title, cost) Ouyang et al. (2019). Features are either numerical or categorical variables.

A fundamental aspect of modern recommendation models is their reliance on embeddings which map categorical variables into dense representations in an abstract real-valued space. State-of-the-art RMs increasingly use deep neural networks. Most high-performing models use a combination of multi-layer perceptrons (MLPs) to process dense features, linear layers to generate embeddings of categorical features, and either dot products or sub-networks that generate higher-order interactions. The outputs of the interaction sub-networks and MLPs are used as inputs into a linear (logistic) model to produce the CTR prediction. Broadly, the above describes modern deep recommender systems, including: Wide and Deep Cheng et al. (2016), Deep and Cross (DCN) Wang et al. (2017), DeepFM Guo et al. (2017), Field-Aware Factorization Machine (FFM) Juan et al. (2016), Neural Factorization Machine (NFM) He & Chua (2017), AutoInt Song et al. (2019), and xDeepFM Lian et al. (2018).

The contribution of this paper is in developing a simple yet effective overparameterization method of deriving superior embeddings through training-time factorization of linear embedding layers that enhances their cross-category learning. The proposed technique, which we call multi-layer embedding training (MLET), trains embeddings via a sequence of two linear layers, instead of a single one. Figure 1b illustrates the technique and the transformations involved.



(a) (b)

Figure 1: (a) CTR model quality worsens at low embedding dimensions (and model sizes). Improvements due to MLET's factorization can be traded for reduced model size. (b) Our strategy retains inference-time model size.

Effectiveness of training-time factorization of the embedding layer is puzzling. We identify the central role of a term that controls the magnitude of updates in different embedding directions. We refer to this term as the enhancement factor of embedding update directions. The non-homogeneous weighting of MLET breaks the sparse nature of traditional embedding updates and is cross-category informative: training on queried items also leads to embedding updates of all other non-queried items. We also point out that the properties of non-homogeneous weighting explain why training results are highly correlated with the inner dimension of embedding factorization. The inner dimension is very important because it determines the size of training-time embedding and is thus a practical limitation.

The improvements are consistent and non-negligible. A major challenge for RMs is balancing the improved performance afforded by higher embedding vector dimensions Naumov (2019); Yin & Shen (2018) with the increasingly heavy cost in deployment of embedding tables Ginart et al. (2019a). The memory required to store the embedding tables scales linearly with the embedding dimension and the trade-off is illustrated in Figure 1a using the DLRM model Naumov et al. (2019) on the Criteo-Kaggle dataset Labs (2014). This fundamental trade-off between model quality and size makes it natural to think of learning improvements provided by MLET in terms of not just higher-quality models but also reductions in model size at constant model quality. We implement MLET in PyTorch and test it on seven state-of-the-art recommendation models for CTR prediction. We demonstrate the substantial benefits of MLET in producing superior models – it achieves the same or better performance than single-layer models while using up to 16x less (5.8x less on average) parameters in embeddings on two public CTR datasets.

2 Better Embeddings via Forced Cross-Category Learning

In training recommendation models, a key feature that differs from other DNN setups is the sparsity of queries in embedding training. Further, some items are rarely-occurring. *How can we improve their learning quality?* We first observe that in each iteration of stochastic gradient descent (SGD), the only updated embeddings in the single-layer model are those of the queried categories/items. Because of this, the embedding update in each training iteration is sparse and embeddings of rarely-occurring items are updated infrequently.

However, we show that with multi-layer embedding training, embeddings of all categories/items are updated in each training iteration. Extra knowledge of the queried categories/items is extracted and shared to update embeddings of other categories/items that are not queried. We refer to this extra knowledge shared from the queried categories/items to other non-queried categories/items as *cross-category information*. Cross-category information leads to dense embedding updates and much more effective learning of rarely-occurring items.

2.1 EMBEDDINGS AND THEIR FACTORIZATION

We now introduce the notation and details of MLET. Let the full embedding table be W of size $d \times n$, where n is the number of elements in the table and d is the embedding dimension. Note that each column of the table represent the embedding of a category/item.

$$W \in \mathbb{R}^{d \times n} \tag{1}$$

We consider a two-layer architecture and factorize the embedding table W in terms of W_1 and W_2:

$$W = W_1 W_2 \quad \text{with } W_1 \in \mathbb{R}^{d \times k}, W_2 \in \mathbb{R}^{k \times n} \tag{2}$$

In the above equations, k is a hyperparameter for the inner dimension of embedding factorization. (Note that k does not need to be equal to d.) Let the vector $q \in \mathbb{Z}^n$ denote a one-hot encoding of a query to n categories/items. The embedding lookup is represented by a matrix-vector product:

$$r = W_1 W_2 q \tag{3}$$

Here, $r \in \mathbb{R}^d$ is the embedding of the queried item. W_1 and W_2 are trained jointly. After training there is no need to keep both W_1 and W_2, and we only store their product, $W = W_1 W_2$. This reduces a two-layer embedding into a single one for inference-time evaluation and storage. Note that this simple transformation reduces the model storage cost from O(nk) to O(nd), which is significant since k is usually much larger than d.

2.2 BREAKING SPARSITY OF UPDATES VIA EMBEDDING FACTORIZATION

Why should we expect to obtain a better embedding if we factorize the linear layer? Specifically, in the MLET operating regime of $k \geq d$, any embedding defined by a two-layer model lies in the search space of a single-layer model. Therefore, if there is an optimal solution found by a two-layer model, our intuition is that a single-layer model should also be able to find it, and, thus, a two-layer model should not be better than a single-layer model. Yet, empirically, we find that two-layer models consistently outperform their single-layer counterparts. To understand why factorization helps, we analyze the dynamics of embedding updates for both single-layer and multi-layer models. Our analysis assumes that SGD with batch size 1 is used in model training. The same conclusions can be easily extended to SGD with other batch sizes.

Our analysis pins down the superiority of multi-layer training to its more frequent and more informative embedding updates that learn the correlation between items. Mathematically, we show that these more informative updates can be seen as the result of applying a special linear transformation on single-layer models' updates. This linear transformation is fully parameterized by the SVD of training-time embeddings.

