Data Science and Engineering (2026) 11:230–247
https://doi.org/10.1007/s41019-025-00327-5

RESEARCH PAPERS

Sparse Gradient Training for Recommender Systems

Yunke Qu[1] · Liang Qu[2] · Tong Chen[1] · Xiangyu Zhao[3] · Jianxin Li[2] · Hongzhi Yin[1]

Received: 9 June 2025 / Revised: 5 October 2025 / Accepted: 10 November 2025 / Published online: 25 January 2026

Abstract

Recommender systems are widely applied in numerous online platforms such as shopping and social media platforms. They typically utilize large embedding tables that map users and items to dense vectors of uniform sizes. As the number of users and items continues to grow, this design leads to significant memory consumption and computational inefficiencies. This challenge is particularly pronounced in scenarios such as federated learning, where model parameters are updated locally on edge devices with limited computational resources before being transmitted to a central server for aggregation. Numerous approaches have been proposed to address this issue, among which embedding pruning methods have emerged as a compelling solution. Compared to parameter-sharing and variable-size embedding techniques, embedding pruning methods offer lower training costs and leverage sparse embeddings for improved efficiency. Notably, embedding pruning methods based on the Dynamic Sparse Training (DST) paradigm maintain consistent sparsity throughout training and provide a controllable memory budget, establishing them as state-of-the-art lightweight embedding solutions for resource-constrained environments. However, embedding pruning methods are not without limitations. First, despite the use of sparse embeddings during forward passes, dense gradients are still computed in backward passes, introducing inefficiencies. Second, DST's weight exploration mechanism tends to prioritize users or items from the most recent batch, reactivating pruned parameters that do not necessarily enhance overall performance. In this work, we introduce SparseRec, a lightweight embedding method designed to overcome these obstacles. SparseRec accumulates gradients to better identify inactive parameters that, when reactivated, contribute more meaningfully to model performance. Additionally, SparseRec avoids dense gradient computation during backpropagation by selectively sampling key vectors. Gradients are calculated only for parameters in this subset, ensuring sparsity throughout both forward and backward passes. Experiments on three benchmark datasets show that SparseRec achieves up to 11.79% performance gains across three base recommenders and multiple density configurations, highlighting its effectiveness in optimizing memory-constrained recommendation systems.

Keywords Recommender Systems · Model Pruning · Collaborative Filtering

✉ Hongzhi Yin
h.yin1@uq.edu.au

Yunke Qu
yunke.qu@uq.net.au

Liang Qu
l.qu@ecu.edu.au

Tong Chen
tong.chen@uq.edu.au

Xiangyu Zhao
xy.zhao@cityu.edu.hk

Jianxin Li
jianxin.li@ecu.edu.au

[1] School of Electrical Engineering and Computer Science, The University of Queensland, Brisbane, Queensland, Australia

[2] School of Business and Law, Edith Cowan University, Perth, Western Australia, Australia

[3] School of Electrical Engineering and Computer Science, City University of Hong Kong, Hong Kong, China

1 Introduction

Recommender systems are extensively utilized across a variety of online platforms, including e-commerce websites and social media networks. Typically, recommender systems are deployed on cloud servers and learn user/item embeddings of fixed and uniform sizes based on user-item interactions. These embeddings are subsequently utilized to predict users' preferred items by computing similarity scores between user and item embeddings. However, as the number of users and items increases, this cloud-based architecture encounters significant limitations, including privacy risks, high resource consumption and increased latency [1–3].

In light of these limitations, there is an increasing demand for deploying recommender systems directly on edge devices, such as mobile phones. This shift has motivated on-device recommender systems, which primarily focus on the efficient deployment and updating of lightweight embedding tables on resource-constrained devices.

Automated machine learning (AutoML) reduces human effort in model development by automating feature selection, hyperparameter tuning, and architecture search. Its goal is to streamline model design while improving efficiency, which is particularly relevant in large-scale and resource-constrained settings [4–8]. In recommender systems, lightweight embedding techniques can be viewed as domain-specific applications of AutoML, aiming to discover efficient parameterizations under strict memory constraints. Building on this connection, research in lightweight embeddings can be grouped into three main approaches: (1) parameter-sharing methods [9–12], (2) variable-size embedding methods [13–15], and (3) embedding-pruning methods [16–18].

Parameter-sharing methods reduce memory usage by enabling different users and items to share embedding parameters. However, these approaches still utilize dense embeddings during forward passes, which restricts the memory efficiency at inference time. Additionally, high-frequency users/items typically necessitate larger embedding sizes, whereas low-frequency users/items require smaller embedding sizes. Dense embeddings employ uniform embedding sizes across all entities, which may result in overfitting for high-frequency users/items and underfitting for low-frequency users/items [4, 5].

Variable-size embedding methods assign different dimensions to users and items based on frequency and contextual information, often utilizing AutoML techniques [5], but they typically require iterative model training to query the model performance and thus result in long training time.

In contrast, embedding-pruning methods enforce sparsity in the dense embedding table by setting certain parameters to zero, ensuring reduced memory usage while maintaining performance. These methods offer two key advantages over parameter-sharing and variable-size embedding approaches. First, unlike parameter-sharing methods, embedding pruning operates directly on sparse embeddings during forward passes, thereby reducing memory costs during inference. Table 1 presents the memory usage to store user/item embeddings for various lightweight embedding methods on the Gowalla[1] dataset, measured in bytes, during inference time. The results demonstrate that embedding pruning and variable-size methods employing sparse embeddings during inference significantly reduce memory overhead compared to parameter-sharing approaches that utilize dense embeddings. Second, in contrast to variable-size embedding methods, embedding pruning does not require iterative training for performance evaluation, thus reducing overall training time. This efficiency is particularly essential in scenarios such as federated learning, where training is distributed across multiple resource-constrained devices, and frequent communication with a central server can significantly increase both computational and communication overhead. Furthermore, embedding-pruning methods (e.g., DSL [17]) based on the Dynamic Sparse Training (DST) paradigm also provide the benefits of maintaining consistent sparsity throughout training and controllable memory budget, establishing them as the favored state-of-the-art approach for lightweight recommendation.

Embedding-pruning methods, despite producing sparse embeddings, rely on dense gradients during the backward pass, which results in a dense gradient table throughout the training process. This dense computation undermines the efficiency gains promised by embedding sparsity and slows down on-device updates. Moreover, the DST paradigm adopted by recent embedding-pruning methods introduces additional challenges. First, traditional DST initializes the mask matrices, which is used to prune parameters, with random uniform initialization. This strategy randomly removes a certain number of embedding parameters to conform to a predefined density ratio. Various studies [20, 21] have shown that this initialization strategy leads to suboptimal

Table 1 Inference-time memory usage (in bytes) of lightweight embedding methods on the Gowalla dataset, with full embedding size set to 128 (float32 storage). The density ratio indicates the proportion of active parameters relative to the full embedding size

Density	Embedding-Pruning	Variable-Size	Parameter-Sharing
	PEP [16]/DSL [17]	CIESS [14]/BET [19]	CERP [9]/DHE [11]
0.25	18,135,300	18,135,300	36,269,568
0.125	9,067,908	9,067,908	36,269,568
0.0625	4,534,212	4,534,212	36,269,568

[1] https://snap.stanford.edu/data/loc-Gowalla.html

post-training performance because it creates unstructured and inefficient connections. Alternatives like Erdos-Renyl initialization, which generates layer-wise sparse masks in DNNs and CNNs [21, 22] based on the input and output size of each layer and is equivalent to random uniform initialization on embedding tables. Gradient-based initialization strategies employ dense gradients as importance scores to prune parameters [23–25], increasing computational overhead and negating the efficiency benefits of DST.

Second, although traditional DST reactivates a portion of parameters and has proven effective in CNNs and DNNs, it performs suboptimally in recommender systems due to their unique challenges, where each training sample (i.e., user–item interaction) only updates the corresponding embeddings, resulting in limited parameter updates. Specifically, in CNNs and DNNs, parameters are tightly coupled: a single sample can influence a substantial subset of the network parameters through the interconnected layers, which are reused across all samples. For example, in a fully connected DNN, every neuron in one layer connects to all neurons in the next layer, so the forward and backward passes of a single sample generate gradients for nearly all weights in the model. Consequently, reactivating pruned weights based on large-magnitude gradients from the current batch can still improve global feature representations. In contrast, embedding tables in recommender systems consist of independent vectors, each corresponding to a specific user or item. For instance, when training on a user–item interaction, only the embeddings of that particular user and item are involved in computation and receive gradient updates, while all other embeddings remain unchanged. As a result, large-magnitude gradients are primarily associated with the sampled entities within the current batch, providing limited benefit to the overall model performance. In summary, despite improving training time and memory efficiency, embedding-pruning methods face three critical challenges: **(1) ineffective sparse initialization, (2) suboptimal regrowth strategies, and (3) use of dense gradients**. These issues limit their potential for broader applicability and performance optimization.

To overcome the aforementioned challenges, we propose SparseRec, a lightweight embedding method that enables sparse gradients and improves the efficiency of embedding pruning within the DST paradigm. First, SparseRec resolves the limitations of the DST paradigm. Specifically, it initializes the mask matrix with Nonnegative Matrix Factorization (NMF). NMF can be configured to generate sparse representations for users/items, where unimportant parameters tend to be zero and important parameters tend to remain nonzero. Via binarization, we get mask matrices containing only 0 s and 1 s for each parameter. The NMF output mask matrices can easily be configured to be sparse. Contrary to random uniform initialization, this data-driven method initializes the mask matrices to a local optimum, effectively addressing (1).

Secondly, SparseRec accumulates gradients over several consecutive batches to regrow inactive embedding parameters. This simple yet effective strategy reuses the gradients and avoids the need to compute additional importance scores. Compared to instantaneous gradients from a single batch, cumulative gradients can better identify inactive parameters that can significantly improve model performance after activation, tackling (2).

