A Universal Framework for Compressing Embeddings in CTR Prediction

Kefan Wang[1], Hao Wang[1](✉), Kenan Song[2], Wei Guo[2], Kai Cheng[1], Zhi Li[3], Yong Liu[2], Defu Lian[1], and Enhong Chen[1](✉)

[1] State Key Laboratory of Cognitive Intelligence, University of Science and Technology of China, Hefei, China
{wangkefan,ck2020}@mail.ustc.edu.cn
{wanghao3,liandefu,,cheneh}@ustc.edu.cn
[2] Huawei Singapore Research Center, Singapore
{songkenan,guowei67,liu.yong6}@huawei.com
[3] Shenzhen International Graduate School, Tsinghua University, Shenzhen, China
zhilizl@sz.tsinghua.edu.cn

Abstract. Accurate click-through rate (CTR) prediction is vital for online advertising and recommendation systems. Recent deep learning advancements have improved the ability to capture feature interactions and understand user interests. However, optimizing the embedding layer often remains overlooked. Embedding tables, which represent categorical and sequential features, can become excessively large, surpassing GPU memory limits and necessitating storage in CPU memory. This results in high memory consumption and increased latency due to frequent GPU-CPU data transfers. To tackle these challenges, we introduce a Model-agnostic Embedding Compression (MEC) framework that compresses embedding tables by quantizing pre-trained embeddings, without sacrificing recommendation quality. Our approach consists of two stages: first, we apply popularity-weighted regularization to balance code distribution between high- and low-frequency features. Then, we integrate a contrastive learning mechanism to ensure a uniform distribution of quantized codes, enhancing the distinctiveness of embeddings. Experiments on three datasets reveal that our method reduces memory usage by over 50x while maintaining or improving recommendation performance compared to existing models. The implementation code is accessible in our project repository https://github.com/USTC-StarTeam/MEC.

Keywords: CTR Prediction, Quantization, Lightweight, Memory-Efficient

1 Introduction

Click-through rate (CTR) prediction is a pivotal task in online advertising and recommender systems, aimed at estimating the likelihood of a user clicking on a given item. Traditional approaches such as Logistic Regression (LR) [15] and Factorization Machines (FM) [14] model feature interactions to capture linear and pairwise relationships, effectively laying the groundwork for CTR prediction. However, these methods often fall short of capturing complex high-order

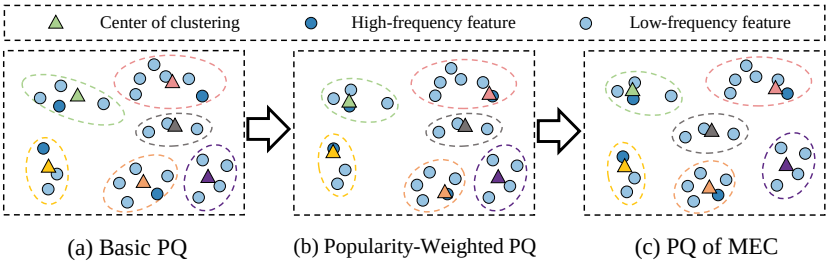

Fig. 1. Distribution of different quantization methods

interactions. Deep learning-based models have emerged to address these limitations, significantly improving the ability to capture intricate feature interactions [27,16,29]. Notable examples include DeepFM [8], which combines FM [14] with deep neural networks, and Product-based Neural Networks (PNN) [13], which explicitly model feature interactions through product layers. Further advancements like Adaptive Factorization Networks (AFN) [3] and Deep & Cross Networks (DCN) [22] introduce adaptive mechanisms and innovative cross-layer structures. Recently, Gated Deep & Cross Networks (GDCN) [21] have been proposed to offer greater flexibility and improved performance. Additionally, user modeling-based CTR models such as Deep Interest Network (DIN) [33] and Deep Interest Evolution Network (DIEN) [32] further enhance CTR prediction by capturing user interests and their evolution over time.

Despite advancements in CTR prediction, embedding tables can become exceedingly large, often reaching hundreds of gigabytes in industrial settings [17,30,9]. This substantial memory consumption frequently exceeds GPU capacity, causing data transfers between GPU and CPU during inference, which impacts performance and introduces latency in real-time systems [25]. To address this, memory-efficient techniques have been proposed, generally falling into hash-based and quantization-based methods. Hash-based methods, such as DHE [12], DoubleHash [19], and QRTrick [17], reduce memory usage by mapping features to fewer buckets using hash functions. However, they suffer from collision issues that degrade performance by confusing unrelated concepts.

In contrast, quantization methods like Product Quantization (PQ) [10] offer better performance with reduced memory consumption by decomposing high-dimensional embeddings into subspaces and quantizing them separately. Recent advancements include DPQ [1], AutoDPQ [7], and CCE[20]. Yet, these methods often overlook the quality of quantized representation distribution, leading to suboptimal results due to ***imbalances in code allocation*** and ***uneven distributions of code embeddings***. Frequency-independent quantization can be disrupted by low-frequency features, overshadowing high-frequency ones and weakening representation, as shown in Fig. 1(a). Moreover, directly considering frequency might overwhelm low-frequency features, as depicted in Fig. 1(b). Additionally, CTR tasks require evenly distributed embeddings to maintain distinctiveness, but traditional methods often lead to concentrated embeddings, compromising predictive performance [24].