Consider the 2-layer embedding $W = W_1 W_2$ of Eq.3. The factorization explicitly formulates the embeddings as linear combinations of the embedding basis formed by the columns in W_1. Because the embedding basis is shared, correlations between all items are implicitly explored during training. At each iteration of training, the loss on a few queried items is used to adjust the whole embedding basis. By analyzing the gradient updates, we show that the learning of correlation between items leads to more frequent and more informative embedding updates. Let the loss function be L and loss gradient $G = \frac{\partial L}{\partial w}$. Given a learning rate η, for training of the single-layer model, embedding updates are given by:

$$W = W - \eta G \tag{4}$$

Note that matrix G is sparse, with only one column being non-zero. To see this, first notice that q is a one-hot encoding vector representing the queried item. Let $g = (g_1, g_2, .., g_d)^T$ be the gradient of loss w.r.t. r, i.e., $g = \partial L / \partial r$. Let C represent the index of the queried item.

$$G_{ij} = \frac{\partial L}{\partial W_{ij}} = \sum_{k=1}^{d} \frac{\partial L}{\partial r_k} \frac{\partial r_k}{\partial W_{ij}} = g_i q_j \tag{5}$$

Eq.5 uses the fact that $\partial L / \partial r_k = g_k$ and $\partial r_k / \partial W_{ij} = q_j$ when $k = i$ and is 0 otherwise. In matrix form it is equivalent to $G = g q^T$, because $q_j = 1$ only when $j = C$, otherwise $q_j = 0$. It follows that only the C^{th} column of G is non-zero and is equal to g. With batch size $b > 1$, the conclusion extends: no more than b columns are non-zero. Because $b << n$ for most embedding tables, G is

still sparse. In contrast, with the same learning rate, embedding updates of the 2-layer model are captured by:

$$W = W - \eta W_1 W_1^T G - \eta G W_2^T W_2 \tag{6}$$

The derivation is as follows. First, the gradients w.r.t. W_1 and W_2 are (derivation similar to Eq.5):

$$\frac{\partial L}{\partial W_1} = g q^T W_2^T = G W_2^T \tag{7}$$

$$\frac{\partial L}{\partial W_2} = W_1^T g q^T = W_1^T G \tag{8}$$

Let $W(t)$ be the embedding at t^{th} iteration. G is the gradient of embedding at the same iteration, $G = \partial L / \partial W(t)$. The embedding updates of the 2-layer model are:

$$\begin{aligned} W(t+1) &= W_1(t+1) W_2(t+1) \\ &= (W_1(t) - \eta \frac{\partial L}{\partial W_1(t)})(W_2(t) - \eta \frac{\partial L}{\partial W_2(t)}) \end{aligned} \tag{9}$$

Bringing Eq.7,8 into Eq.9 and replacing $W_1(t)W_2(t)$ with $W(t)$, we get:

$$\begin{aligned} W(t+1) = W(t) &- \eta W_1(t) W_1(t)^T G \\ &- \eta G W_2(t)^T W_2(t) + O(\eta^2) \end{aligned} \tag{10}$$

Following the convention of gradient flow analysis, we ignore the $O(\eta^2)$ term in Eq.10 to get Eq.6. Comparing Eq.4 and Eq.6, we can see that *the embedding updates of 2-layer training are more frequent than those of single-layer training and have the property of being cross-category informative, which cannot be obtained by single-layer training*. We first observe that in each step of training a single-layer model (Eq.4), only one column of W is updated. However, in each step of training a 2-layer model (Eq.6), the whole embedding table W is updated because the term $\eta G W_2^T W_2$ breaks the sparsity ($W_2^T W_2$ is usually dense and post-conditioning a column-sparse matrix by dense matrices breaks the column-sparsity). This implies that given the same number of training iterations (N), embeddings are updated N times more frequently for the 2-layer model than for the single-layer model.

In summary, Eq.4 and Eq.6 show that the updates of the two models differ by a pre-conditioning term $W_1 W_1^T$ and a post-conditioning term $W_2^T W_2$ on the gradient G. Further, the post-conditioning term breaks the sparsity.

2.3 Improving Cross-Category Learning via Embedding Factorization: Theory

Why does breaking the sparsity in this way help embedding training? We now explain it via the structure of the updates of the two models. The theory reformulates the process of embedding updates and represents the difference between updates by a single term that scales different embedding directions. This scaling term effectively strengthens the updates along directions that have significant contribution to embedding's SVD spectrum and suppresses the updates along directions that have minor contribution. The theory also helps explain other properties of MLET, namely, its dependence on inner dimension.

We use the following notation: vec(X) is the vectorization of the matrix X, formed by stacking the columns of X into a single column vector. $\otimes$ is the Kronecker product operator. The SVDs of W_1 and W_2 are denoted by $W_1 = U \Sigma_1 X^T$ and $W_2 = Y \Sigma_2 V^T$, and u_i and v_j represent the i^{th} column of U and j^{th} column of V, respectively. $\sigma_1(i), \sigma_2(j)$ are the i^{th} singular value in Σ_1 and j^{th} singular value in Σ_2. Note that the range of index i is from 1 to d and that of index j is from 1 to n. We make the following two claims.

Claim 1. $S = \{v_j \otimes u_i, i \in \{1, 2, ..., d\}, j \in \{1, 2, ..., n\}\}$ is an orthornormal basis in $\mathbb{R}^{nd}$.

Claim 2. There must exist a set of g_{ij} with $i \in \{1, 2, ..., d\}$ and $j \in \{1, 2, ...n\}$ such that

$$\text{vec}(G) = \sum_{i,j} g_{ij} v_j \otimes u_i \tag{11}$$

The proof to **Claim 1.** is presented in Appendix. **Claim 2.** follows directly from the fact that vec(G) is in $\mathbb{R}^{nd}$ and S is an orthornormal basis in the same vector space. Based on the above claims, embedding updates for both models can be re-formulated as in the following theorem.