Thirdly, SparseRec utilizes sparse gradients, which directly addresses (3). Heddes et al. proposed GSE [26] to achieve sparse gradients on multilayer CNNs and DNNs by sampling and reactivating inactive parameters likely to have large gradients. However, it estimates the magnitudes of the gradient through the input activations of previous layers and is thus not applicable to embedding tables without input activations. Inspired by this, we propose a more efficient alternative: sampling embedding vectors based on their associated user/item frequency and computing sparse gradients exclusively for parameters within this subset. This frequency-based sampling strategy is computationally efficient, as probabilities are computed only once, avoiding iterative probability updates, and the storage overhead for one score per row is negligible. This method ensures that sparsity is maintained for both forward and backward passes. By calculating probabilities only for each user/item and using sparse gradients, SparseRec significantly reduces memory usage while preserving sparsity throughout the training phase. In short, the contributions of our paper can be outlined as follows:

- We introduce a novel NMF-based mask matrix initialization method specially designed for DST operating on GNN-based recommenders.
- We empirically demonstrate that the regrowth strategy using instantaneous gradients fails to allocate sufficient parameters to important users/items and propose a novel regrowth strategy to address this issue.
- To our knowledge, we are the first to address the problem of dense gradients in recommender systems. We present a sparse gradient technique that samples a subset of embedding vectors and evaluates gradients only for parameters in this subset.
- We combine SparseRec with three base recommenders and conduct empirical evaluations on three datasets. Comparing its performance with various state-of-the-art lightweight embedding algorithms affirms its efficacy for GNN-based recommenders.

2 Related Works

2.1 Deep Recommender Systems

Researchers have proposed various deep recommender models to capture user-item relationships [27]. The initial developments in deep recommender systems are based on Matrix Factorization (MF) [28–30], which aims to decompose the user-item interaction matrix to derive latent user and item embeddings. He et al. [31] advanced this approach by integrating Matrix Factorization with an MLP and proposed NCF. Beyond MF-based techniques, there has been exploration into GNN architectures [32–34]. For example, NGCF [32] extends NCF by incorporating graph convolution networks to model user-item interactions. LightGCN [33] further refines NGCF by removing self-connections, feature transformations, and linear activation functions. LightGCN is further enhanced by UltraGCN [34] and MGDCF [35]. In this paper, we focus on sparsifying the embedding table of GNN-based models because they outperform MF models such as NCF. Apart from user-item interactions, other data types such as location [36–38], activity level [39], text [40, 41], images [42], videos [43], user feedback transitions [44, 45] and social networks [46] have also been utilized.

2.2 Lightweight Embedding Methods

The pursuit of computational efficiency is a common theme in modern data-intensive applications, such as blockchain-enabled IoT systems where secure and lightweight operations are paramount [47–49]. To address these limitations in recommender systems, a variety of lightweight embedding methods have been developed to reduce memory consumption without significantly sacrificing model quality. For instance, parameter-sharing methods [9–12] improve memory efficiency by reusing embedding parameters across multiple users or items, typically through hashing techniques or compositional embeddings. Variable-size methods [13–15], on the other hand, dynamically adjust the embedding size for each user or item based on importance or usage frequency.

Embedding pruning remove weights based on some pruning criteria such as weight magnitudes [50–52], Taylor expansion [53], derivatives [54–56]. Liu et al. [16] proposed to learn the pruning threshold for each parameter in an end-to-end differentiable manner. Frankle et al. [57] introduced the lottery ticket hypothesis, stating a sparse sub-network can match the performance of the larger, randomly-initialized dense network when it is trained in isolation. To find such as sub-network, Han et al. [58] proposed a train-prune-retrain paradigm that trains a dense network, prunes it and retrains it. In an effort to avoid dense training and preserve sparsity throughout training, sparse training methods have been proposed, including static sparse training (SST) [23–25] and dynamic sparse training (DST) [17, 21, 59]. SST prunes the dense network at initialization prior to training while DST iteratively explores weights during training by pruning old weights and activate new ones. SST can be seen as the initialization step in DST. Our proposed method, SparseRec, can be classified as a DST method.

3 Preliminary

This section outlines the general framework of DST when it is applied to recommender models as in [59]. Let U and V denote the user and item embedding table, respectively. They can be stacked together to form the overall embedding table $\mathrm{E} = [\mathrm{U}; \mathrm{V}]$. We first initialize the binary mask matrix $\mathrm{M} = [\mathrm{M}^U; \mathrm{M}^V]$, which can be split into a user part M^U and an item part M^V. We will delve into the step of mask matrix initialization in Sect. 4.1. The embedding table can thus be sparsified by $\mathrm{E} \circ \mathrm{M}$, where $\circ$ denotes element-wise multiplication. The density ratio d of the embedding table can be expressed as

$$d = \frac{\|\mathrm{M}\|_0}{(m+n)s}, \tag{1}$$

where m is the number of users, n is the number of items, s is the maximal embedding size and $\|\cdot\|_0$ is the number of nonzero elements.

Once training begins, DST periodically performs weight exploration that follows a prune-regrow schedule every ΔT training steps. At the t-th step, we first calculate the mean weight magnitude of each user/item embedding table and normalize it with the mean weight magnitude of the entire embedding table:

$$\begin{aligned} \mu^U &= \frac{\sum_{i=1}^{m}\sum_{j=1}^{s}|\mathrm{U}_{ij}|}{\sum_{i=1}^{m+n}\sum_{j=1}^{s}|\mathrm{E}_{ij}|} \\ \mu^V &= \frac{\sum_{i=1}^{m}\sum_{j=1}^{s}|\mathrm{V}_{ij}|}{\sum_{i=1}^{m+n}\sum_{j=1}^{s}|\mathrm{E}_{ij}|} \end{aligned} \tag{2}$$

where $|\cdot|$ is the element-wise absolute value operation and subscript ij denotes the (i, j)-th element of a matrix. Note that μ^U and μ^V can be seen as the contribution of each user/item embedding table to the overall mean weight magnitude.

Then we enter the pruning stage at the t-th step, when a proportion (with pruning rate ρ_t) of active parameters in each user/item embedding table with the smallest magnitudes are pruned:

$$\begin{aligned}\mathcal{P}_t^U &= \text{top-k}(|\mathrm{U}|, \rho_t \|\mathrm{M}^U\|_0) \\ \mathcal{P}_t^V &= \text{top-k}(|\mathrm{V}|, \rho_t \|\mathrm{M}^V\|_0),\end{aligned} \tag{3}$$

where $\mathcal{P}_t^U$ (or $\mathcal{P}_t^V$) denote the set of active parameters to be pruned in the user and item embedding tables at the t-th step. Note that $\mathcal{P}^U$ and $\mathcal{P}^V$ are updated periodically during training, but we omit its subscript t henceforth for simplicity. We also omit this subscript for other variables if there is no ambiguity. After the pruning stage, a total of $\text{card}(\mathcal{P}^U) + \text{card}(\mathcal{P}^V)$ parameters are pruned. $\text{card}(\cdot)$ denotes the cardinality of a set. We can remove elements in $\mathcal{P} = \mathcal{P}^U \cup \mathcal{P}^V$ from E by setting their corresponding mask in M to zeros and performing element-wise multiplication.

Cosine annealing is applied to the pruning rate. Let ρ_0 denote the initial pruning rate and T_{end} denote the total number of training epochs, the pruning rate at the t-th step can be expressed as:

$$\rho_t = \frac{\rho_0}{2}\left(1 + \cos(\frac{\pi t}{T_{end}})\right) \tag{4}$$

Following the pruning stage is the regrowth stage, when inactive parameters are reactivated according to a gradient-based regrowth criterion which we will shed light on in Sects. 4.2 and 4.3. Newly activated parameters are initialized to zeros so they will not affect the output of the base recommender. The number of parameters to be activated in the user/item embedding table is the total number of pruned parameters times their respective magnitude contribution, namely:

$$\begin{aligned}\text{card}(\mathcal{G}^U) &= \mu^U\Big(\text{card}(\mathcal{P}^U) + \text{card}(\mathcal{P}^V)\Big) \\ \text{card}(\mathcal{G}^V) &= \mu^V\Big(\text{card}(\mathcal{P}^U) + \text{card}(\mathcal{P}^V)\Big),\end{aligned} \tag{5}$$

where $\mathcal{G}_U$ and $\mathcal{G}_V$ denote the regrowth sets containing the inactive parameters to be activated in the user and item embedding tables. Table 2 summarizes key notations.

Table 2 Summary of key notations

Symbol	Description
U, V	User and item embedding tables
R	Adjacency matrix
E	Overall embedding table, $E = [U; V]$
M	Binary mask matrix indicating active parameters
M_U, M_V	Mask matrices for user and item embeddings
$\circ$	Element-wise multiplication
s	Full embedding size
m, n	Number of users and items
d	Embedding table density ratio
$\|M\|_0$	Number of non-zero elements in M
ρ_t	Pruning rate at step t
ΔT	Weight exploration interval
P_U, P_V	Sets of parameters to be pruned from U and V
G_U, G_V	Sets of parameters to be reactivated in U and V
μ_U, μ_V	Mean magnitude contribution of user and item tables
∇L	Gradient of the loss function L
$\nabla_{A \cup S} L$	Sparse gradient w.r.t. active and sampled parameters
S_U, S_V	Sampled user and item embedding vectors
p_U, p_V	Sampling probabilities for users and items
f_U, f_V	User and item frequencies
ω	Sampling ratio for gradient estimation
C_U, C_V	Cumulative gradient matrices for U and V
A	Set of currently active parameters
P	Set of pruned parameters
G	Set of reactivated parameters
T_{end}	Total number of training epochs
η	Learning rate

4 Methodology

In short, SparseRec initializes a binary mask matrix using NMF to create a sparse embedding table. During training, it alternates between forward and backward passes, where the mask matrices enforce sparsity in embeddings. Sparse gradients are computed only for a sampled subset of parameters, significantly reducing memory usage and computational overhead. Every few iterations, SparseRec performs weight exploration by pruning parameters with the smallest magnitudes and regrowing parameters based on cumulative gradients over recent steps, overcoming the shortcomings of the traditional regrowth method that relies on instantaneous gradients. The pseudo code of SparseRec is provided by Algorithm 1 and a toy example of the sparse initialization and weight exploration process is given by Fig. 1.

4.1 Initialization of Mask Matrices

Modern recommender systems are predominantly embedding-based, where user and item identifiers are mapped to dense vector representations stored in large embedding tables. These tables account for the vast majority of model parameters [6, 9–11, 13] and thus largely determine the model's memory footprint and training efficiency. Regardless of architectural differences, whether matrix factorization (e.g., NCF [31]) or graph-based models (e.g., LightGCN [33], MGDCF [35]), the core computation remains centered around user–item embeddings and their interactions. Among them, GNN-based models have emerged as state-of-the-art systems for capturing high-order user–item interactions.

