Dynamic Embedding Size Search with Minimum Regret for Streaming Recommender System

Bowei He
City University of Hong Kong
Hong Kong SAR
boweihe2-c@my.cityu.edu.hk

Xu He
Huawei Noah's Ark Lab
Shenzhen, China
hexu27@huawei.com

Renrui Zhang
The Chinese University of Hong Kong
Hong Kong SAR
zhangrenrui@pjlab.org.cn

Yingxue Zhang
Huawei Noah's Ark Lab Montreal
Montreal, Canada
yingxue.zhang@huawei.com

Ruiming Tang
Huawei Noah's Ark Lab
Shenzhen, China
tangruiming@huawei.com

Chen Ma*
City University of Hong Kong
Hong Kong SAR
chenma@cityu.edu.hk

ABSTRACT

With the continuous increase of users and items, conventional recommender systems trained on static datasets can hardly adapt to changing environments. The high-throughput data requires the model to be updated in a timely manner for capturing the user interest dynamics, which leads to the emergence of streaming recommender systems. Due to the prevalence of deep learning-based recommender systems, the embedding layer is widely adopted to represent the characteristics of users, items, and other features in low-dimensional vectors. However, it has been proved that setting an identical and static embedding size is sub-optimal in terms of recommendation performance and memory cost, especially for streaming recommendations. To tackle this problem, we first rethink the streaming model update process and model the dynamic embedding size search as a bandit problem. Then, we analyze and quantify the factors that influence the optimal embedding sizes from the statistics perspective. Based on this, we propose the **D**ynamic **E**mbedding **S**ize **S**earch (**DESS**) method to minimize the embedding size selection regret on both user and item sides in a non-stationary manner. Theoretically, we obtain a sublinear regret upper bound superior to previous methods. Empirical results across two recommendation tasks on four public datasets also demonstrate that our approach can achieve better streaming recommendation performance with lower memory cost and higher time efficiency.

CCS CONCEPTS

• **Information systems → Recommender systems**.

KEYWORDS

Streaming recommender system; Embedding size search; Contextual bandit; Automated machine learning

*Corresponding author

1 INTRODUCTION

Recommender systems (RS) have been widely adopted to reduce information overload and satisfy users' diverse needs. Considering the ever-increasing users and items, as well as users' continuous interest shift, conventional static RS, however, can hardly adapt to the evolving environment. To tackle such challenges, streaming recommender systems (SRS), whose recommendation strategy updates in a dynamic manner, was proposed in the last decade along with the rapid accumulation of massive data from online applications like Google and Twitter [5, 7, 11, 13, 17, 38]. Recently, deep learning-based recommender systems [19, 49] make a breakthrough in improving the recommendation performance, which provides a new direction for the evolution of streaming recommender systems.

To enable the success of deep RS, embedding learning plays a central role in representing users, items, and other features in low-dimensional vectors. Due to the inherent characteristics of different users and items, the conventional design that assigns an identical embedding size to each user or item limits the model performance and brings huge memory costs. To solve this problem, embedding size search is introduced in [23] accompanied by the rapid development of the automated neural network structure design in computer vision and natural language processing tasks [4, 28, 32, 36, 48]. Recently, embedding size search has received widespread attention [15, 23, 24, 30], and various methods have been proposed to search for ID-specific embedding sizes. Most of them adopt an external controller or the differential search to decide the embedding sizes from a discrete or continuous candidate space [9, 23, 29, 31, 33, 52]. Due to the broad application of streaming recommender systems, it has also been noticed that assigning a constant embedding size for users or items along the timeline will lead to unsatisfactory performance. Correspondingly, the dynamic embedding size search problem has also been gradually investigated and several methods [28, 29, 41, 51] have been proposed to search the best time-varying embedding sizes.