To address these challenges, we propose a Model-agnostic Embedding Compression (MEC) Framework. Firstly, to tackle **imbalances in code allocation**, we introduce popularity-weighted regularization (PWR). PWR preserves high-frequency feature information, preventing it from being overshadowed by low-frequency features. It penalizes imbalances, ensuring low-frequency features aren't dominated by high-frequency ones. By adaptively assigning unique cluster centers to high-frequency features while promoting code-sharing among low-frequency features, PWR enhances representation capacity. Secondly, to mitigate **uneven distributions of code embeddings**, we integrate a contrastive learning mechanism. Using random code replacement to generate semi-synthetic negatives, the contrastive loss encourages uniform distribution of codes, ensuring distinctiveness. As shown in Fig. 1 (c), this method enhances recommendation performance by providing more accurate and diverse representations.

Our proposed framework operates in two stages, which enhances its generalization capabilities and allows for easy integration with any state-of-the-art quantization model or CTR model. We instantiated the MEC algorithm on three representative CTR models and conducted extensive experiments on three datasets. The results demonstrate that MEC can adaptively perform high-quality quantization for data with varying popularity levels, achieving comparable or even superior results to existing advanced CTR models while saving more than 50x the memory. In summary, the contributions of this work are as follows:

- We successfully address imbalances in code allocation by leveraging data popularity distribution through an adaptive two-stage quantization-based CTR prediction method, thereby enhancing recommendation performance while significantly reducing memory usage.
- To tackle the challenging issue of uneven distributions of code embeddings in PQ quantization for CTR tasks, we introduce PWR and contrastive learning methods to significantly enhance the quality of quantized codes, thereby improving recommendation performance.
- Extensive experiments on three real-world datasets validate that our MEC framework achieves over 50x memory optimization compared to models with conventional embedding layers, surpassing baselines in both recommendation performance and memory usage.

2 Related Work

2.1 Click-Through-Rate Prediction

CTR prediction focuses on estimating the likelihood of user clicks. Existing methods are mainly divided into feature interaction-based and user behavior modeling-based approaches. Feature interaction models, such as Wide&Deep [2] and DeepFM [8], utilize network architectures to capture complex interactions. For example, GDCN [21] employs dual tower networks to model both explicit and implicit interactions.

User behavior modeling aims to extract personal preferences from historical interactions [16,31]. Techniques have evolved from attention networks [34] to

efficient Transformers. DIN [34] uses attention to assign scores to user behaviors, capturing diverse interests.

2.2 Memory-Efficient Recommendation

Memory-efficient techniques, including hashing and quantization, are crucial for resource-constrained systems. Hashing methods convert embeddings into compact codes, improving memory efficiency but risking noise from unrelated mappings [26,12]. Quantization reduces memory usage by representing embeddings through codebooks, achieving higher compression rates [1]. Methods like AutoDPQ adaptively determine codebook sizes, while CCE combines PQ and hashing for dynamic updates [7].

However, these approaches often overlook the distribution quality of quantized embeddings. Our adaptive two-stage training framework optimizes embedding distributions post-quantization, enhancing representation quality without significant computational overhead. This approach complements existing methods and offers potential for future research advancements.

3 Preliminary

The CTR prediction task is formulated as predicting the probability $\hat{y} = P(\text{click}|\mathbf{x})$ of a user clicking based on input features $\mathbf{x}$, which is crucial for online recommendation platforms. The input features can be divided into two types: categorical features and numerical features. For categorical features, we start from an embedding layer that transforms the raw features of a training sample into embedding vectors. For a feature A with vocabulary size v_A, we first encode the categorical information into a one-hot or multi-hot vector $\mathbf{x}_{\mathrm{A}} \in \{0,1\}^{v_A}$, then an embedding lookup operation is conducted to transform the high-dimensional feature vector into a low-dimensional embedding:

$$\mathbf{e}_{\mathrm{A}} = \mathbf{E}_{\mathrm{A}}\mathbf{x}_{\mathrm{A}}, \tag{1}$$

where $E_A \in R^{d_A \times v_A}$ is the embedding table of feature A. Noticed that for sequential multi-hot categorical features here, we apply the average pooling here to transform it into a single vector. For numerical features, each one is processed through a DNN layer to map the raw value into a vector. Let $\mathbf{x}_a$ be the a-th continuous feature, transformed as follows:

$$\mathbf{e}_a = \mathrm{DNN}_a(\mathbf{x}_a), \tag{2}$$

where DNN_a is the layer specific to the a-th feature. By concatenating all features, we represent an instance as:

$$\mathbf{e} = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_N], \tag{3}$$

with N being the total number of features. The combined embedding vectors are then input into the CTR prediction model to capture complex interactions and predict click probabilities.

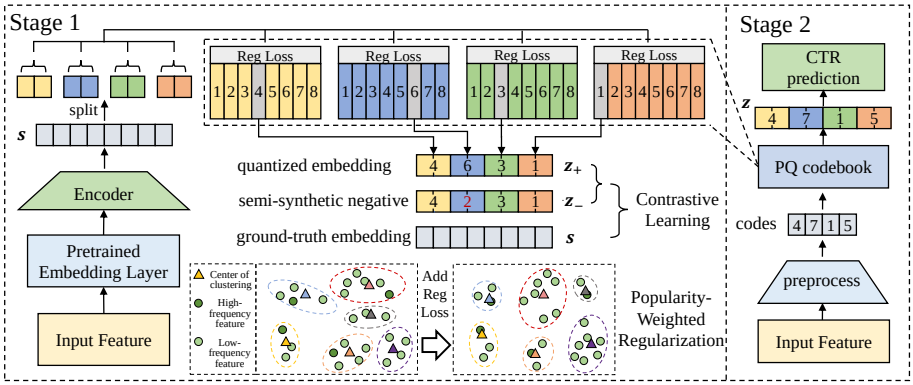

Fig. 2. Overview of our MEC framework. The framework consists of two stages: pre-training and downstream task training. In the first stage (left), a PQ codebook is learned by combining existing embeddings. In the second stage (right), the input features are quantized based on the PQ codebook and used to train a CTR model for the downstream task. During online inference, the pre-quantized features are fed into the downstream task to achieve memory-efficient CTR prediction.