Given the dominant role of embedding tables in both model capacity and computational cost, recent research has explored embedding pruning techniques to reduce redundancy while preserving performance. The effectiveness of

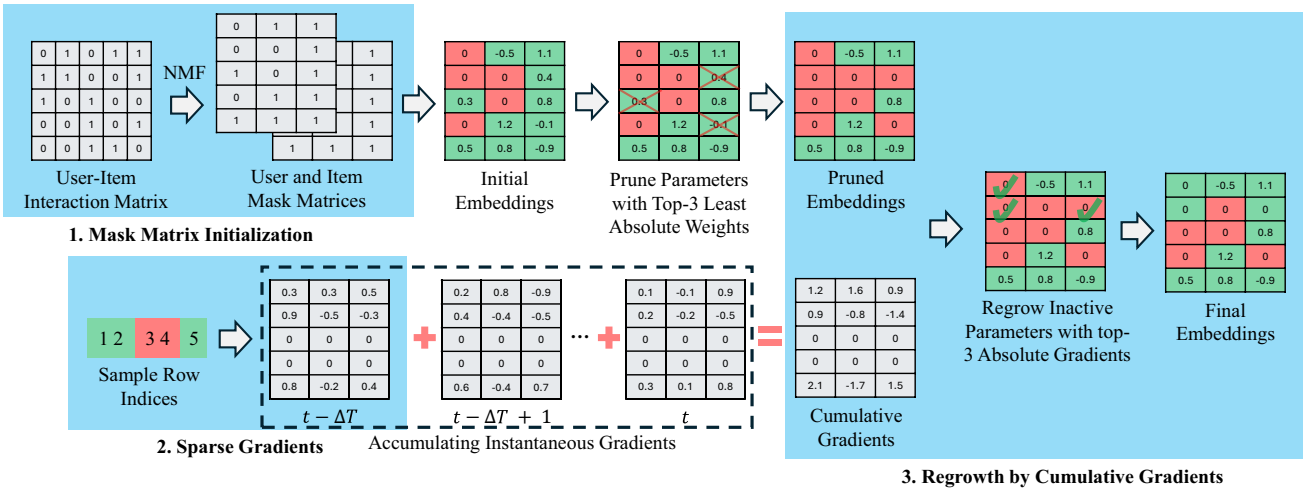


Fig. 1 A toy example of SparseRec. It is important to note that an equal number of parameters must be pruned and reactivated during the pruning and regrowth stages to ensure the parameter budget remains consistent

such pruning is highly sensitive to how the binary mask matrix is initialized, as it determines which embedding parameters are retained or removed. A widely used method to initialize the mask matrix is random uniform initialization, which constructs the binary mask matrix (used for pruning) by randomly setting a fixed proportion of embedding parameters to zero to achieve the target sparsity. Prior studies [20, 21] have shown that such randomness produces unstructured and inefficient connections in the sparse embedding table, ultimately impeding post-training performance. Moreover, random uniform initialization ignores both the interaction frequencies and the structural properties of the user–item graph, treating all entities as equally important.

Alternative initialization strategies have been proposed in other domains–for example, Erdos–Rényi initialization in CNNs and DNNs [20, 23]–but these methods are not directly applicable to recommender systems, where the primary goal is to prune embedding tables rather than dense convolutional layers. Gradient-based methods can provide more informed initialization [23–25], but they introduce significant computational overhead due to their reliance on dense gradients.

To tackle this challenge, we propose to use NMF to derive the binary mask matrix M. Matrix Factorization (MF) has been commonly used as recommender systems [28–30]. Unlike other MF methods, NMF receives the sparse adjacency matrix as input and can be configured to produce sparse representations for users and items, where important parameters are likely to be 1 s and unimportant ones are likely to be 0s. In contrast to random uniform initialization, this data-driven approach can initializes the mask matrix to a local optimum. Concretely, we factorize the adjacency matrix $\mathrm{R} \in \mathbb{R}^{m \times n}$ into $\mathrm{W} \in \mathbb{R}^{m \times s}$ and $\mathrm{H} \in \mathbb{R}^{n \times s}$ such that $\mathrm{R} \approx \mathrm{W}\mathrm{H}^T$. By binarizing W and H, we obtain M^U and M^V:

$$\mathrm{M}^U_{ij} = \begin{cases} 1 & \text{if } \mathrm{W}_{ij} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$
$$\mathrm{M}^V_{ij} = \begin{cases} 1 & \text{if } \mathrm{H}_{ij} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

If the resultant density exceeds the desired level, we prune the weights with the smallest magnitudes in W or H. Conversely, if the density falls short of the target, we enter a regrowth stage after the first epoch and activate enough parameters to reach the target density. Additionally, it is worth noting that the proposed mask matrix initialization approach only initializes M. The non-zero parameters of the embedding table E are still initialized randomly.

4.2 Sparse Gradients

Following the initialization of mask matrices, the training stage commences, comprising forward and backward passes. During forward passes, sparse user and item embeddings are calculated to generate predictions and the mask matrices enforce embedding sparsity. In backward passes, SparseRec computes sparse gradients exclusively for the

selected embedding parameters, ensuring computational efficiency and reduced memory usage.

Since the gradients for inactive parameters are not necessarily zero, they need to be computed for both active and inactive parameters in the regrowth stage. To avoid dense gradients, we compute gradients for the active parameters plus a subset of inactive parameters that are predicted to be important to performance improvement. Previous studies [13, 16] have shown that frequently occurring users/items often require larger embedding sizes. Leveraging this insight, we sample a portion (with sampling ratio ω) of embedding vectors without replacement from the user and item embedding tables based on user/item frequencies. This approach incurs minimal computational and storage overhead, as sampling probabilities are computed once and then stored for subsequent use. Consequently, vectors associated with more frequently occurring users/items are more likely to be sampled. The sampling process can be viewed as selecting $\omega m/\omega n$ elements from a multinomial probability distribution with probabilities $\mathrm{p}^U/\mathrm{p}^V$:

$$\begin{aligned} \mathcal{S}^U &= \mathrm{Multinomial}(\mathrm{p}^U, \omega m) \\ \mathcal{S}^V &= \mathrm{Multinomial}(\mathrm{p}^V, \omega n), \end{aligned} \tag{7}$$

where $\mathcal{S}^U$ and $\mathcal{S}^V$ are the sets of embeddings vectors sampled from the user and item embedding tables. The probabilities can be calculated as:

$$\begin{aligned} \mathrm{p}^U &= \mathrm{Softmax}(\mathrm{f}^U) \\ \mathrm{p}^V &= \mathrm{Softmax}(\mathrm{f}^V), \end{aligned} \tag{8}$$

where f^U and f^V are the user and item frequencies. Notably, the stochastic nature of the sampling strategy, combined with the Softmax function for smoothing probabilities, ensures that infrequent yet critical users/items are adequately considered. Unlike a deterministic heuristic-based strategy that prioritizes frequent users/items, our stochastic approach promotes a more balanced activation. Let $\mathcal{A}^U$ and $\mathcal{A}^V$ denote the active parameters from the user and item embedding table. Define $\mathcal{A} = \mathcal{A}^U \cup \mathcal{A}^V$ as the set of active parameters in both tables. We only need to compute gradients w.r.t. the parameters in the union of active parameters and the sampled set:

$$\nabla_{\mathcal{A}\cup\mathcal{S}}\mathcal{L}_{ij} = \begin{cases} \nabla\mathcal{L}_{ij} & \text{if } ij \in \mathcal{A}\cup\mathcal{S} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

where $\nabla_{\mathcal{A}\cup\mathcal{S}}\mathcal{L}$ is the sparse gradients w.r.t. $\mathcal{A}\cup\mathcal{S}$, $\nabla\mathcal{L}$ is the dense gradients and $\mathcal{L}$ is the loss function:

$$\mathcal{L} = \mathcal{L}_{BPR} + \|\mathrm{E}\circ\mathrm{M}\|^2 \tag{10}$$

4.3 Parameter Regrowth by Cumulative Gradients

Upon entering the regrowth stage at step t, traditional DST uses instantaneous gradients at the t-th batch to assist parameter activation. Consequently, when they are applied to recommender systems, users/items sampled in the t-th batch, which do not necessarily contribute to the model performance a lot, enjoy more parameter activation. To reactivate parameters for the users/items that truly matter, we accumulate the sparse gradients w.r.t. the parameters in $\mathcal{S}$ from the $(t-\Delta T)$-th to the t-th step and store the cumulative gradients in $\mathcal{C}^U$ and $\mathcal{C}^V$:

$$\begin{aligned} \mathrm{C}^U &= \sum_{i=t-\Delta T}^{t} \left(\nabla_{\mathcal{S}^U}\mathcal{L}\right)_i \\ \mathrm{C}^V &= \sum_{i=t-\Delta T}^{t} \left(\nabla_{\mathcal{S}^V}\mathcal{L}\right)_i \end{aligned} \tag{11}$$

In the pruning stage, a total number of $\mathrm{card}(\mathcal{P})$ parameters are pruned. Therefore, to maintain target density, we need to regrow the same number of parameters in the regrowth stage. We use the mean magnitude contribution to redistribute parameters across the user and item embedding tables by activating $\mu^U\mathrm{card}(\mathcal{P})$ parameters from the user embedding table and $\mu^V\mathrm{card}(\mathcal{P})$ parameters from the item embedding table with the greatest cumulative gradient magnitudes:

$$\begin{aligned} \mathcal{G}^U &= \text{top-k}\Big(|\mathrm{C}^U|, \mu^U\mathrm{card}(\mathcal{P})\rho\Big) \\ \mathcal{G}^V &= \text{top-k}\Big(|\mathrm{C}^V|, \mu^V\mathrm{card}, (\mathcal{P})\rho\Big) \end{aligned} \tag{12}$$

where ρ is the pruning rate. Upon completion of the regrowth stage, $\mathcal{C}^U$ and $\mathcal{C}^V$ are re-initialized to zero matrices.