Theorem 1. (Main Theorem) The embedding updates of the 1-layer model and those of the 2-layer model can be represented in terms of the basis:

$$W - \eta G = W - \eta \sum_{i,j} g_{ij} u_i v_j^T \tag{12}$$

$$W - \eta(W_1 W_1^T G + G W_2^T W_2) = W - \eta \sum_{i,j} g_{ij} (\sigma_1(i)^2 + \sigma_2(j)^2) u_i v_j^T \tag{13}$$

The proof is presented in the Appendix. This theorem re-formulates the embedding updates as weighted sums of a set of base matrices, formed by outer products of singular vectors of embeddings. *Thus, it pins down the source of cross-category information to a re-weighting process that is guided by squared terms of singular values of embeddings.*

In the single-layer model (Eq.12), the update in direction $u_i v_j^T$ is with magnitude g_{ij}. In the 2-layer model (Eq.13), the magnitude is adjusted by a factor of $(\sigma_1(i)^2 + \sigma_2(j)^2)$. This term acts as an enhancement factor of updates and is adaptive to the training-time embeddings. If u_i and v_j make significant contributions to embeddings, the corresponding $\sigma_1(i)$ and $\sigma_2(j)$ will be large, so updates in the direction $u_i v_j^T$ get boosted. If u_i and v_j make minimal contribution to embeddings, the corresponding $\sigma_1(i)$ and $\sigma_2(j)$ will be small and the update in the direction $u_i v_j^T$ gets attenuated.

Note that Eq.12 is sparse; Eq.4 was shown to be sparse in Section 2.2. Now this sparsity can be seen as the result of homogeneous weighting of dense embedding matrices $g_{ij} u_i v_j^T$. In contrast, the enhancement factor $(\sigma_1(i)^2 + \sigma_2(j)^2)$ produces heterogeneous weighting of these dense matrices, thereby breaking the sparsity.

2.4 Learning Enhancement: Requirements on Overparameterization

There are two aspects of MLET that seem quite surprising. The first is why, with an appropriate choice of inner dimension, it is superior to single-layer training at the same embedding dimension? The second is why its quality improves with a larger inner dimension? Empirically, higher ks consistently achieve higher performance than small ones. With $k < d$, MLET can be worse than a single-layer model. Since the inference-time embedding table is of size $n \times d$, factorization with $k > d$ introduces more parameters ($nk + kd$) than needed (nd) and overparameterizes the model. *Why do the benefits of such overparameterization increase with k?*

There are two factors contributing to these phenomena: degeneration of enhancement factor and search space of the embedding learning. In Section 2.3, we attribute the superiority of 2-layer embeddings over 1-layer embeddings to a term $(\sigma_1(i)^2 + \sigma_2(j)^2)$. Notice that $i \in \{1, 2, .., d\}$ and $j \in \{1, 2, .., n\}$ and $\sigma_1(x) = \sigma_2(x) = 0$ if $x > k$. It means that if k is small, the majority of $\sigma_1(i)$ and $\sigma_2(j)$ will be 0. *We believe this explains, why the performance of MLET degrades when the inner dimension shrinks.* Further, we observe that in 1-layer model (Eq.12), the enhancement factor of different update directions $u_i v_j^T$ can be treated as constant 1.

For $k \geq d$, consider two MLET models with inner dimensions $k_{larger}, k_{smaller}$ ($k_{larger} > k_{smaller} \geq d$). The number of non-zero enhancement factors is the same in both models: dn. However, the number of enhancement factors with a non-zero σ_2 differs since it is equal to kd. Thus the MLET model with inner dimension $k_{smaller}$ has $d(k_{larger} - k_{smaller})$ fewer effective enhancement factors because of σ_2 being 0. Notice that $\sigma_2(j)$ measures the importance of v_j to the embedding table—it is informative in determining the confidence in taking update $u_i^T v_j, i \in \{1, 2, .., d\}$. The use of less informative enhancement factors in a larger number of update directions explains why MLET with inner dimension $k_{smaller}$ performs worse than MLET with inner dimension k_{larger}.

For $k < d$, there are $(-k^2 + (n+d)k)$ non-zero enhancement factors in the 2-layer model. (To see this, consider that the number of non-zero enhancement factors equals to the number of terms with at least one of $\sigma_1(i)$ and $\sigma_2(j)$ being non-zero. For $i \in \{1, .., k\}$, all $\sigma_1(i)$s are non-zero, so their related enhancement factors are non-zero and there are $k \times n$ such values. For $i \in \{k+1, .., d\}$, all $\sigma_1(i)$s are zero and enhancement factors are non-zero only when $j \in \{1, .., k\}$. There are $(d-k)k$ such values. Thus, there are $kn + (d-k)k$ non-zero enhancement factors in total.) However, the number

of non-zero enhancement factors in the 1-layer model is dn (for all dn terms, the enhancement factor is 1). Notice that $dn > (-k^2 + (n+d)k)$ for all $k < d$. It indicates that single-layer models consider more $u_i v_j^T$ terms in their embedding updates than MLET models with $k < d$. For such MLET models, this lack of flexibility in update directions cancels the benefit of being more informative in each direction, and possibly worsens performance. We point out that there is another, intuitive reason for MLET being worse than a single-layer model in the case of $k < d$: the search space of the multi-layer model is reduced to a subset of the single-layer search space. We do not recommend working in the $k < d$ regime.

3 Experiments

We evaluate the proposed MLET technique using a large set of state-of-the-art recommendation models on two public datasets for click-through rate tasks: Criteo-Kaggle and Avazu. Both datasets are composed of a mix of categorical and real-valued features (Table 1).

The Criteo-Kaggle dataset is split based on the time of data collection: the first six days are used for training and the seventh day is split evenly into the test and validation sets. The Avazu dataset is randomly split into training and test sets of 90% and 10%, respectively. The models are implemented in PyTorch and trained on systems with NVIDIA GPUs (CUDA acceleration enabled). We evaluate MLET on seven state-of-the-art recommendation models. DLRM is tested both on Criteo-Kaggle and Avazu. Other models are tested exclusively on the Avazu dataset because of its shorter run time relative to Criteo-Kaggle. We use publicly available implementations of non-DLRM models from the open-source recommendation model library DeepCTR-Torch Shen (2019). To decrease the impact of randomized initialization and run-to-run variation due to non-deterministic GPU execution, the reported results are averages over several training runs. All DLRM results are the mean of five training runs. Results from the other models are the mean of three training runs. We report two quality metrics: area under the ROC curve (AUC) and binary cross-entropy (LogLoss).