4 Methodology

Contemporary CTR prediction methods face challenges with large embedding spaces, leading to increased storage demands and reduced inference efficiency. Traditional quantization techniques often overlook high-frequency features and fail to evenly distribute embeddings, degrading model performance. Our approach compresses embedding spaces while maintaining accuracy, as detailed in Section 4.1 for the MEC framework, Section 4.2 for popularity-weighted regularization (PWR), and Section 4.3 for contrastive learning to address uneven representation distribution.

4.1 Overview

Existing models typically rely on dense embeddings, resulting in large tables and slower inference. While quantization is a common solution, it must effectively retain feature richness for accurate CTR prediction. Current models are not fully optimized for this, adversely affecting recommendation performance. As shown in Fig. 2, our innovative two-stage framework decouples quantization from inference, allowing easy adaptation and efficient updates.

In the first stage, an auxiliary CTR model pre-trains embeddings. The input features $\mathbf{x}$ are transformed into embedding vectors $\mathbf{e}$ via an embedding function F_{Emb}, then encoded and partitioned for quantization:

$$\mathbf{e} = F_{\text{Emb}}(\mathbf{x}), \quad \mathbf{s} = \text{Encoder}(\mathbf{e}). \tag{4}$$

Here, $\mathbf{x}$ represents the input features, $\mathbf{e}$ is the resulting embedding vector, and $\mathbf{s}$ is the encoded vector prepared for quantization. The vector $\mathbf{s}$ is split into M sub-vectors $\{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_M\}$, each of dimension d/M, where d is the dimension of the encoded vector. For Product Quantization (PQ), each sub-vector $\mathbf{s}_i$ is

quantized using a codebook $\mathbf{C}_i$, initialized with size K. The closest codeword is identified by minimizing the Euclidean distance:

$$j_i = \arg\min_j \|\mathbf{s}_i - \mathbf{c}_{i,j}\|_2, \quad \mathbf{q}_i = \mathbf{c}_{i,j_i}, \tag{5}$$

where $\mathbf{c}_{i,j}$ is the j-th codeword in the codebook $\mathbf{C}_i$. The quantized vector $\mathbf{q}$ is formed by concatenating all $\mathbf{q}_i$:

$$\mathbf{q} = [\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_M]. \tag{6}$$

To ensure $\mathbf{q}$ closely approximates $\mathbf{s}$, we minimize the reconstruction loss:

$$\mathcal{L}_{\text{recon}} = \|\mathbf{s} - \mathbf{q}\|_2^2. \tag{7}$$

After convergence, the learned codebook $\mathbf{C}$ and quantization method ϕ are retained for CTR prediction. Enhancements include Popularity-weighted Regularization and Contrastive Product Quantization, which improve the distribution of quantized features.

Stage 2: Memory-Efficient Embedding for CTR Prediction Using the saved quantization method ϕ, input features $\mathbf{x}$ are transformed into quantized codes $\mathbf{c}$, retrieving embeddings for CTR model training and inference:

$$\mathbf{c} = \phi(\mathbf{x}), \quad \mathbf{e} = \mathbf{E}_\phi \mathbf{c}, \tag{8}$$

where $\mathbf{E}_\phi$ is initialized from the retained codebook $\mathbf{C}$. The CTR model predicts click-through rate probability using the embedding vector $\mathbf{e}$:

$$\hat{y} = \psi(\mathbf{e}), \tag{9}$$

where ψ represents the feature interaction operations in the downstream CTR model. The prediction is optimized using a binary classification loss:

$$\mathcal{L}_{\text{binary}} = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})), \tag{10}$$

where y is the true label. By compressing the embedding table with the PQ codebook, we address memory consumption and enhance model inference efficiency and generalization.

4.2 Popularity-Weighted Regularization

In quantized CTR recommendation tasks, models often focus on memory efficiency, which may lead to imbalanced code allocation and overshadow critical high-frequency features. Methods ignoring frequency fail to account for varying feature importance, causing representation issues. However, introducing feature frequency directly usually overwhelms low-frequency features, leading to degraded performance.

To address these issues, we propose a popularity-weighted regularization method. We weigh features based on popularity, using a logarithmic transformation to smooth frequency disparities and calculate weights more effectively. The formula is as follows:

$$r_j = \lfloor \log_2(n_j) \rfloor \tag{11}$$

where n_j denotes the frequency count of feature j. This allows the quantization model to adaptively distinguish features of different popularities. However, this also causes the cluster centers to shift too much towards high-frequency features, making it difficult to effectively model low-frequency features. This is because, during the weighting process, although the weights of high-frequency features are smoothed, they still occupy a large proportion, thereby affecting the clustering effect of low-frequency features.