It is important to note that $\mathcal{P}$, $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{G}$ are implemented as binary matrices instead of actual hash sets to conserve memory. As a result, all set operations can be efficiently performed using logical bit-wise operators. The whole weight exploration schedule is depicted by Venn diagrams in Fig. 2.

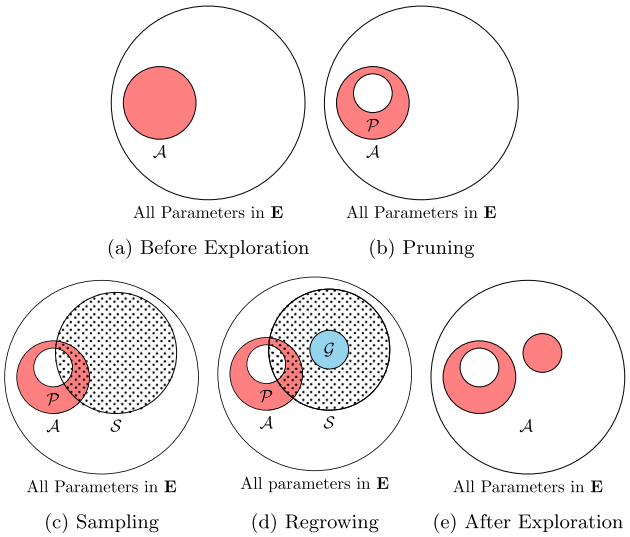


Fig. 2 Venn diagrams illustrating the relationship between $\mathcal{A}$, $\mathcal{S}$, $\mathcal{G}$ and $\mathcal{P}$ in different stages of weight exploration. $\mathcal{A}$ (red) denotes active parameters; $\mathcal{G}$ (blue) denotes the parameters to be activated, $\mathcal{S}$ (dotted) denotes the sampled parameters, and the white area denotes inactive parameters. $\mathcal{G}$ is mutually exclusive with $\mathcal{A}$ but not necessarily with $\mathcal{P}$. $\mathcal{S}$ and $\mathcal{A}$ are not necessarily not exclusive because active parameters in $\mathcal{S}$ will be filtered out when inactive parameters are reactivated

4.4 Memory Complexity Analysis

To assess the efficiency of SparseRec, this subsection analyzes its memory complexity across the main training components, including mask initialization and the weight exploration process.

4.4.1 NMF-based Mask Matrix Initialization

Mask matrix initialization precedes the weight exploration process. In this subsection, we provide a detailed analysis of the memory complexity of two mask matrix initialization approaches: random uniform initialization and our novel NMF-based initialization.

The adjacency matrix $\mathrm{R} \in \mathbb{R}^{m \times n}$ is highly sparse and typically stored in compressed formats such as Compressed Sparse Row or Compressed Sparse Column. The memory requirement for this storage is $O(\mathrm{nnz}(\mathrm{R}))$, where $\mathrm{nnz}(\cdot)$ is the number of nonzero elements.

In random initialization, the embedding tables are first populated with dense weights, which are then sorted and pruned to meet the target sparsity. This requires $O(mk + nk)$

$\mathbf{M}^U, \mathbf{M}^V \leftarrow$ Eq. (6) ▷ NMF sparse initialization
$\mathbf{U} \leftarrow \mathbf{U} \circ \mathbf{M}^U$ and $\mathbf{V} \leftarrow \mathbf{V} \circ \mathbf{M}^V$; ▷ Apply masks
for $t = 1, \cdots, T_{end}$ **do**
  if $t == 1$ or $t \bmod \Delta T == 0$ **then**
    $\mathcal{S}^U, \mathcal{S}^V \leftarrow$ Eq. (7) ▷ Sampling vectors
    $\mathbf{C} \leftarrow \mathbf{0}$; ▷ Set cumulative sparse gradients to $\mathbf{0}$
  end if
  $\mathbf{C}^U \leftarrow \mathbf{C}^U + \nabla_{\mathcal{S}^U} \mathcal{L}$ and $\mathbf{C}^V \leftarrow \mathbf{C}^V + \nabla_{\mathcal{S}^V} \mathcal{L}$
  $\mathbf{U} \leftarrow \mathbf{U} - \eta \nabla_{\mathcal{A}^U \cup \mathcal{S}^U} \mathcal{L}$ and $\mathbf{V} \leftarrow \mathbf{V} - \eta \nabla_{\mathcal{A}^V \cup \mathcal{S}^V} \mathcal{L}$
  if $t \bmod \Delta T == 0$ **then**
    $\mathcal{A}^U, \mathcal{A}^V \leftarrow \mathrm{getActiveParameters}(\mathbf{M})$
    $\mu^U, \mu^V \leftarrow$ Eq. (2)
    $\mathcal{P}^U, \mathcal{P}^V \leftarrow$ Eq. (3) ▷ Pruning parameters
    $\mathcal{G}^U, \mathcal{G}^V \leftarrow$ Eq. (12) ▷ Regrowing parameters
  end if
  $\mathbf{M} \leftarrow \mathrm{setActiveParameters}(\mathcal{A} - \mathcal{P} \cup \mathcal{G})$
  $\mathbf{U} \leftarrow \mathbf{U} \circ \mathbf{M}^U$ and $\mathbf{V} \leftarrow \mathbf{V} \circ \mathbf{M}^V$
  $\rho_t \leftarrow$ Eq. (4);
end for

Algorithm 1 SparseRec

memory. Including the storage of the sparse interaction matrix, the total memory usage up to initialization is:

$$O(mk + nk + \text{nnz}(\text{R})) \tag{13}$$

In contrast, the NMF-based initialization employs a Coordinate Descent solver directly on the sparse interaction matrix R. The solver requires the same $O(\text{nnz}(\text{R}))$ to store the input, plus auxiliary memory for latent factors and temporary variables. Specifically, the memory requirement is $O(mk + nk + k^2)$, where the k^2 term arises from intermediate updates during factorization [60]. Hence, the total memory usage is:

$$O(mk + nk + k^2 + \text{nnz}(\text{R})) \tag{14}$$

Since $k \ll m$ and $k \ll n$, k^2 is negligible compared to *mk* and *nk* in large-scale recommendation settings ($k \ll m, n$). Therefore, NMF-based does not incur significantly more memory usage than random uniform initialization.

More importantly, initialization is a one-time cost that can be performed offline on resource-rich servers, after which training and inference proceed efficiently on memory-constrained devices.

4.4.2 Weight Exploration

Here, we provide a theoretical proof that the weight exploration process of SparseRec has improved memory efficiency compared to other embedding pruning baselines PEP and DSL. We analyze the memory complexity of SparseRec w.r.t. the sizes of the sparse embedding table E, the sparse gradients $\nabla_{\mathcal{A}\cup\mathcal{S}}\mathcal{L}$ and the cumulative gradient table C, all of which are sparse. The total number of parameters can be hence expressed as $\text{params} = ds(m+n) + \text{card}(\mathcal{A}\cup\mathcal{S}) + \text{card}(\mathcal{S})$, which reaches its maximum in the virtually impossible case where $\mathcal{S}$ and $\mathcal{A}$ are mutually exclusive:

$$\begin{aligned}\text{params} &= ds(m+n) + \text{card}(\mathcal{A}\cup\mathcal{S}) + \text{card}(\mathcal{S}) \\ &\leq ds(m+n) + \text{card}(\mathcal{A}) + \text{card}(\mathcal{S}) + \text{card}(\mathcal{S})\end{aligned} \tag{15}$$

$$\begin{aligned}&= ds(m+n) + ds(m+n) + \omega s(m+n) + \omega s(m+n) \\ &= (2d + 2\omega)s(m+n)\end{aligned} \tag{16}$$

The memory usage of DSL (traditional DST) is dominated by the sparse embedding table and the dense gradients, resulting in a total of $(d+1)s(m+n)$ parameters. We can compare it to the memory usage of SparseRec:

$$(2d + 2\omega)s(m+n) \leq (d+1)s(m+n) \Rightarrow \omega \leq \frac{1-d}{2} \tag{17}$$

Therefore, SparseRec uses less memory than conventional DST methods such as DSL when $\omega \leq \frac{1-d}{2}$.

The memory usage of PEP is dominated by a dense embedding table, the dense gradients and a dense table storing the pruning thresholds, resulting in a total of $3s(m+n)$ parameters, which is higher than SparseRec and DSL. In summary, the space complexity for SparseRec, DSL and PEP are $\mathcal{O}\Big((2d+2\omega)s(m+n)\Big)$, $\mathcal{O}\Big((d+1)s(m+n)\Big)$ and $\mathcal{O}\Big(3s(m+n)\Big)$, respectively.

5 Experiments

In this section, we aim to answer the following research questions:

- **RQ1**: How does SparseRec perform compared to other state-of-the-art lightweight embedding methods?
- **RQ2**: Can NMF-based sparse initialization improve the performance of SparseRec?
- **RQ3**: Can parameter regrowth by cumulative gradients improve the performance of SparseRec?
- **RQ4**: Is there any correlation between the user/item frequency and embedding size?
- **RQ5**: Can SparseRec handle cases where infrequent but critical users/items?

5.1 Experimental Setup

5.1.1 Datasets

Each lightweight embedding method is evaluated on the user rating/check-in data from three datasets: Gowalla[2], Yelp2018[3] and Movielens 10M[4] as in [32, 33, 35], all of which are commonly used in lightweight embedding research [9, 10, 14, 15, 19]. The Gowalla dataset comprises 1,027,370 interactions involving 29,858 users and 40,981 items and the Yelp dataset contains 1,561,406 interactions among 31,668 users and 38,048 items. 10-core processing is applied on the Movielens 10M dataset and the processed Movielens 10M dataset has 9,995,470 interactions among 69,878 users and 9,708 items. Compared to Gowalla and Yelp2018, the Movielens 10M dataset exhibits higher data density. All the datasets are split into training, validation, and test sets with a ratio of 70%, 10%, and 20%, respectively. We treat all observed user-item interactions as positive and unobserved interactions as negative. By leveraging

[2] https://snap.stanford.edu/data/loc-Gowalla.html

[3] https://www.yelp.com/dataset

[4] https://grouplens.org/datasets/movielens/10m/

datasets from three distinct domains with varying data densities, we aim to comprehensively evaluate the effectiveness of SparseRec across diverse recommendation scenarios.