Following prior work Naumov et al. (2019), we train all models for a single epoch. Two optimizers are used in the experiments: SGD and Adagrad. DLRM and its MLET variants are trained using SGD with a learning rate of 0.2. Other models are trained with Adagrad with a learning rate of 0.02. Most recommendation systems use the same overall architecture (embeddings + interactions + MLP). The biggest difference between most models is in the interaction layers. In all experiments, d stands for embedding dimension. For DLRM, on both datasets we configure its top MLP to have two hidden layers with 512 and 256 nodes. On the Avazu dataset, we set DLRM's bottom MLP to be $256 \rightarrow 128 \rightarrow d$. On the Criteo-Kaggle dataset, we configure DLRM's bottom MLP to be $512 \rightarrow 256 \rightarrow 128 \rightarrow d$. For other models we keep the same criteria for d and use all default model parameters from the DeepCTR implementations. We do not tune any model architecture (e.g., number, type, and size of layers). We summarizd the configurations used in Table 2.

3.1 Learning Enhancement

The experiments demonstrate the effectiveness of MLET in producing superior models compared to the baseline single-layer embedding implementation. Figures 2 and 3 summarize the more extensive experiments with DLRM carried out on two datasets for models trained using SGD. Figure 4 presents the main results for three other models: DCN, NFM, and AutoInt. Table 2 summarizes the results of MLET across 7 models for several values of k and d.

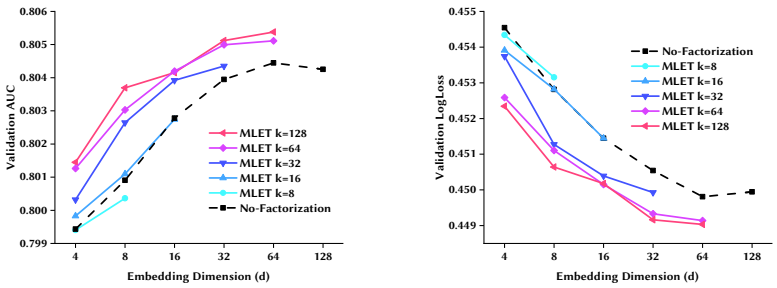


Figure 2: MLET with DLRM on the Criteo-Kaggle dataset, trained with SGD.

Table 1: Dataset composition.

Dataset	Samples	Dense Features	Sparse Features
Criteo-Kaggle Labs (2014)	45,840,617	13	26
Avazu Kaggle (2014)	40,400,000	1	21

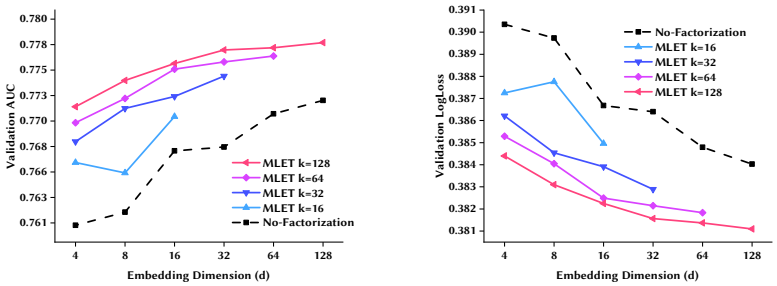


Figure 3: MLET with DLRM on the Avazu dataset, trained with SGD.

As Figures 2 and 3 show, MLET consistently squeezes more performance out of fixed-size embeddings of DLRM model. The benefits begin to be observed in MLET curves even for $k = d$. Increasing k for a given d leads to a monotonic improvement in model accuracy. *For CTR systems, an improvement of 0.001 in AUC is considered substantial*. The maximum AUC benefit of MLET for Criteo-Kaggle is 0.0027, and the maximum benefit for Avazu is 0.1241. This improvement in model accuracy saturates as k grows, e.g., on the Criteo-Kaggle dataset the curves with $k = 64$ and $k = 128$ are very similar.

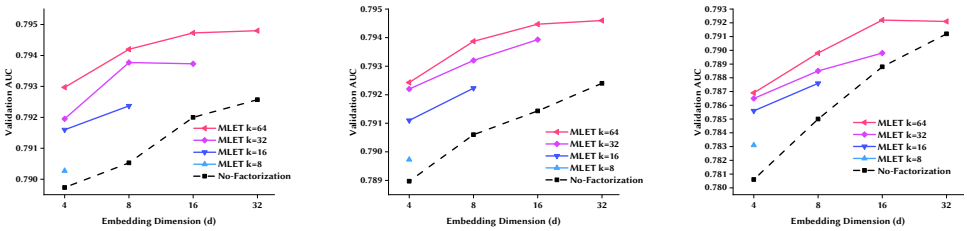
We further observe that the relative quality improvements are largely defined by k/d.

As can be seen in Figures 2 and 4, the general LogLoss vs. vector dimension behavior is similar across the different models evaluated. We note that not only is the overall behavior similar, but also that MLET provides substantial benefits for most models with $4-16\times$ savings of embedding parameters while maintaining the same or better performance as compared to the single-layer embedding training.

3.2 Learning Quality for High- and Low-Frequency Embeddings

In Section 2 we argue that the embedding updates of MLET are cross-category informative and are more frequent. They lead to better learning quality of embeddings, especially those of the least frequently queried items. To verify this intuition, we conduct experiments that compare the performance of MLET and that of single-layer training on two test sets.

The first set (A) is composed by 10% test samples with the most frequently queried items. The second set (B) is composed by 10% test samples with the least frequently queried items. Experiments are done with three models (DCN, AutoInt, and xDeepFM) on the Avazu dataset. We use the relative



(a) AutoInt (b) DCN (c) NFM

Figure 4: MLET on several state-of-the-art RM models on the Avazu dataset, trained with Adagrad.