Although the aforementioned approaches are effective and insightful, there are still several avenues for improvement. First, previous embedding size search methods [28, 29, 41, 52] with neural network-based or reinforcement learning-based controllers require large amounts of interaction data to converge, which drags down their effectiveness in the early stages of the recommendation stream. Moreover, the neural network-based method can hardly balance the exploration and exploitation during the online embedding size

search. Second, existing methods [23, 28, 29, 41, 52] only consider the browsing frequency throughout the whole historical record when deciding the appropriate embedding sizes. Nevertheless, this is far from reflecting users' recent browsing behavior characteristics. In fact, the optimal embedding size is mainly associated with the information amount of users' browsing records, which should be determined by both the frequency and diversity of browsing records. Third, in previous works [28, 29, 41, 52], the model update process still follows the conventional update pattern of static models, which optimizes the model on the training set until convergence and then evaluating it on the untimely test set. However, in a real SRS like Twitter, the system needs to recommend content to users according to their real-time interests when they post tweets, which means training and testing should alternate over time. Therefore, the optimization objective of dynamic embedding size search should consider the model's performance at each timestep throughout the whole timeline. Moreover, existing methods often suffer from huge memory costs and low time efficiency which make it difficult to put them into practical use.

To address these issues, we rethink the streaming recommendation model update process, and first model the dynamic optimal embedding size search as a cumulative regret minimization bandit problem. Then, we justify the change of the embedding size by analyzing and quantifying the user behavior characteristics via two indicators proposed in this paper. On this basis, we propose the non-stationary LinUCB bandit-based **D**ynamic **E**mbedding **S**ize **S**earch (**DESS**) algorithm to minimize the performance regret. In the method design, the weighted forgetting mechanism is adopted to reduce interference from outdated data and help the search policy pay more attention to recent user behaviors. We provide a sublinear dynamic regret upper bound which guarantees the effectiveness of our method. To help validate the superiorities of our approach, we design an embedding size adaptive neural network based on Neural Collaborative Filtering [45], whose *embedding input* part can be shared among different base recommendation models. Following previous works [28, 29, 51], we conduct the top-k recommendation and rating score prediction tasks on four public recommendation datasets. The experimental results demonstrate the effectiveness of our method with lower memory cost and higher time efficiency.

To summarize, the main contributions of this work are as follows:

- We model the dynamic embedding size search as the cumulative regret minimization bandit and propose a non-stationary LinUCB-based algorithm **DESS** with a sublinear regret upper bound.
- We provide two effective indicators *IND* and *POD* reflecting user behavior characteristics that can guide the search for appropriate embedding sizes from a statistical perspective.
- Experiments on four real-world datasets show that **DESS** outperforms the state-of-the-art methods in dynamic embedding size search for streaming recommender systems.

2 RELATED WORK

Streaming Recommender System. SRS is a kind of newly developed recommender system in coping with the high -throughput user data and their incremental properties [6, 7, 11, 12, 17, 38, 44].Different from the conventional recommender systems [8, 19,

49], SRS needs to update its recommendation strategy in a dynamic manner to catch the user interest temporal dynamics. Early works [5, 39], known as memory-based methods, leverage similarity relationships in aggregated historical data to predict ratings. Some more advanced works [6, 12, 13]propose to extend popular static recommender models like collaborative filtering and matrix factorization to the streaming fashion. However, most previous methods suffer from a common drawback: dividing the entire data stream into two segments, training on the former segment until convergence, and then testing on the latter segment, which is far from the real SRS scenario. In this work, we split the data of each time slice to two parts, train on the first part and test on the second part alternatively along the timeline, in such way to better fit the real streaming recommendation task.