5.1.2 Baselines

We compare SparseRec against several approaches, including embedding-pruning methods:

- PEP [16]: PEP (Plug-in Embedding Pruning) reduces memory usage by adaptively learning pruning thresholds from data to eliminate redundant embedding parameters for each feature, resulting in a mixed-dimension embedding scheme that can be flexibly applied to various base recommendation models.
- DSL [17]: DSL introduces a dynamic sparse training paradigm that learns sparse recommender models from scratch by periodically pruning and regrowing parameters based on their importance, ensuring consistent sparsity and minimal parameter usage throughout both training and inference.
- CIESS [14]: CIESS leverages reinforcement learning with a random walk-based exploration strategy to continuously search for optimal, fine-grained embedding sizes for each user and item, enabling memory-efficient recommendation without relying on predefined discrete size sets.
- BET [19]: BET formulates the search for user/item embedding sizes as a table-level set-based optimization problem guided by an action fitness predictor, ensuring the final embedding table strictly adheres to a pre-specified memory budget while avoiding inefficient instance-level search.
- CERP [9]: CERP constructs compact representations by composing each user/item embedding from two jointly pruned, smaller meta-embedding tables using learnable thresholds, while enforcing regularization to encourage complementary information encoding between the tables.
- DHE [11]: DHE replaces traditional embedding tables with a deep embedding network that generates embeddings on the fly by transforming multi-hash encoded identifier vectors through a neural network, enabling efficient handling of high-cardinality and unseen categorical features without storing explicit embeddings.

PEP and DSL are embedding-pruning methods, CIESS and BET are variable-size embedding methods, and CERP and DHE are parameter-sharing methods. Since DHE does not use an embedding table, we are unable to fix its density. Therefore, we search $\{8, 16, 32, 64, 128\}$ for its best output embedding size and present its best performance.

5.1.3 Base Recommenders

To demonstrate its effectiveness and generalizability with other GNN recommenders, we integrate SparseRec and the selected baselines with the following base recommenders:

- XSimGCL [61]: XSimGCL simplifies contrastive learning for recommendation by replacing graph augmentations with a noise-based embedding perturbation strategy, enabling more uniform user/item representations that mitigate popularity bias and improve long-tail item exposure.
- MGDCF [35]: MGDCF reformulates GNN-based collaborative filtering as a Markov diffusion process that generates fixed context features, which are then encoded by a fully connected layer in a simplified network representation learning framework, highlighting the critical role of ranking loss in performance.
- LightGCN [33]: LightGCN simplifies graph convolutional networks for collaborative filtering by removing feature transformation and nonlinear activation, relying solely on linear neighborhood aggregation and layer-wise embedding summation to learn user and item representations more effectively and efficiently.

XSimGCL and MGDCF demonstrate state-of-the-art recommendation performance while LightGCN is a classic recommendation model widely used in recommendation research [9, 10, 14, 15, 19]. Each method with adjustable embedding density ratio is tested with the mean embedding size set to 8, 16 and 32, corresponding to density ratios of 6.25%, 12.5% and 25% when the full embedding size is 128. The combination of three base recommenders, three datasets and three density ratios lead to 27 experimental settings.

5.1.4 Metrics

We evaluate the performance of each method using Recall@20 and NDCG@20, which are common evaluation metrics in recommendation [32, 33, 62].

5.2 Implementation Details

Here we provide the implementation details of SparseRec. Early stopping is applied after 300 training epochs, evaluating model performance on the validation set every 5 epochs, with a patience threshold of 5 epochs. The total number of training epochs is T_{end} is 500. NMF is solved using the Coordinate Descent solver. Learning rate decay is used with the initial learning rate set to 0.01. It decays after every epoch by a factor of 0.99 when the LightGCN base recommender is trained on the Yelp dataset. The decay rate is 0.995 in all

Table 3 Comparison of Recall@20 (R@20) and NDCG@20 (N@20) scores in percentage (%) for various lightweight embedding methods on the Gowalla, Yelp, and Movielens 10M datasets, with density ratio d set to 0.0625, 0.125 and 0.25. The highest scores are highlighted and the second highest scores are underlined. The sampling ratio ω is set to $\frac{1-d}{2}$ in all settings

d	Model	Gowalla						Yelp						Movielens 10M					
		XSimGCL		MGDCF		LightGCN		XSimGCL		MGDCF		LightGCN		XSimGCL		MGDCF		LightGCN	
		R@20	N@20	R@20	N@20	R@20	N@20	R@20	N@20	R@20	N@20	R@20	N@20	R@20	N@20	R@20	N@20	R@20	N@20
0.0625	PEP	**17.31**	13.35	16.30	12.64	15.39	11.87	8.86	7.47	7.50	6.29	7.57	6.20	27.83	35.49	28.17	35.92	25.29	31.87
	BET	15.46	12.01	15.06	11.81	14.21	11.13	7.49	6.27	7.51	6.29	7.63	6.37	28.86	35.60	28.97	37.35	26.93	33.71
	CIESS	15.29	11.70	14.92	11.64	14.05	10.87	7.36	6.11	7.46	6.25	7.51	6.23	28.00	34.52	28.43	36.67	26.55	33.49
	DSL	16.58	12.75	15.95	12.48	14.04	10.90	8.96	7.58	7.66	6.51	6.93	5.74	28.97	36.45	28.58	36.95	26.47	33.20
	CERP	15.72	12.17	15.60	12.45	14.77	11.46	8.21	6.89	7.67	6.40	7.60	6.19	27.00	33.71	28.09	36.12	25.08	31.79
	SD	11.74	8.91	12.22	9.54	11.97	9.07	6.90	5.68	6.14	5.11	6.72	5.53	24.88	31.08	21.94	28.32	24.67	31.11
	Ours	17.21	**13.44**	**17.27**	**13.61**	**15.98**	**12.61**	**9.31**	**7.91**	**8.31**	**6.99**	**8.49**	**7.16**	**29.30**	**37.43**	**29.39**	**37.85**	**28.08**	**35.14**
0.125	PEP	18.18	14.04	18.38	14.25	16.80	12.87	9.25	7.80	8.78	7.35	8.27	6.80	28.15	35.97	28.17	35.95	26.20	33.10
	BET	16.97	13.06	17.94	14.06	16.41	12.73	8.43	7.09	**9.44**	**8.02**	8.66	7.30	29.44	36.35	29.13	37.64	27.40	33.87
	CIESS	16.32	12.60	17.52	13.83	16.37	12.72	8.31	6.98	8.91	7.56	8.42	7.09	28.62	35.39	28.70	37.10	27.05	33.78
	DSL	18.00	13.98	18.60	14.67	15.50	11.92	9.43	8.00	9.03	7.67	7.36	6.06	29.00	37.02	29.13	37.76	27.04	34.28
	CERP	16.85	13.01	17.98	13.94	16.38	12.63	8.73	7.32	8.74	7.34	8.10	6.65	27.57	34.28	26.30	33.71	26.55	33.49
	SD	14.75	11.39	15.89	12.34	15.18	11.65	8.33	6.99	7.71	6.47	7.74	6.43	27.13	33.93	25.13	31.97	25.80	32.72
	Ours	**18.22**	**14.18**	**19.00**	**14.93**	**17.18**	**13.49**	**9.76**	**8.33**	9.27	7.93	**9.07**	**7.62**	**29.73**	**37.65**	**29.84**	**38.84**	**28.57**	**35.02**
0.25	PEP	18.75	14.45	20.05	15.66	17.54	13.47	9.50	8.01	10.06	8.42	8.60	7.12	28.25	36.08	28.78	36.93	26.93	34.09
	BET	17.82	13.75	20.19	15.90	18.05	13.89	9.12	7.70	9.98	8.42	9.01	7.62	29.34	35.95	29.51	38.12	27.80	33.86
	CIESS	17.45	13.54	20.00	15.71	17.98	13.83	9.12	7.66	9.85	8.32	8.95	7.53	29.19	36.01	29.31	37.67	27.51	33.80
	DSL	18.76	14.56	20.18	15.84	16.63	12.64	9.86	8.33	10.02	8.48	7.80	6.34	29.26	37.28	29.34	38.13	27.70	34.59
	CERP	17.55	13.59	19.97	15.53	17.41	13.41	9.18	7.70	9.96	8.35	8.64	7.12	27.28	32.56	27.95	36.02	27.45	34.05
	SD	17.33	13.41	19.57	15.14	17.17	13.25	8.99	7.52	9.83	8.29	8.19	6.80	28.94	35.68	27.38	35.29	26.91	33.79
	Ours	**18.80**	**14.71**	**20.79**	**16.34**	**18.23**	**14.24**	**10.03**	**8.54**	**10.34**	**8.84**	**9.49**	**7.97**	**29.84**	**37.48**	**30.22**	**39.00**	**29.00**	**35.50**
	DHE	13.29	10.16	20.39	15.92	14.50	11.12	7.25	4.50	10.22	8.69	7.34	6.06	26.19	33.56	26.94	34.68	27.99	34.78

other settings. The minimal learning rate is 0.0005. The full embedding size s is 128. Bayesian Personalized Ranking Loss [63] is used to train the recommenders. Adam optimizer [64] is used with weight decay of 0.0001. The batch size is 8,000 for the Gowalla and Yelp datasets and 20,000 for the Movielens 10M dataset. Empirical results show that large batch size substantially speeds up training with the same recommendation accuracy. The sampling ratio ω is set to $\frac{1-d}{2}$ as SparseRec saves memory compared to PEP and DSL when $\omega \leq \frac{1-d}{2}$ as discussed in Sect. 4.4. For the base recommenders, we inherit their optimal settings as reported in their original papers. After training, the embedding table is quantized into signed 8-bit integers as quantization is a common practice in the real world. During testing, it is converted back to floating-point format.

5.3 Overall Performance Comparison

Table 3 presents the Recall@20 and NDCG@20 scores for various lightweight embedding methods across different base recommender models and density ratios. Our proposed method achieves state-of-the-art performance in 25 out of the 27 settings, effectively addressing RQ1. The largest performance gain over the second-best baseline is 11.79%, achieved on the Yelp dataset with a 0.0625 density ratio and the LightGCN base recommender. Other methods such as DSL, PEP and BET also show competitive performance in many settings. For the settings where SparseRec achieves the best performance, we perform paired student's t-test between SparseRec and the best baseline method to further affirm the advantageous efficacy of SparseRec. With all p-values being less than 0.02, we can confidently conclude that the superior performance of SparseRec is statistically significant and not due to random chance.