To enhance the learning of low-frequency features, it is essential to incorporate regularization to balance the influences of high-frequency and low-frequency features. We propose a loss function that integrates a regularization term to penalize imbalances in code allocation. Given the efficacy of entropy-based tests in detecting uniformity [6], we employ a modified entropy-based metric as the loss function to evaluate the uniformity of code distribution. This metric is selected for its capacity to measure the deviation from a uniform distribution, thereby facilitating fair code assignments. The regularization loss is calculated as follows:

$$\mathcal{L}_{\text{reg}} = \exp\left(-\sum_{i=1}^{K} p_i \log(p_i + \epsilon)\right), p_i = \frac{\sum_{j \in S_i} r_j}{\sum_{j \in S} r_j} \tag{12}$$

where K is the number of codes (embeddings), ϵ is a very small number (e.g., $1e-10$) to prevent the logarithm from being zero, p_i is the probability of the i-th code, S_i is the set of features assigned to code i, S is the set of all features, and r_j represents the popularity weighting of feature j. This calculation ensures that p_i reflects the proportion of total feature frequency assigned to code i. Through this popularity-weighted regularization method, we can better balance the modeling of high-frequency and low-frequency features, thereby improving the overall performance.

In summary, the popularity-weighted regularization method weights and smooths samples, allowing the quantization model to better handle features with different frequencies, avoiding excessive bias towards high-frequency features and neglect of low-frequency features. This method not only improves the accuracy of the model but also enhances its robustness and generalization ability.

4.3 Contrastive Product Quantization

While PQ embeddings and popularity-weighted regularization address code allocation imbalances, prior research [24] highlights that quantized embeddings often suffer from uneven distributions, leading to homogenization and reduced diversity. This lack of diversity hampers the model's ability to differentiate features, compromising CTR prediction accuracy. To address this, we integrate contrastive learning to enhance embedding diversity, ensuring balanced code allocations and improving recommendation performance.

Table 1. Dataset statistics

Dataset	#Interactions	#Feature Fields	#Features
Criteo	45,840,617	39	1,086,810
Avazu	40,428,968	22	2,018,012

In contrastive learning, informative negative samples are crucial. We synthesize enhanced feature indices as negatives. However, fully synthetic indices may be too distant from real features. Thus, we create semi-synthetic codes based on real item codes to serve as effective hard negatives.

Given a real feature code $\mathbf{c} = [j_1, j_2, \ldots, j_M]$, we randomly replace one index with a probability $\rho \in (0, 1)$ while keeping all others unchanged. This effective technique ensures the semi-synthetic codes resemble real features but still offer enough variability to act as challenging hard negative samples. The semi-synthetic code $\mathbf{z}$ is generated as follows:

$$G(\mathbf{z_i}) = \begin{cases} \text{Uni}(\{1, \ldots, K\}), & \text{if } X = 1 \\ \mathbf{c_i}, & \text{if } X = 0 \end{cases} \tag{13}$$

where $X \sim \text{Bernoulli}(\rho)$, and $\text{Uni}(\cdot)$ uniformly samples item codes from the input set. This uniform sampling method guarantees that the code distribution of semi-synthetic indices is similar to that of real indices. The embedding of the real feature code $\mathbf{c}'$ and the corresponding semi-synthetic hard negative sample instance $\mathbf{z}'$ is given by:

$$\mathbf{c}' = \text{Emb-Pool}(\mathbf{c}), \mathbf{z}' = \text{Emb-Pool}(\mathbf{z}), \tag{14}$$

where $\text{Emb-Pool}(\cdot)$ denotes the embedding lookup and aggregation. We use concatenation for aggregation here for simplicity.

To incorporate contrastive learning into our framework, we define a contrastive loss function that encourages the model to distinguish between real item codes and their semi-synthetic hard negative samples. The contrastive loss $\mathcal{L}_{\text{con}}$ is defined as follows:

$$\mathcal{L}_{\text{con}} = -\frac{1}{N} \sum_{i=1}^{N} \log \frac{\exp(\text{sim}(\mathbf{s}, \mathbf{c}'))}{\exp(\text{sim}(\mathbf{s}, \mathbf{c}')) + \sum_{j=1}^{K} \exp(\text{sim}(\mathbf{s}, \mathbf{z}'))}, \tag{15}$$

where N is the number of features, $\text{sim}(\cdot, \cdot)$ denotes a similarity function (e.g., cosine similarity), $\mathbf{s}$ is the encoded vector, $\mathbf{c}'$ is the quantized embedding vector, and $\mathbf{z}'$ are the quantized embeddings of K semi-synthetic hard negative samples. By minimizing this contrastive loss, the model learns to bring the embeddings of the encoded vector and quantized vector closer together while pushing the semi-synthetic hard negative samples further apart. This results in a more balanced and diverse code embedding distribution.

5 EXPERIMENTS

5.1 Experiment Setup

Datasets. We evaluated our model using three datasets: two public and one private industrial dataset.

- Criteo: A standard dataset from Kaggle[4] with one week of user click data for CTR prediction. It includes 45 million samples and 39 features (13 continuous, 26 categorical), useful for model evaluation.
- Avazu: Another Kaggle dataset[5], used for CTR prediction with 11 days of user click data. It contains about 40 million samples and 24 features, serving as a solid base for testing algorithms.
- Industrial: A private dataset from the Huawei ad platform with over 400 million user impressions. It includes hundreds of features, both categorical and numerical, with 32 features having vocab sizes over 100,000, and the largest reaching six million.

We followed AFN [3] preprocessing, splitting Criteo and Avazu datasets into training, validation, and test sets in a 7:2:1 ratio by time order. The Industrial dataset was split into train/val/test sets in a 6:1:1 ratio. Table 1 provides detailed statistics for the public datasets.