Table 2: Effectiveness of MLET across models for two common settings.

Model	Dataset	Baseline AUC $d = 4$	Baseline AUC $d = 16$	MLET AUC (k, d) 64,4	MLET AUC (k, d) 64,16	Maximum Memory Reduction (same or higher performance)
DLRM	Criteo	0.799	0.803	0.801	0.804	4x (d 32 $\rightarrow$ 8 , $k \geq 128$)
	Avazu	0.761	0.768	0.770	0.775	16x (d 64 $\rightarrow$ 4 , 128 $\rightarrow$ 8 , $k \geq 128$)
Wide & Deep	Avazu	0.789	0.790	0.792	0.793	4x (d 16 $\rightarrow$ 4 , $k \geq 64$)
DeepFM	Avazu	0.790	0.794	0.792	0.795	1x
xDeepFM	Avazu	0.792	0.796	0.794	0.798	2x (d 16 $\rightarrow$ 8 , $k \geq 64$)
Deep & Cross	Avazu	0.789	0.791	0.792	0.794	8x (d 32 $\rightarrow$ 8 , 16 $\rightarrow$ 4 , $k \geq 64$)
AutoInt	Avazu	0.790	0.792	0.793	0.795	8x (d 32 $\rightarrow$ 4 , $k \geq 32$)
NFM	Avazu	0.781	0.789	0.787	0.792	4x (d 16 $\rightarrow$ 4 , $k \geq 64$)

improvement in PR-AUC to evaluate MLET's enhancement in the learning quality of embeddings. We choose PR-AUC instead of ROC-AUC because it is more robust to imbalanced data (20% of set A are clicked while only 15% of set B are clicked) and is also more sensitive to the improvements for the positive class Czakon (2021) (clicked samples in CTR task).

As shown in Table 3, MLET generally improves embedding quality on both sets of samples: on both set A and B, PR-AUC is improved with MLET except for one configuration (xDeepFM with k=4,d=8). Further, MLET consistently improves performance on the least popular samples (set B) and the improvements on them are larger than the improvements on the most popular samples (set A). This empirical observation aligns with our expectation from the theory that MLET's dense and cross-category informative updates are most beneficial to the learning quality of the embeddings of rarely-occurring items.

Table 3: Improvement of PR-AUC on test samples with the most frequently queried items and on test samples with the least frequently queried items. Aligned with our theory, the learning quality of low-frequency embeddings benefits more from MLET.

Model (d / k)	Set A (Most Frequent)	Set B (Least Frequent)
DCN (4 / 8)	+0.0004	+0.0013
DCN (16 / 64)	+0.0033	+0.0045
AutoInt (4 / 8)	+0.0005	+0.0008
AutoInt (16 / 64)	+0.0038	+0.0048
xDeepFM (4 / 8)	-0.0009	+0.0019
xDeepFM (16 / 64)	+0.0001	+0.0024

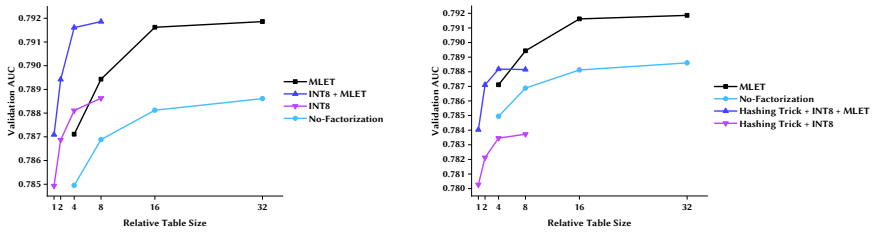
3.3 MLET AND MODEL COMPRESSION

We have demonstrated MLET's capability in producing superior models. However, there are many other model compression techniques that can also be used to squeeze more performance out of a fixed-size model. We develop experiments to compare and test the composition of MLET with these model size reduction techniques. Specifically, we include three commonly adopted techniques in our experiments – low rank SVD approximation, hashing, and post-training quantization. We found that MLET consistently outperforms low rank SVD approximation of embedding tables and improves model quality with all combinations of quantization and hashing. MLET consistently enhances the INT8 post-training quantization and hashing, as observed in Figure 5. Experimental results on SVD, further results on quantization and hashing, and a discussion of experimental settings are in the Appendix due to space limitations.

4 RELATED WORK

We discuss three primary tracks of prior work related to MLET: (1) theoretical aspects of training evolution, (2) table-compression approaches, and (3) table-decomposition approaches.

The theory we develop for the evolution of embeddings in gradient descent training of recommendation models is inspired by recent work on the benefits of training with overparameterization Arora et al. (2018; 2019). However, we find that this prior work does not directly apply to MLET be-



(a) MLET and quantization

(b) MLET with quantization and hashing trick

Figure 5: Composition of MLET with quantization and Hashing Trick on DCN. The combination of MLET and quantization provides improved performance.

cause MLET-trained embeddings are not lower rank than with traditional training. Moreover, this prior theoretical framework does not predict the impact of the inner dimension on the evolution of parameters, which is because of their restrictive assumptions on the initialization of the matrices. Our newly-developed theory relaxes the restrictive assumptions and successfully explains the empirically-observed benefits of using higher inner dimensions and the impact of initialization variance.

The benefit of MLET is in producing embedding tables with superior performance for fixed table size. An orthogonal set of approaches for achieving this goal include compressing a large trained embedding with pruning and quantization Ling et al. (2016); Tissier et al. (2019); Sun et al. (2016); pruning and quantization applied during training Alvarez & Salzmann (2017); Naumov et al. (2018); and applying different types of hashing tricks to share embeddings within or between tables Attenberg et al. (2009); Shi et al. (2019). Techniques that utilize statistical knowledge of embedding usage (access frequency) have also been developed to adapt the embedding dimension or precision to usage, with more-compact representations of less-accessed embeddings Ginart et al. (2019b); Yang et al. (2020).

In addition to being an orthogonal approach to those above, MLET produces high-quality embeddings without assuming any prior knowledge of access frequency and without reducing parameter precision. Instead, under the same inference-time embedding size, we seek a superior training-time embedding architecture that achieves better performance by promoting more frequent and more informative updates of embeddings than those in single-layer training.