In most cases, we observe that the performance improvement of SparseRec over SD is more substantial in low-density settings, where efficient parameter utilization is crucial to minimizing performance degradation. This highlights SparseRec's robustness in memory-constrained scenarios such as mobile and WoT devices.

On the Movielens 10M dataset, increasing parameter density notably enhances recommendation performance of conventional small dense networks; however, the performance improvement from increasing parameter density is comparatively marginal when utilizing lightweight embedding algorithms, particularly with XSimGCL and MGDCF as base models. For example, the Recall@20 score of PEP increases by only 0.006 when parameter density is raised from 0.0625 to 0.25 using MGDCF as the base model. This observation indicates that, on the Movielens 10M dataset, lightweight embedding algorithms tend to achieve strong performance even at low parameter densities, where efficient allocation of parameters is more critical.

Moreover, although DHE employs fewer parameters than all other methods by eliminating the conventional embedding table, it exhibits the worst performance in top-k recommendation.

5.4 Model Component Analysis

Here we showcase the performance efficacy of the core components of SparseRec, which are our main contributions. MGDCF is chosen as the base recommender as it has the best performance. The density ratio is set to 0.125.

5.4.1 NMF-based Sparse Initialization

To investigate the influence of NMF mask matrix initialization, we run two versions of SparseRec: one initializes the sparse network with random uniform distribution as used in traditional DST and the other does so using NMF. Table 4 shows NMF initialization improves the performance of SparseRec, thus answering RQ2.

5.4.2 Regrowth by Cumulative Gradient

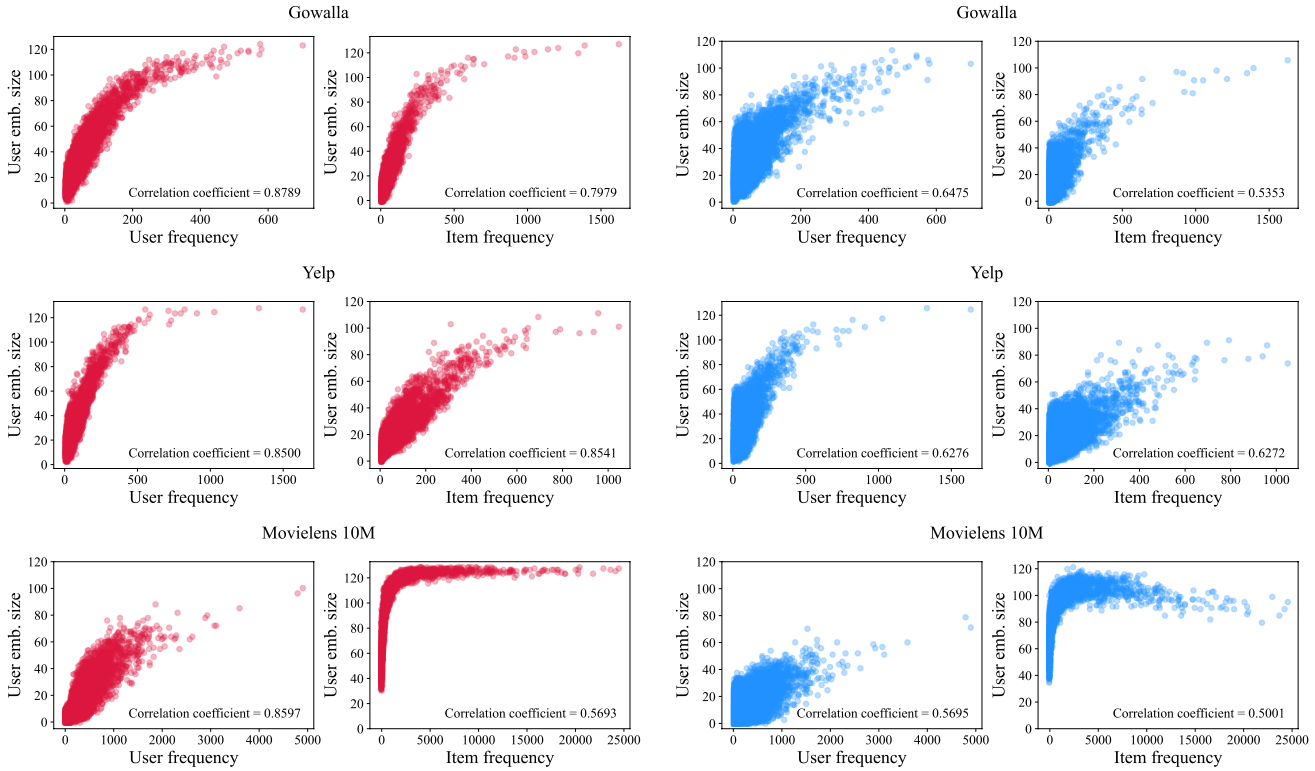
To illustrate the impact of regrowing parameters based on cumulative gradients versus instantaneous gradients, we run two versions of `SparseRec`: one using cumulative gradients and the other using instantaneous gradients as used in traditional DST. Shown in Table 4, using cumulative gradients results in better performance, thus answering RQ3.

To fully understand this performance boost, we calculate and plot the embedding size for each user and item against their frequency. We also calculate the Pearson Correlation Coefficient between the embedding sizes and frequency

Table 4 Recommendation performance of different SparseRec variants on the Gowalla, Yelp, and Movielens 10M datasets w.r.t. Recall@20 (R@20) and NDCG@20 (N@20) scores

		Gowalla		Yelp		Movielens 10M	
Regrowth	Initialization	R@20	N@20	R@20	N@20	R@20	N@20
Cumulative	NMF	**19.04**	**14.97**	**9.22**	**7.89**	**29.87**	**37.99**
Instantaneous	NMF	18.57	14.62	9.02	7.70	29.34	37.95
Cumulative	Uniform	18.09	14.33	8.85	7.54	29.13	37.76
Instantaneous	Uniform	17.54	13.86	8.42	7.17	29.01	37.56

The highest scores are highlighted



(a) Regrowth by cumulative gradients (b) Regrowth by instantaneous gradients

Fig. 3 The relationship between frequency and embedding sizes in regrowth by (**a**) cumulative gradients and (**b**) instantaneous gradients with their respective Pearson Correlation Coefficient

scores in each case to quantify the strength of correlation. To exclude the impact of the frequency-based sampling, we set the sampling ratio to 1 and use a dense cumulative gradient table. We present result in Fig. 3. We can see both regrowth strategies result in a positive correlation between frequency and embedding size, with some critical tail users/items assigned with larger sizes, which answers RQ4. This finding is consistent with the findings in [13]. However, the frequency-size correlation is weaker when instantaneous gradients are used, resulting in a larger number of frequent users/items being assigned smaller embedding sizes. This suggests that parameter regrowth based on instantaneous gradients leads to less reliable regrowth and fails to properly identify important parameters.

5.5 Analysis of Hyperparameter

In this section, we analyze the influence of key hyperparameters. MGDCF is chosen as the base recommender as it has the best performance.

5.5.1 Weight Exploration Interval ΔT

The weight exploration interval specifies the number of training steps between every weight exploration process and it has a vital influence on the performance of DST. If ΔT is too small, the parameters are not updated enough to exceed the pruning threshold; if ΔT is too large, the parameters are not adequately explored and will result in inferior performance [65]. When $\Delta T -> \infty$, it is equivalent to static sparse training, where we train a base recommender that has been pruned at initialization [17]. Let b be the number of steps in an epoch. To identify the optimal value for ΔT, we set it to $1b$, $3b$, $5b$, $7b$ and $9b$. Then we plot the resulting post-training performance in terms of Recall@20 and NDCG@20 scores under all density ratios $d \in \{0.0625, 0.125, 0.25\}$. Figure 4 shows that SparseRec performs best in the majority of settings when ΔT is set to $7b$ on the Yelp dataset and $5b$ on the Gowalla dataset. As to the Movielens 10M dataset, the performance fluctuations are subtle, with slightly better performance achieved with ΔT set to $1b$, $9b$ and $5b$ when the density ratio equals 0.25, 0.125 and 0.0625, respectively.

In summary, our experiments show that recommendation performance is largely insensitive to the choice of ΔT

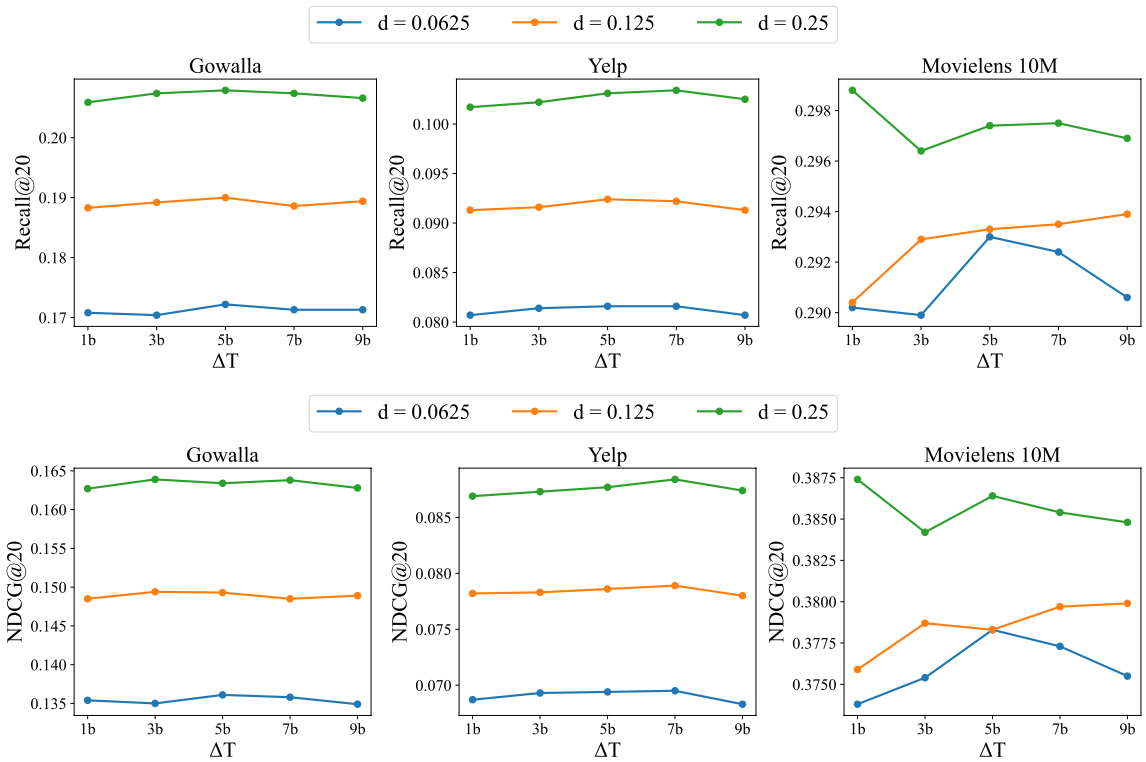


Fig. 4 Hyperparameter analysis of the weight exploration interval ΔT w.r.t. Recall@20 and NDCG@20 on the Gowalla, Yelp and Movielens 10M datasets

across all three datasets. For practical deployment, setting ΔT within the range of $5b$–$7b$ provides a reliable and effective default, making this choice sufficient for most real-world recommendation scenarios.