Baseline Models To demonstrate the effectiveness of the proposed model, we select some representative CTR models for comparison. We also compare our proposed model with some hashing and quantization-based methods to validate the superiority of our model. The details are listed as follows:

Representative CTR Models:

- **LR** [15]: Logistic Regression is a linear model using a logistic function for binary variables. It is often a baseline in recommendation systems due to its simplicity and interpretability.
- **FM** [14]: Factorization Machine models pairwise interactions between features efficiently, making it suitable for sparse data.
- **AFM** [28]: Attentional Factorization Machine enhances FM by using attention to model feature interaction importance.
- **DeepFM** [8]: Combines FM for low-order and DNN for high-order interactions, improving recommendation accuracy.
- **PNN** [13]: Product-based Neural Network uses product operations between feature embeddings to model feature interactions.
- **DCNv2** [23]: Deep & Cross Network v2 captures explicit and implicit feature interactions by stacking cross and deep layers.
- **AutoInt**[18]: Utilizes self-attention mechanisms to automatically learn feature interactions without manual feature engineering.
- **AFN** [3]: Adaptive Factorization Network dynamically learns the importance of feature interactions using a neural network.

[4] https://www.kaggle.com/c/criteo-display-ad-challenge/
[5] https://www.kaggle.com/c/avazu-ctr-prediction/

Table 2. Overall performance comparison between the baselines and MEC instantiated on two competitive CTR prediction models across two datasets. The results table includes AUC, LogLoss, and memory usage for each model on the respective datasets. Bold values indicate a statistically significant level p-value<0.05 comparing MEC with the base model's performance in terms of AUC, LogLoss, and memory usage.

	Criteo			Avazu		
Model	**AUC**	**Logloss**	**Params**	**AUC**	**Logloss**	**Params**
LR	0.7879	0.4615	-	0.7514	0.3737	-
FM	0.7945	0.4564	166.90MB	0.7508	0.3755	103.91MB
AFM	0.8045	0.4475	166.92MB	0.7541	0.3760	103.91MB
AFN	0.8055	0.4471	204.94MB	0.7502	0.3764	141.86MB
SAM	0.8060	0.4490	167.13MB	0.7528	0.3750	103.97MB
DeepFM	0.8069	0.4447	169.38MB	0.7533	0.3744	105.60MB
AutoInt	0.8081	0.4433	169.96MB	0.7515	0.3753	105.91MB
DCNv2	0.8099	0.4415	199.67MB	0.7520	0.3746	116.37MB
GDCN	0.8098	0.4417	172.86MB	0.7507	0.3761	107.81MB
DHE$_{GDCN}$	0.8087	0.4427	8.089MB	0.7505	0.3758	5.707MB
xLightFM$_{GDCN}$	0.8089	0.4422	7.659MB	0.7504	0.3760	5.097MB
MEC$_{GDCN}$	**0.8102**	**0.4415**	**7.649MB**	**0.7538**	**0.3728**	**4.567MB**
PNN	0.8099	0.4418	168.80MB	0.7531	0.3750	105.06MB
DHE$_{PNN}$	0.8088	0.4429	4.028MB	0.7517	0.3763	2.957MB
xLightFM$_{PNN}$	0.8092	0.4423	3.598MB	0.7524	0.3758	2.347MB
MEC$_{PNN}$	**0.8105**	**0.4412**	**3.588MB**	**0.7556**	**0.3736**	**1.817MB**

– **SAM** [4]: Self-Attentive Model leverages self-attention to capture complex feature interactions effectively.
– **GDCN** [21]: Gate-based Deep Cross Network uses gating mechanisms to model the complex relationships between users and items, enhancing collaborative filtering signals.

Hashing and Quantization Models:

– **DHE** [12]: Deep Hash Embedding uses hashing techniques for dimensionality reduction, improving memory and time efficiency.
– **xLightFM** [11]: An extremely memory-efficient Factorization Machine that uses codebooks for embedding composition and adapts codebook size with neural architecture search.

Evaluation Metrics We evaluate the algorithms using AUC and Logloss metrics to ensure a comprehensive performance assessment. AUC (Area Under the ROC Curve) evaluates the model's capability to rank positive instances above negatives, with higher values indicating superior discrimination ability. Logloss, on the other hand, measures the accuracy of predicted probabilities in relation to actual labels, with lower values indicating better model calibration and precision. Our method achieves over 90% memory savings while maintaining performance on par with baseline models. This efficiency, coupled with competitive accuracy, underscores the method's practical value, especially in environments where memory constraints are a concern.

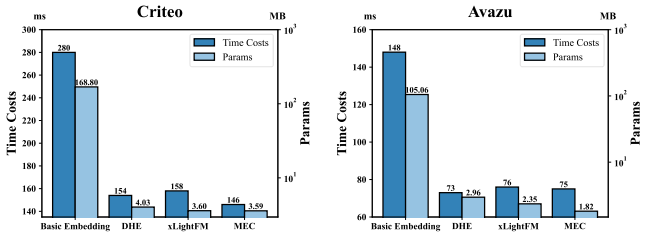

Fig. 3. Time efficiency performance

Parameter Settings All models were implemented using the FuxiCTR library[6]. We standardized the embedding dimension to 40 and batch size to 10,000. The learning rate was chosen from {1e-1, 1e-2, 1e-3, 1e-4}, with L_2 regularization from {0, 1e-1, 1e-2, 1e-3, 1e-4, 1e-5}, and dropout ratios from 0 to 0.9. Models were trained using the Adam optimizer. Codebook sizes were {256, 512, 1024, 2048} and PQ layers {2, 4, 8}. Experiments were repeated five times to ensure reliability, reporting average results. Memory usage was based on full model size, and results are presented with optimal parameters. We used the pre-trained embedding layer of DeepFM as initialization before quantization.