MLET is also related to approaches that are based on decomposition techniques. For example, trained embedding tables can be compressed via a low-rank SVD approximation Bhavana et al. (2019) or using a tensor-train decomposition Khrulkov et al. (2019). TT-Rec Yin et al. (2021) uses tensor-train decomposition to represent embeddings and is similar to MLET in that multiple tensors instead of one are used in learning each embedding table. However, TT-Rec and MLET are orthogonal and completely differ in their working mechanisms and implications. TT-Rec shows that *underparameterization* of an embedding layer maintains performance while reducing the inference-time table size. MLET, in contrast, shows that *overparameterization* of an embedding layer using two layers enhances performance while keeping the same inference-time table size.

With the empirical benefits demonstrated by MLET, we believe many opportunities are now open for exploring the combinations of MLET with the above techniques. For example, one can apply TT-Rec to learn the first layer of MLET, or one can overparameterize the tensors in TT-Rec to see if desirable performance improvement can be obtained.

We conclude this survey by pointing out that no other work has shown how to enhance cross-category training or theoretically analyzed its mechanism.

5 Conclusion

We introduce a simple yet effective multi-layer embedding training (MLET) architecture that trains embeddings via a sequence of linear layers to derive superior models. We present a theory that explains for its superior embedding learning based on the dynamics of embedding updates. We prototype MLET across seven state-of-the-art open-source recommendation models and demonstrate that MLET alone is able to achieve the same or better performance as compared to conventional single-layer training scheme while uses up to 16x less (5.8x less on average) embedding parameters.

REFERENCES

Jose M Alvarez and Mathieu Salzmann. Compression-aware training of deep networks. In *Advances in Neural Information Processing Systems*, pp. 856–867, 2017.

Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. *ArXiv*, abs/1802.06509, 2018.

Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. Implicit regularization in deep matrix factorization. In *NeurIPS*, 2019.

Josh Attenberg, Kilian Weinberger, Anirban Dasgupta, Alex Smola, and Martin Zinkevich. Collaborative email-spam filtering with the hashing trick. CEAS, 2009.

Prasad Bhavana, Vikas Kumar, and Vineet Padmanabhan. Block based singular value decomposition approach to matrix factorization for recommender systems. *arXiv preprint arXiv:1907.07410*, 2019.

Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. *CoRR*, abs/1606.07792, 2016. URL http://arxiv.org/abs/1606.07792.

Jakub Czakon. F1 score vs roc auc vs accuracy vs pr auc: Which evaluation metric should you choose? https://neptune.ai/blog/f1-score-accuracy-roc-auc-pr-auc, 2021. [Online; Updated December 31st, 2021].

Antonio Ginart, Maxim Naumov, Dheevatsa Mudigere, Jiyan Yang, and James Zou. Mixed dimension embeddings with application to memory-efficient recommendation systems. *ArXiv*, abs/1909.11810, 2019a.

Antonio Ginart, Maxim Naumov, Dheevatsa Mudigere, Jiyan Yang, and James Zou. Mixed dimension embeddings with application to memory-efficient recommendation systems, 2019b.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: A factorization-machine based neural network for ctr prediction. In *IJCAI*, 2017.

Xiangnan He and Tat-Seng Chua. Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pp. 355–364, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350228. doi: 10.1145/3077136.3080777. URL https://doi.org/10.1145/3077136.3080777.

Yu-Chin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for ctr prediction. In *RecSys '16*, 2016.

Kaggle. Avazu click-through rate prediction, 2014. https://www.kaggle.com/c/avazu-ctr-prediction.

Valentin Khrulkov, Oleksii Hrinchuk, Leyla Mirvakhabova, and Ivan Oseledets. Tensorized embedding layers for efficient model compression. *arXiv preprint arXiv:1901.10787*, 2019.

Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *ArXiv*, abs/1806.08342, 2018.

Criteo Labs. Kaggle display advertising challenge dataset, 2014. http://labs.criteo.com/2014/02/kaggle-display-advertising-challenge-dataset/.

Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

Shaoshi Ling, Yangqiu Song, and Dan Roth. Word embeddings with limited memory. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 387–392, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-2063. URL https://www.aclweb.org/anthology/P16-2063.

Maxim Naumov. On the dimensionality of embeddings for sparse features and data. *arXiv preprint arXiv:1901.02103*, 2019.

Maxim Naumov, Utku Diril, Jongsoo Park, Benjamin Ray, Jedrzej Jablonski, and Andrew Tulloch. On periodic functions as regularizers for quantization of neural networks. *arXiv preprint arXiv:1811.09862*, 2018.

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 Y. 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. *ArXiv*, abs/1906.00091, 2019.

Wentao Ouyang, Xiuwu Zhang, Shukui Ren, Linlin Li, Zhaojie Liu, and Y. Du. Click-through rate prediction with the user memory network. *ArXiv*, abs/1907.04667, 2019.

Weichen Shen. Deepctr-torch: Easy-to-use,modular and extendible package of deep-learning based ctr models. https://github.com/shenweichen/DeepCTR-Torch, 2019.

Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. Compositional embeddings using complementary partitions for memory-efficient recommendation systems. *arXiv preprint arXiv:1909.02107*, 2019.

Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *CIKM '19*, 2019.

Fei Sun, Jiafeng Guo, Yanyan Lan, Jun Xu, and Xueqi Cheng. Sparse word embeddings using l1 regularized online learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pp. 2915–2921. AAAI Press, 2016.

Julien Tissier, Christophe Gravier, and Amaury Habrard. Near-lossless binarization of word embeddings. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:7104–7111, Jul 2019. ISSN 2159-5399. doi: 10.1609/aaai.v33i01.33017104. URL http://dx.doi.org/10.1609/aaai.v33i01.33017104.

Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & cross network for ad click predictions. In *ADKDD'17*, 2017.

Jie Amy Yang, Jianyu Huang, Jongsoo Park, Ping Tak Peter Tang, and Andrew Tulloch. Mixed-precision embedding using a cache, 2020.