5.5.2 Sampling Ratio ω

The sampling ratio ω plays a crucial role in determining the proportion of embedding vectors selected from the embedding table, thereby limiting gradient calculations to only those sampled vectors. As outlined in Sect. 4.4, ω should not exceed $\frac{1-d}{2}$ to ensure that memory savings are realized compared to traditional DST. If ω is too high, the memory savings diminish, whereas an ω close to zero essentially results in random regrowth. We define $h = \frac{1-d}{2}$ and present the recommendation performance metrics Recall@20 and NDCG@20 along with the peak memory usage as discussed in 4.4 in Table 5. The analysis is conducted for ω values in $\{h, \frac{h}{2}, \frac{h}{4}, \frac{h}{8}, \frac{h}{16}\}$ across different density ratios $d \in \{0.25, 0.125, 0.0625\}$. Meanwhile, we also plot the relationship between peak memory usage against different density ratios d and sampling ratios ω in Fig. 5.

As illustrated in Table 5, SparseRec demonstrates improved recommendation performance with larger ω, which naturally results in increased memory consumption. A notable observation is that the recommendation performance remains relatively stable even as the sampling ratio is halved from h to $\frac{h}{2}$, particularly with low density ratios. However, a larger drop in recommendation performance is observed across all density ratios when ω falls below $\frac{h}{4}$. From Fig. 5, we also observe that higher density ratios and sampling ratios result in higher peak memory usage.

In summary, the sampling ratio ω governs the trade-off between accuracy and memory usage. Larger values of ω improve performance by sampling more embedding vectors but increase memory cost, while smaller values reduce memory usage but risk overly random regrowth. As shown in Table 5, performance drops significantly when $\omega < \frac{h}{2}$ across all three datasets. Thus, setting $\omega = \frac{h}{2}$ offers a robust balance between memory efficiency and recommendation quality. Furthermore, since edge devices often have heterogeneous memory budgets, ω can be flexibly tuned to match available resources without catastrophic performance degradation.

Table 5 Comparison of peak memory usage and recommendation performance between DSL, PEP, and SparseRec with different sampling ratios ω. The sampling ratios are indicated by the subscripts of SparseRec

d	Methods	Gowalla			Yelp			Movielens 10M		
		Memory	R@20	N@20	Memory	R@20	N@20	Memory	R@20	N@20
0.0625	DSL	38.54	15.95	12.48	37.93	7.66	6.51	50.94	28.58	36.95
	PEP	108.81	16.30	12.64	107.08	7.50	6.29	122.24	28.17	35.92
	SparseRec_{h}	37.48	17.27	13.61	36.89	8.31	6.99	42.11	29.39	37.85
	$\text{SparseRec}_{h/2}$	21.01	17.24	13.60	20.68	8.22	6.97	23.61	29.33	37.83
	$\text{SparseRec}_{h/4}$	12.77	17.22	13.54	12.57	8.16	6.95	14.35	28.93	37.36
	$\text{SparseRec}_{h/8}$	8.66	16.94	13.23	8.52	8.11	6.89	9.73	28.86	37.27
	$\text{SparseRec}_{h/16}$	6.60	16.50	12.87	6.49	7.95	6.75	7.41	28.66	37.11
0.125	DSL	40.80	18.60	14.67	40.16	9.03	7.67	45.84	29.13	37.76
	PEP	108.81	18.38	14.25	107.08	8.78	7.35	122.24	28.17	35.95
	SparseRec_{h}	38.83	19.00	14.93	38.21	9.27	7.93	43.62	29.84	38.48
	$\text{SparseRec}_{h/2}$	23.95	18.90	14.90	23.58	9.22	7.89	26.91	29.79	38.45
	$\text{SparseRec}_{h/4}$	16.52	18.87	14.76	16.25	9.17	7.80	18.55	29.58	38.28
	$\text{SparseRec}_{h/8}$	12.80	18.60	14.51	12.59	9.10	7.67	14.38	29.30	37.90
	$\text{SparseRec}_{h/16}$	10.94	18.13	14.11	10.76	8.84	7.48	12.29	29.26	37.83
0.25	DSL	45.34	20.18	15.84	44.62	10.02	8.48	43.29	29.34	38.13
	PEP	108.81	20.05	15.66	107.08	10.06	8.42	122.24	28.78	36.93
	SparseRec_{h}	41.96	20.79	16.34	41.29	10.34	8.84	47.14	30.22	39.00
	$\text{SparseRec}_{h/2}$	30.07	20.57	16.21	29.58	10.29	8.77	33.77	30.14	38.97
	$\text{SparseRec}_{h/4}$	24.10	20.44	16.12	23.72	10.16	8.61	27.08	30.09	38.96
	$\text{SparseRec}_{h/8}$	21.12	20.12	15.77	20.79	10.02	8.51	23.73	23.88	38.64
	$\text{SparseRec}_{h/16}$	19.64	19.64	15.41	19.33	9.71	8.24	22.06	29.74	38.57

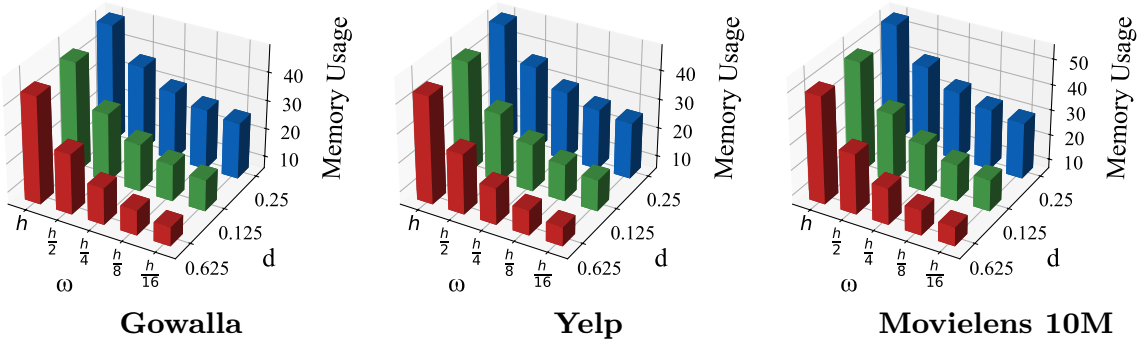


Fig. 5 Hyerparameter analysis of the sampling ratio ω w.r.t. the peak memory usage in MBs on Gowalla, Yelp and Movielens 10M

6 Conclusion and Future Research

Dense embedding tables limit the memory efficiency and scalability of recommender systems in resource-constrained environments. To address over-parameterization and reduce memory usage, we introduce SparseRec, which avoids dense gradients and enhances embedding-pruning methods by overcoming DST limitations. SparseRec uses NMF for sparse embedding table initialization and employs cumulative gradients for selective parameter regrowth. It calculates sparse gradients for active and sampled parameters during backward passes. Experiments on three real-world datasets with three base recommenders demonstrate SparseRec's effectiveness, demonstrating its suitability for deployment on memory-constrained edge devices. In future research, we plan to extend SparseRec with hardware-level evaluations to quantify its impact on latency and energy efficiency, providing a more comprehensive assessment of its suitability for on-device deployment.

Acknowledgements We would like to thank the Australian Research Council for the funding they provide.

Author Contributions Yunke Qu: conceptualization, methodology, software, investigation, writing – original draft, visualization. Liang Qu: writing - review & editing, conceptualization. Tong Chen: conceptualization, methodology, supervision, investigation Xiangyu Zhao: writing - review & editing. Jianxin Li: writing - review & editing. Hongzhi Yin: writing - review & editing, funding acquisition, supervision, resources.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions. The Australian Research Council partially supports this work under the streams of Future Fellowship (Grant No. FT210100624), Discovery Early Career Researcher Award (Grants No. DE230101033), the Discovery Project (Grant No. DP2401011081 and DP240101814), and the Linkage Projects (Grant No. LP230200892 and LP240200546).

Availability of data and materials The experiments are conducted on publicly available datasets: Gowalla (available at https://snap.stanford.edu/data/loc-Gowalla.html), Yelp (available at https://www.yelp.com/dataset), and Movielens 10M (https://grouplens.org/datasets/movielens/10m/). The code is available at https://anonymous.4open.science/r/SparseRec-7698/.

Declarations

Competing interests The authors declare that they have no conflict of interest.