5.2 Performance Comparison

In this section, we compare MEC with baseline models in terms of both CTR performance and memory usage, and the results are presented in Table 2. We also conduct Wilcoxon signed rank tests [5] to evaluate the statistical significance of MEC with the base model. We have the following observations:

(1) Embedding tables significantly impact model parameters. In traditional CTR models, most parameters are concentrated in the embedding layer. Despite variations in model structure and complexity, the overall parameter size remains similar. Table 2 shows that optimizing embedding tables can reduce model parameters by over 90%, highlighting the potential for quantization and compression to enhance efficiency and performance.

(2) Hashing and quantization models have benefits but limitations. Rows 9 to 16 in Table 2 indicate that methods like DHE [12] and xLightFM [11] reduce parameters significantly but degrade performance. This is due to their focus on compression without addressing data distribution imbalances caused by traditional quantization. These methods also require end-to-end training and lack portability. Our approach overcomes these issues by rebalancing quantized data distribution, achieving better performance.

(3) Embedding lookup time correlates with embedding table size. Fig. 3 shows that embedding lookup time is reduced by about 50% with quantization due to smaller vocabulary sizes, indicating a strong link between inference time and memory size. Training latency analysis reveals that PQ quantization modifications add negligible latency, confirming the method's practicality. Full results are in Section 5.4. MEC's parameter compression and high-quality quantization enhance both inference efficiency and model performance.

[6] https://fuxictr.github.io/

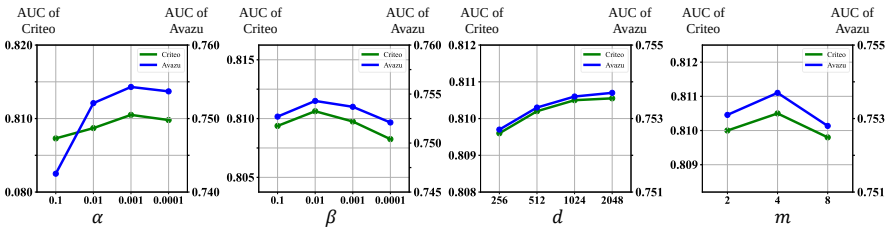

Fig. 4. Hyper-Parameter Performance

Table 3. Performance on industrial dataset

Metric	PRAUC↑	PCOC↑	AUC↑	LogLoss↓	Params↓
baseline	0.91913	0.96561	0.84141	0.40721	3142.3MB
MEC	**0.91918**	**0.96673**	**0.84143**	**0.40683**	**7.014MB**

(4) MEC excels across all datasets. Table 2 shows that MEC_{GDCN} outperforms GDCN by 0.4% on Avazu and 0.07% on Criteo, with fewer parameters. As a pluggable model, MEC improves performance by up to 0.33% and 0.41% over PNN and GDCN, especially on Avazu due to its long-tailed distribution. This distribution often limits conventional models, but our method's popularity-based regularization enhances feature embedding quality.

5.3 Industrial Dataset Performance

Table 3 clearly demonstrates that MEC achieves over 99.7% reduction in embedding layer memory usage through efficient Product Quantization. The integration of popularity-weighted regularization and contrastive learning effectively addresses typical challenges like code imbalance and uneven quantization distribution. Despite significant quantization, our method consistently maintains or even surpasses the performance of the baseline model.

5.4 In-depth Analysis

Analysis of Hyper-Parameter. As shown in Fig. 4, we analyzed key hyper-parameters in our MEC framework: the regularization loss coefficient (α), contrastive loss coefficient (β), embedding dimension (d), and number of embedding layers (m). Optimal values were $\alpha = 0.001$, $\beta = 0.01$, $d = 2048$, and $m = 4$. Deviations from these values caused issues like underfitting, overfitting, information loss, or suboptimal relationship modeling.

Ablation Study of MEC To thoroughly assess the contributions of different components in MEC, we evaluated three distinct variants on PNN [13]: (1) without contrastive learning (w/o cons.); (2) without popularity-weighted regularization (w/o reg.); (3) using basic Product Quantization (PQ); and (4) frequency-based PQ (freq. PQ). The results, as shown in Table 4, were obtained using the Criteo and Avazu datasets.

Directly applying PQ reduces performance by weakening feature representation and CTR prediction accuracy. Simply including frequency information exacerbates this by allowing high-frequency features to dominate, poorly repre-

Table 4. Ablation study of MEC

	Criteo		Avazu	
Variants	**AUC**	**Logloss**	**AUC**	**Logloss**
MEC$_{PNN}$	**0.8105**	**0.4412**	**0.7556**	**0.3736**
w/o cons.	0.8096	0.4418	0.7542	0.3739
w/o reg.	0.8092	0.4421	0.7540	0.3741
freq. PQ	0.8077	0.4451	0.7511	0.3755
basic PQ	0.8084	0.4437	0.7528	0.3746
PNN	0.8099	0.4418	0.7531	0.3750

Table 5. Latency of training stage

Metric	**Criteo**	**Avazu**
w/o PQ	**24.07s**	**13.96s**
basic PQ	24.92s	14.67s
w/o cons.	24.95s	14.68s
w/o reg.	25.50s	15.29s
MEC$_{PNN}$	25.56s	15.31s

senting low-frequency features. Regularization mitigates this imbalance by preventing domination and better representing low-frequency features. Contrastive learning enhances representation separation and code distribution. Together, regularization and contrastive learning balance feature frequencies and improve quantized representation quality.