Chunxing Yin, Bilge Acun, Xing Liu, and Carole-Jean Wu. Tt-rec: Tensor train compression for deep learning recommendation models. *ArXiv*, abs/2101.11714, 2021.

Zi Yin and Yuanyuan Shen. On the dimensionality of word embedding. *ArXiv*, abs/1812.04224, 2018.

A Appendix

A.1 Proof to Claim 1

Proof. Consider two vectors from S: $v_j \otimes u_i$ and $v_q \otimes u_p$.

$$(v_j \otimes u_i)^T \cdot (v_q \otimes u_p) = (v_j^T \otimes u_i^T) \cdot (v_q \otimes u_p) = (v_j^T v_q) \otimes (u_i^T u_p) \tag{14}$$

Bringing $i = p, j = q$ into Eq.14, we see that both $(u_i^T u_i)$ and $(v_j^T v_j)$ are 1 so the product of every vector $v_j \otimes u_i \in S$ to itself is 1. The product of any two different vectors is 0. Indeed, $i = p$ and $j = q$ cannot hold true at the same time, so either $v_i^T v_p = 0$ or $v_i^T v_p = 0$. In consequence, $(v_i^T v_p) \otimes (u_j^T u_q) = 0$. □

A.2 Proof to Main Theorem

Proof. Eq.12 follows from **claim 2**. Now we show that the right side of Eq.13 equals to its left side.

Recall that the SVDs of W_1 and W_2 are $W_1 = U\Sigma_1 X^T$ and $W_2 = Y\Sigma_2 V^T$. Let I_k denote the identity matrix of shape $k \times k$. Use the property $\text{vec}(ABC) = (C^T \otimes A)\text{vec}(B)$

$$\begin{aligned}
&\text{vec}(W_1 W_1^T G + G W_2^T W_2) \\
&= \text{vec}(U\Sigma_1\Sigma_1^T U^T G) + \text{vec}(GV\Sigma_2^T\Sigma_2 V^T) \\
&= \text{vec}(U\Sigma_1\Sigma_1^T U^T G I_n) + \text{vec}(I_d G V\Sigma_2^T\Sigma_2 V^T) \\
&= (I_n \otimes U\Sigma_1\Sigma_1^T U^T)\text{vec}(G) + (V\Sigma_2^T\Sigma_2 V^T \otimes I_d)\text{vec}(G)
\end{aligned} \tag{15}$$

Use the property $AB \otimes CD = (A \otimes C)(B \otimes D)$ and $(A \otimes B)^T = (A^T \otimes B^T)$

$$\begin{aligned}
\text{Let } Q =& (I_n \otimes U\Sigma_1\Sigma_1^T U^T) + (V\Sigma_2^T\Sigma_2 V^T \otimes I_d) \\
=& (VI_nV^T \otimes U\Sigma_1\Sigma_1^T U^T) + (V\Sigma_2^T\Sigma_2 V^T \otimes UI_dU^T) \\
=& (V \otimes U)(I_n \otimes \Sigma_1\Sigma_1^T)(V^T \otimes U^T) \\
&+ (V \otimes U)(\Sigma_2^T\Sigma_2 \otimes I_d)(V^T \otimes U^T) \\
=& (V \otimes U)(I_n \otimes \Sigma_1\Sigma_1^T + \Sigma_2^T\Sigma_2 \otimes I_d)(V \otimes U)^T \\
=& \sum_{i,j} (v_j \otimes u_i)(\sigma_1(i)^2 + \sigma_2(j)^2)(v_j \otimes u_i)^T
\end{aligned} \tag{16}$$

Bringing Eq.16,11 into Eq.15 and notice that $v_j \otimes u_i$ are orthornormal, we have

$$\begin{aligned}
\text{vec}(W_1 W_1^T G + G W_2^T W_2) =& Q\text{vec}(G) \\
=& \sum_{i,j} g_{ij}(\sigma_1(i)^2 + \sigma_2(j)^2)(v_j \otimes u_i)
\end{aligned} \tag{17}$$

Notice that $(v_j \otimes u_i)$ is the vectorization of $(u_i v_j^T)$, so Eq.17, when converted into matrix form, is equivalent to

$$W_1 W_1^T G + G W_2^T W_2 = \sum_{i,j} g_{ij}(\sigma_1(i)^2 + \sigma_2(j)^2)(u_i v_j^T). \tag{18}$$

Bringing Eq.18 into Eq.6, we get Eq.13 and conclude the proof. □

A.3 Extended Experiments Details

Initialization strategy used for embedding layers is of critical importance in training RMs. In conventional RMs, the embedding table of each sparse feature is represented by a single linear layer. We follow a conventional approach in initializing this layer that uses Xavier initialization scheme Glorot & Bengio (2010).

MLET adds another linear factorization layer. We use a Gaussian distribution to initialize this second factorization layer. To make MLET effective, initialization variance cannot be too small. As

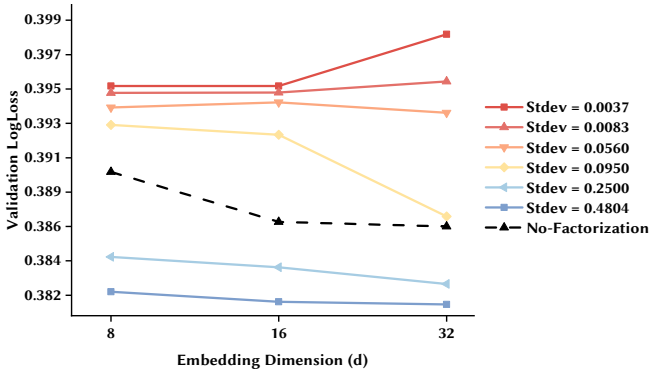


Figure 6: Search of initialization variance (embedding factorization) for DLRM with MLET $k = 32$. Small variance leads to vanishing enhancement factors hence poor performance.

suggested by our **Main Theorem**, small initialization effectively leads to vanishing enhancement factor and slows down embedding updates. This results in poor performance as shown in Figure 6. Empirically, if variance is too high then the training suffers from convergence issue. In all the experiments, we set the initialization standard deviation to 0.25 for DLRM and 0.5 for other models unless otherwise noted. Those values ensure the effectiveness of MLET while preserving training-time convergence.