References

1. Xia X, Yu J, Wang Q, Yang C, Hung NQV, Yin H (2023) Efficient on-device session-based recommendation. TIIS. 41(4)
2. Xia X, Yin H, Yu J, Wang Q, Xu G, Nguyen QVH (2022) On-device next-item recommendation with self-supervised knowledge distillation. In: SIGIR, pp. 546–555
3. Wang Q, Yin H, Chen T, Huang Z, Wang H, Zhao Y, Viet Hung NQ (2020) Next point-of-interest recommendation on resource-constrained mobile devices. In: WWW, pp. 906–916
4. Addula SR, Sekhar Sajja G (2024) Automated machine learning to streamline data-driven industrial application development. In: 2024 Sec Intl Conf Comput Charact Techniques Eng Sci (IC3TES), pp. 1–4
5. Zheng R, Qu L, Chen T, Zheng K, Shi Y, Yin H (2024) Personalized elastic embedding learning for on-device recommendation. TKDE 36:3363–3375
6. Li S, Guo H, Tang X, Tang R, Hou L, Li R, Zhang R (2024) Embedding compression in recommender systems: a survey. ACM Comput Surv 56(5):1–21

7. He X, Zhao K, Chu X (2021) Automl: a survey of the state-of-the-art. Knowl Based Syst 212:106622
8. Shen Z, Zhang Y, Wei L, Zhao H, Yao Q (2024) Automated Machine Learning: From Principles to Practices
9. Liang X, Chen T, Nguyen QVH, Li J, Yin H (2023) Learning compact compositional embeddings via regularized pruning for recommendation. In: ICDM, pp. 378–387
10. Liang X, Chen T, Cui L, Wang Y, Wang M, Yin H (2024) Lightweight embeddings for graph collaborative filtering. In: SIGIR
11. Kang W-C, Cheng DZ, Yao T, Yi X, Chen T, Hong L, Chi EH (2021) Learning to embed categorical features without embedding tables for recommendation. In: KDD, pp. 840–850
12. Shi H-JM, Mudigere D, Naumov M, Yang J (2020) Compositional embeddings using complementary partitions for memory-efficient recommendation systems. In: KDD, pp. 165–175
13. Liu H, Zhao X, Wang C, Liu X, Tang J (2020) Automated embedding size search in deep recommender systems. In: SIGIR, pp. 2307–2316
14. Qu Y, Chen T, Zhao X, Cui L, Zheng K, Yin H (2023) Continuous input embedding size search for recommender systems. In: SIGIR, pp. 708–717
15. Qu Y, Qu L, Chen T, Xiangyu Z, Hung NQV, Yin H (2024) Scalable dynamic embedding size search for streaming recommendation. In: CIKM
16. Liu S, Gao C, Chen Y, Jin D, Li Y (2021) Learnable embedding sizes for recommender systems. In: ICLR
17. Wang S, Sui Y, Wu J, Zheng Z, Xiong H (2024) Dynamic sparse learning: A novel paradigm for efficient recommendation. In: WSDM, pp. 740–749
18. Wang Y, Sui Y, Wang X, Liu Z, He X (2022) Exploring lottery ticket hypothesis in media recommender systems. IJIS 37(5):3006–3024
19. Qu Y, Chen T, Nguyen QVH, Yin H (2024) Budgeted embedding table for recommender systems. In: WSDM, pp. 557–566
20. Peng H, Gurevin D, Huang S, Geng T, Jiang W, Khan O, Ding C (2022) Towards sparsification of graph neural networks. In: 2022 IEEE 40th Intl Conf on Computer Design, pp. 272–279
21. Evci U, Gale T, Menick J, Castro PS, Elsen E (2020) Rigging the lottery: making all tickets winners. In: ICML, pp. 2943–2952
22. Mocanu DC, Mocanu E, Stone P, Nguyen PH, Gibescu M, Liotta A (2018) Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. Nat Commun 9(1):2383
23. Lee N, Ajanthan T, Torr P (2019) Snip: Single-shot network pruning based on connection sensitivity. In: ICLR
24. Wang C, Zhang G, Grosse R (2020) Picking winning tickets before training by preserving gradient flow. In: ICLR
25. Tanaka H, Kunin D, Yamins DLK, Ganguli S (2020) Pruning neural networks without any data by iteratively conserving synaptic flow. In: NeurIPS
26. Heddes M, Srinivasa N, Givargis T, Nicolau A (2024) Always-Sparse Training by Growing Connections with Guided Stochastic Exploration
27. Yin H, Qu L, Chen T, Yuan W, Zheng R, Long J, Xia X, Shi Y, Zhang C (2024) On-Device Recommender Systems: A Comprehensive Survey
28. He X, Zhang H, Kan M-Y, Chua T-S (2016) Fast matrix factorization for online recommendation with implicit feedback. In: SIGIR, pp. 549–558
29. Hu Y, Koren Y, Volinsky C (2008) Collaborative filtering for implicit feedback datasets. In: ICDM, pp. 263–272
30. Koren Y, Rendle S, Bell R (2021) Advances in collaborative filtering. Recommender syst handbook, 91–142
31. He X, Liao L, Zhang H, Nie L, Hu X, Chua T-S (2017) Neural collaborative filtering. In: WWW, pp. 173–182
32. Wang X, He X, Wang M, Feng F, Chua T-S (2019) Neural graph collaborative filtering. In: SIGIR, pp. 165–174
33. He X, Deng K, Wang X, Li Y, Zhang Y, Wang M (2020) Lightgcn: Simplifying and powering graph convolution network for recommendation. In: SIGIR, pp. 639–648
34. Mao K, Zhu J, Xiao X, Lu B, Wang Z, He X (2021) Ultragcn: Ultra simplification of graph convolutional networks for recommendation. In: CIKM, pp. 1253–1262
35. Hu J, Hooi B, Qian S, Fang Q, Xu C (2022) Mgdcf: distance learning via markov graph diffusion for neural collaborative filtering. TKDE 36:3281–3296
36. Yin H, Cui B, Huang Z, Wang W, Wu X, Zhou X (2015) Joint modeling of users' interests and mobility patterns for point-of-interest recommendation. In: MM, pp. 819–822
37. Wang Q, Yin H, Chen T, Huang Z, Wang H, Zhao Y, Viet Hung NQ (2020) Next point-of-interest recommendation on resource-constrained mobile devices. In: WWW, pp. 906–916
38. Yin H, Cui B (2016) Spatio-Temporal Recommendation in Social Media. 1st edn. Springer
39. Zhang C, Shi T, Zhang X, Zheng Y, Xie R, Liu Q, Xu J, Wen J-R (2024) QAGCF: Graph Collaborative Filtering for Q &A Recommendation
40. Li J, Wang M, Li J, Fu J, Shen X, Shang J, McAuley J (2023) Text is all you need: Learning language representations for sequential recommendation. In: KDD, pp. 1258–1267
41. Zhang C, Chen S, Zhang X, Dai S, Yu W, Xu J (2024) Reinforcing long-term performance in recommender systems with user-oriented exploration policy. In: SIGIR, pp. 1850–1860
42. Wang S, Wang Y, Tang J, Shu K, Ranganath S, Liu H (2017) What your images reveal: Exploiting visual contents for point-of-interest recommendation. In: WWW, pp. 391–400
43. Lee J, Abu-El-Haija S, Varadarajan B, Natsev A (2018) Collaborative deep metric learning for video understanding. In: KDD, pp. 481–490
44. Zhang C, Shi T, Zhang X, Liu Q, Xie R, Xu J, Wen J-R (2024) Modeling Domain and Feedback Transitions for Cross-Domain Sequential Recommendation
45. Wang Y, Li Z, Zhang C, Chen S, Zhang X, Xu J, Lin Q (2024) Do not wait: Learning re-ranking model without user feedback at serving time in e-commerce. In: RecSys, pp. 896–901
46. Chen H, Yin H, Chen T, Wang W, Li X, Hu X (2022) Social boosted recommendation with folded bipartite network embedding. TKDE 34(2):914–926
47. Pawar PP, Femy FF, Rajkumar N, Jeevitha S, Bhuvanesh A, DK (2025) Blockchain-enabled cybersecurity for iot using elliptic curve cryptography and black winged kite model. Intl J Info Tech
48. Liu S, Ha DS, Shen F, Yi Y (2021) Efficient neural networks for edge devices. Comput Electr Eng 92:107121
49. Wang X, Jia W (2025) Optimizing Edge AI: A Comprehensive Survey on Data, Model, and System Strategies
50. Janowsky SA (1989) Pruning versus clipping in neural networks. Phys Rev A 39:6600–6603
51. Ström N (1997) Sparse connection and pruning in large dynamic artificial neural networks. In: Proc 5th European Conf on Speech Commn Tech, pp. 2807–2810
52. Thimm G, Fiesler E (1995) Evaluating pruning methods. In: Proc of the Intl Symp Artif Neural Netw, pp. 20–25
53. Molchanov P, Mallya A, Tyree S, Frosio I, Kautz J (2019) Importance estimation for neural network pruning. In: CVPR, pp. 11264–11272
54. LeCun Y, Denker J, Solla S (1989) Optimal brain damage. In: Touretzky D. (ed.) NeurIPS, 2: 598–605
55. Mozer MC, Smolensky P (1989) Skeletonization: A technique for trimming the fat from a network via relevance assessment. In: NeurIPS, pp. 107–115

56. Molchanov P, Tyree S, Karras T, Aila T, Kautz J (2017) Pruning convolutional neural networks for resource efficient inference. In: ICLR
57. Frankle J, Carbin M (2019) The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: ICLR
58. Han S, Pool J, Tran J, Dally WJ (2015) Learning both weights and connections for efficient neural networks. In: NeurIPS, pp. 1135–1143
59. Dettmers T, Zettlemoyer L (2019) Sparse Networks from Scratch: Faster Training without Losing Performance
60. Wu C, Xu Y (2020) Greedy coordinate descent method on non-negative quadratic programming. In: 2020 IEEE 11th Sensor Array and Multichannel Sig Proc Workshop (SAM), pp. 1–5. IEEE, ???
61. Yu J, Xia X, Chen T, Cui L, Hung NQV, Yin H (2024) Xsimgcl: towards extremely simple graph contrastive learning for recommendation. TKDE 36(2):913–926
62. Qu Y, Qu L, Chen T, Nguyen QVH, Yin H (2025) Efficient multimodal streaming recommendation via expandable side mixture-of-experts. In: CIKM
63. Rendle S, Freudenthaler C, Gantner Z, Schmidt-Thieme L (2009) Bpr: Bayesian personalized ranking from implicit feedback. In: Proc of the 25th Conf on Uncertainty in Artif Intell, pp. 452–461
64. Kingma DP (2014) Adam: A method for stochastic optimization. arXiv:1412.6980
65. Liu S, Yin L, Mocanu D, Pechenizkiy M (2021) Do we actually need dense over-parameterization? in-time over-parameterization in sparse training. In: ICML, pp. 6989–7000