Analysis of Training Latency In this section, we analyze the training latency, given the computational constraints in real-world scenarios where training time needs careful consideration. Table 5 presents the results based on the Criteo and Avazu datasets, averaged over 10 runs. The PQ component adds negligible training time compared to the main CTR model. Popularity-weighted regularization does not increase time complexity, and while contrastive learning slightly increases training time, it remains manageable. Overall, our approach improves quantized embedding quality without significantly impacting training latency.

Analysis of Quantization Methods We evaluated quantization methods for MEC, comparing Product Quantization (PQ) with Additive Quantization (AQ) and Residual Quantization (RQ). Table 6 shows that PQ outperforms the others. RQ suffers from code correlation, destabilizing contrastive learning, while AQ introduces redundancy and noise. PQ's independent sub-codebooks maintain stability and memory efficiency, making it the best choice for MEC.

Analysis of Pre-train Models We evaluated the impact of different pre-trained models (FM [14], DeepFM [8], DCNv2 [23]) on our framework for embedding generation and quantization, using PNN for downstream CTR prediction. Table 7 shows minor performance variations across models, highlighting our framework's robust generalizability. This enables the use of simpler models in industrial applications without significant performance loss, allowing efficient deployment in various scenarios.

Table 6. Comparison of quantization methods

	Criteo		Avazu	
Pretrain model	**AUC**	**Logloss**	**AUC**	**Logloss**
AQ+GDCN	0.8025	0.4507	0.7407	0.3871
RQ+GDCN	0.8094	0.4421	0.7513	0.3752
PQ+GDCN	**0.8102**	**0.4415**	**0.7538**	**0.3728**
AQ+PNN	0.8038	0.4480	0.7451	0.3813
RQ+PNN	0.8099	0.4418	0.7528	0.3749
PQ+PNN	**0.8105**	**0.4412**	**0.7556**	**0.3737**

Table 7. Performance on multiple pretrain models

	Criteo		Avazu	
Pretrain model	**AUC**	**Logloss**	**AUC**	**Logloss**
FM	0.8104	0.4413	0.7551	0.3741
DeepFM	**0.8105**	**0.4412**	**0.7556**	0.3737
DCNv2	0.8104	0.4412	0.7542	**0.3736**

6 Conclusion

This paper tackles the memory consumption challenge in CTR prediction models caused by large embedding tables. We proposed a Model-agnostic Embedding Compression (MEC) framework, which combines popularity-weighted regularization (PWR) and contrastive learning to compress embeddings while maintaining high recommendation performance. Experiments on three real-world datasets demonstrate that MEC reduces memory usage by over 50x while achieving comparable or superior performance to state-of-the-art models. Our findings highlight MEC's potential for efficient and scalable CTR prediction.

Acknowledgments. This work was supported by the National Natural Science Foundation of China (U23A20319, 62472394, 62441239, 62202443) as well as the Anhui Province Science and Technology Innovation Project (202423k09020011), Anhui Provincial Science and Technology Major Project (No. 2023z020006), and the China Postdoctoral Science Foundation (No. 2024T170497).

References

1. Chen, T., Li, L., Sun, Y.: Differentiable product quantization for end-to-end embedding compression. In: International Conference on Machine Learning. pp. 1617–1626. PMLR (2020)
2. Cheng, H.T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., et al.: Wide & deep learning for recommender systems. In: Proceedings of the 1st workshop on deep learning for recommender systems. pp. 7–10 (2016)
3. Cheng, W., Shen, Y., Huang, L.: Adaptive factorization network: Learning adaptive-order feature interactions. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 3609–3616 (2020)
4. Cheng, Y., Xue, Y.: Looking at ctr prediction again: Is attention all you need? In: Proceedings of the 44th International ACM SIGIR Conference on Research