A.4 MLET AND MODEL COMPRESSION

We have demonstrated MLET's capability in producing superior models. However, there are many other model compression techniques that can also be used to squeeze more performance out of a fixed-size model. We develop the following experiments to compare and test the composition of MLET with these model size reduction techniques. Specifically, we include three commonly adopted techniques in our experiments – low rank SVD approximation, hashing, and post-training quantization. Experiments on the comparison between MLET and SVD compression are performed on three models – DCN, AutoInt, and xDeepFM. Experiments on the composition of MLET with quantization and hashing are performed on two models – DCN and DLRM. All experiments are done on the Avazu dataset.

A.4.1 LOW RANK SVD APPROXIMATION

As pointed out by Bhavana et al. (2019), the numerical rank of embedding tables can be much smaller than their embedding dimension, and hence, SVD factorization allows the matrix to be stored inexpensively, which means that the original embedding table then can be recovered with these low-dimensional factor matrices. Low rank SVD compression and MLET share a common feature of utilizing factorized matrices in deriving embedding tables and for both methods, the number of embedding parameters at inference can be greatly reduced as compared to training time.

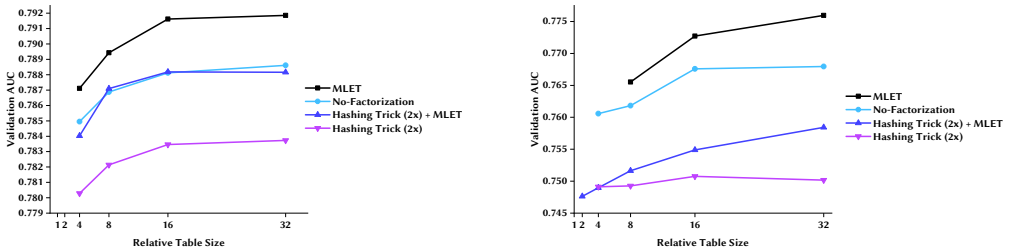
Table 4 summarizes a comparison of two methods applied on three models at different embedding sizes. For a MLET model, the effective embedding size is its embedding dimension. For a SVD-compressed model, it is the number of reserved ranks in the low rank SVD approximation of its embedding tables. For example, SVD d=64 with effective embedding size 32 means that the single-layer model with embedding dimension 64 is first trained and then its embedding tables (of shape $n \times 64$) are approximated by rank-32 SVD approximations. As shown in the table, for the three models we tested, SVD compression is most effective when the number of reserved ranks is at least half of the original embedding dimension. Further, MLET maintains its advantages over low rank SVD compression at all levels of embedding budgets.

Table 4: MLET vs. Low Rank SVD Approximation of embedding tables. With the same size of embeddings (at both training- and inference-time), MLET produces better models than low rank SVD compression.

Model	Configuration	Effective Embedding Size 4	8	16	32
DCN	SVD d=32	0.7783	0.7903	0.7914	-
	MLET k=32	0.7922	0.7932	0.7939	-
	SVD d=64	0.7659	0.7902	0.7923	0.7927
	MLET k=64	0.7924	0.7939	0.7945	0.7946
AutoInt	SVD d=32	0.7812	0.7916	0.7929	-
	MLET k=32	0.7920	0.7937	0.7938	-
	SVD d=64	0.7761	0.7910	0.7927	0.7930
	MLET k=64	0.7930	0.7942	0.7947	0.7948
xDeepFM	SVD d=32	0.7672	0.7818	0.7920	-
	MLET k=32	0.7933	0.7955	0.7972	-
	SVD d=64	0.7618	0.7783	0.7895	0.7962
	MLET k=64	0.7935	0.7957	0.7978	-

A.4.2 HASHING

The hashing trick as described in Attenberg et al. (2009) reduces table height by hashing the indices of table rows into a smaller index space. In this case, we choose the frequently used modulo operation as the hash function. We choose to hash the tables to half of their original size and only apply hashing on the two largest tables in the Avazu dataset (device_ip and device_id). These two tables jointly account for 99.7% of all embeddings. We do not apply hashing to other tables as hashing degrades model quality and the hashing of small tables has negligible memory savings.



(a) DCN (b) DLRM

Figure 7: Composition of MLET and Hashing Trick. Hashing leads to degradation. Quality is improved when combined with MLET.

A.4.3 QUANTIZATION

When post-training quantization Krishnamoorthi (2018) is applied, all values in the embedding tables are quantized to be 8-bit integers. We adopt a uniform symmetric quantizer where the scale factor of each table is determined by a grid search that minimizes the L2 error between original FP32 embeddings and the de-quantized embeddings. We select eight (INT8) as the bitwidth because near-zero degradation is observed in this case. Bitwidths smaller than eight lead to model quality degradation. Note that we only quantize embedding tables and not other parts of the model. Quantization that accelerates the computation (e.g., activations, weights of top MLP, bottom MLP in DLRM) is not our concern.

A.4.4 CALCULATION OF TABLE SIZE

Unlike our previous experiments, embedding dimension is not the sole factor that determines table size. In addition to embedding dimension, table size reduction due to quantization and hashing must be considered when calculating the table size. In our experiments, we use relative table size to measure the size of embedding tables. Relative table size normalizes the size of derived tables

with respect to a set of un-hashed tables with embedding dimension 1. For example, the orginal FP32 model with no hashing trick applied is of relative table size d where d is its embedding dimension. An INT8 quantized model with hashing trick applied is of size $\frac{d}{8}$ where the extra division by 8 comes from 2x reduction of table heights by hashing and 4x reduction by quantization to int8 representation. To make a fair comparison between the various combinations of techniques, their performance under different table size are plotted. We present their AUC-table size curves in Figure 7 and Figure 5.

A.4.5 DISCUSSION

As shown in the figures, MLET consistently improves model quality with all combinations of quantization and hashing. Hashing reduces the size at some quality loss; but, when combined with MLET, the quality-size trade-off produced by hashing is improved. We observe that the best quality-size trade-off is achieved by combining quantization and MLET.