and Development in Information Retrieval. SIGIR '21, ACM (Jul 2021). https://doi.org/10.1145/3404835.3462936, http://dx.doi.org/10.1145/3404835.3462936
5. Derrick, B., White, P.: Comparing two samples from an individual likert question. International Journal of Mathematics and Statistics **18**(3), 1–13 (2017)
6. Dudewicz, E.J., Van Der Meulen, E.C.: Entropy-based tests of uniformity. Journal of the American Statistical Association **76**(376), 967–974 (1981)
7. Gan, X., Wang, Y., Zhao, X., Wang, W., Wang, Y., Liu, Z.: Autodpq: Automated differentiable product quantization for embedding compression. In: Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 1833–1837 (2023)
8. Guo, H., Tang, R., Ye, Y., Li, Z., He, X.: Deepfm: a factorization-machine based neural network for ctr prediction. arXiv preprint arXiv:1703.04247 (2017)
9. Guo, W., Wang, H., Zhang, L., Chin, J.Y., Liu, Z., Cheng, K., Pan, Q., Lee, Y.Q., Xue, W., Shen, T., et al.: Scaling new frontiers: Insights into large recommendation models. arXiv preprint arXiv:2412.00714 (2024)
10. Jegou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE transactions on pattern analysis and machine intelligence **33**(1), 117–128 (2010)
11. Jiang, G., Wang, H., Chen, J., Wang, H., Lian, D., Chen, E.: xlightfm: Extremely memory-efficient factorization machine. Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (2021), https://api.semanticscholar.org/CorpusID:235792382
12. Kang, W.C., Cheng, D.Z., Yao, T., Yi, X., Chen, T., Hong, L., Chi, E.H.: Learning to embed categorical features without embedding tables for recommendation. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. pp. 840–850 (2021)
13. Qu, Y., Cai, H., Ren, K., Zhang, W., Yu, Y., Wen, Y., Wang, J.: Product-based neural networks for user response prediction. In: 2016 IEEE 16th international conference on data mining (ICDM). pp. 1149–1154. IEEE (2016)
14. Rendle, S.: Factorization machines. In: 2010 IEEE International conference on data mining. pp. 995–1000. IEEE (2010)
15. Richardson, M., Dominowska, E., Ragno, R.: Predicting clicks: estimating the click-through rate for new ads. In: Proceedings of the 16th international conference on World Wide Web. pp. 521–530 (2007)
16. Shen, T., Wang, H., Wu, C., Chin, J.Y., Guo, W., Liu, Y., Guo, H., Lian, D., Tang, R., Chen, E.: Predictive models in sequential recommendations: Bridging performance laws with data quality insights. arXiv preprint arXiv:2412.00430 (2024)
17. Shi, H.J.M., Mudigere, D., Naumov, M., Yang, J.: Compositional embeddings using complementary partitions for memory-efficient recommendation systems. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 165–175 (2020)
18. Song, W., Shi, C., Xiao, Z., Duan, Z., Xu, Y., Zhang, M., Tang, J.: Autoint: Automatic feature interaction learning via self-attentive neural networks. In: Proceedings of the 28th ACM international conference on information and knowledge management. pp. 1161–1170 (2019)
19. Tito Svenstrup, D., Hansen, J., Winther, O.: Hash embeddings for efficient word representations. Advances in neural information processing systems **30** (2017)
20. Tsang, H., Ahle, T.: Clustering the sketch: dynamic compression for embedding tables. Advances in Neural Information Processing Systems **36**, 72155–72180 (2023)

21. Wang, F., Gu, H., Li, D., Lu, T., Zhang, P., Gu, N.: Towards deeper, lighter and interpretable cross network for ctr prediction. In: Proceedings of the 32nd ACM International Conference on Information and Knowledge Management. pp. 2523–2533 (2023)
22. Wang, R., Fu, B., Fu, G., Wang, M.: Deep & cross network for ad click predictions (2017), https://arxiv.org/abs/1708.05123
23. Wang, R., Shivanna, R., Cheng, D., Jain, S., Lin, D., Hong, L., Chi, E.: Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In: Proceedings of the web conference 2021. pp. 1785–1797 (2021)
24. Wang, W., Bao, H., Lin, X., Zhang, J., Li, Y., Feng, F., Ng, S.K., Chua, T.S.: Learnable tokenizer for llm-based generative recommendation (2024), https://arxiv.org/abs/2405.07314
25. Wang, Z., Wei, Y., Lee, M., Langer, M., Yu, F., Liu, J., Liu, S., Abel, D.G., Guo, X., Dong, J., et al.: Merlin hugectr: Gpu-accelerated recommender system training and inference. In: Proceedings of the 16th ACM Conference on Recommender Systems. pp. 534–537 (2022)
26. Weinberger, K., Dasgupta, A., Langford, J., Smola, A., Attenberg, J.: Feature hashing for large scale multitask learning. In: Proceedings of the 26th annual international conference on machine learning. pp. 1113–1120 (2009)
27. Wu, L., Zheng, Z., Qiu, Z., Wang, H., Gu, H., Shen, T., Qin, C., Zhu, C., Zhu, H., Liu, Q., et al.: A survey on large language models for recommendation. World Wide Web **27**(5), 60 (2024)
28. Xiao, J., Ye, H., He, X., Zhang, H., Wu, F., Chua, T.S.: Attentional factorization machines: Learning the weight of feature interactions via attention networks (2017), https://arxiv.org/abs/1708.04617
29. Xu, X., Wang, H., Guo, W., Zhang, L., Yang, W., Yu, R., Liu, Y., Lian, D., Chen, E.: Multi-granularity interest retrieval and refinement network for long-term user behavior modeling in ctr prediction. arXiv preprint arXiv:2411.15005 (2024)
30. Yan, B., Wang, P., Liu, J., Lin, W., Lee, K.C., Xu, J., Zheng, B.: Binary code based hash embedding for web-scale applications. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management. pp. 3563–3567 (2021)
31. Yin, M., Wang, H., Guo, W., Liu, Y., Zhang, S., Zhao, S., Lian, D., Chen, E.: Dataset regeneration for sequential recommendation. In: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. pp. 3954–3965 (2024)
32. Zhou, G., Mou, N., Fan, Y., Pi, Q., Bian, W., Zhou, C., Zhu, X., Gai, K.: Deep interest evolution network for click-through rate prediction. In: Proceedings of the AAAI conference on artificial intelligence. vol. 33, pp. 5941–5948 (2019)
33. Zhou, G., Song, C., Zhu, X., Fan, Y., Zhu, H., Ma, X., Yan, Y., Jin, J., Li, H., Gai, K.: Deep interest network for click-through rate prediction (2018), https://arxiv.org/abs/1706.06978
34. Zhou, G., Zhu, X., Song, C., Fan, Y., Zhu, H., Ma, X., Yan, Y., Jin, J., Li, H., Gai, K.: Deep interest network for click-through rate prediction. In: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining. pp. 1059–1068 (2018)