Embedding Size Search. The embedding size search problem gradually attracts much attention because of the large models' increasing memory cost [1, 10] and the rapid development of neural architecture search techniques [4, 28, 32, 36, 48]. Some initial works focus on embedding size search on static recommender systems [15, 23, 24, 30, 33]. These approaches break the long-standing uniform-size embedding structure design. However, once determined, these embedding sizes can no longer be dynamically adjusted. As streaming recommender systems are more and more adopted, some recent works [29, 41, 51, 52, 52] start to investigate the corresponding dynamic embedding size problem which means the embedding size for each ID is no more fixed. Generally speaking, previous methods can be divided into two mainstream schemes: *soft-selection* and *hard-selection*. In the *soft-selection* scheme [28, 51, 52], the input of following representation learning layers is actually the weighted summation of transformed vectors corresponding to each embedding size. However, this category of methods suffer from the cold-start problem and the excessive memory cost. In the *hard-selection* scheme [29, 41], only one size of embedding is selected at each time step. Nevertheless, previous works are still not reasonable enough for modeling the SRS update process. In addition, the algorithm performance lacks theoretical guarantee and needs to be improved if applied to the practical application. In this work, we follow the *hard-selection* scheme and provide a more efficient dynamic embedding size search method.

Base Recommendation Model. To make effective recommendations, a number of models have been proposed [34], including matrix factorization (MF)-based models, distance-based models, and multi-layer perceptron-based models. Matrix factorization [26] is a representative and widely-used recommendation method which applies an inner product between the user and item embeddings to capture the interactions between users and items. Distance-based models, generally, compute the Euclidean distance between users and items for capturing fine-grained user preference [20]. On the other hand, Neural Collaborative Filtering (NCF) [19], a type of multi-layer perceptron-based method, models user-item interactions through neural network architectures, so that high-level non-linearities within the user-item interaction can be learned. Following the settings in previous works [23, 29, 51, 52], we also choose NCF as the base recommendation model. Furthermore, we modify its architecture to fit the dynamic embedding size setting, which is detailed in Section 4.5. MF-based models and distance-based models are also explored to prove the wide applicability of our method.

3 PRELIMINARIES

In this section, we first formalize the streaming recommendation problem. Then we derive the dynamic embedding size search task from the streaming model update process.

3.1 Streaming Recommendation

One prominent advantage of SRS is that they can update and respond instantaneously for catching users' intentions and demands [5, 6]. Due to the high volume of online data, previous works [17, 29, 42, 43, 51] split the user-item interaction stream into short-term segments. Following this setting, we split the whole data stream of length L into T consecutive segments $D_1, ..., D_t, ..., D_T$ with the same length $|D_t|$ ($L = |D_t| \times T$). Each segment D_t is then divided into the training part D_t^{tr} and test part D_t^{te}. On this basis, the streaming recommendation task is formulated as: given $D_1^{tr}, D_2^{tr}, ..., D_t^{tr}, ..., D_T^{tr}$, it is supposed to train a model M to predict the user preference in $D_1^{te}, D_2^{te}, ..., D_t^{te}, ..., D_T^{te}$, where $D_t^{tr} \cup D_t^{te} = D_t$ and $D_t^{tr} \cap D_t^{te} = \emptyset$. The overall performance is evaluated by the average recommendation accuracy over the entire timeline.

3.2 Dynamic Embedding Size Search

With the wide use of deep recommendation models, embeddings are largely investigated to represent users, items, and other auxiliary features. However, the conventional design of setting identical and static embedding sizes suffers from the unsatisfying model prediction performance and the unacceptable memory cost. To solve these problems, many works [21, 23, 29, 41, 52] are proposed for the embedding size search. To make the search fit into the streaming scenario, a **streaming update process** should be first introduced: the model M inherits the parameters from the previous moment M_{t-1} and updates itself to M_t with the current training data D_t^{tr}. Following these, the dynamic embedding size search task can be further formulated: optimizing embedding size search policies π_{SE}^u and π_{SE}^i which accordingly control user and item embedding sizes of recommendation model M at each timestep, so that the overall model performance can be satisfying (see Figure 1). For the ease of illustration, π_{SE} refers to both π_{SE}^u and π_{SE}^i.

4 METHODOLOGY

In this section, we introduce our approach for dynamic embedding size search in streaming recommender systems. First, we model the embedding size search as a bandit problem and formalize the objective. Then we analyze and quantify the characteristics that can determine optimal embedding sizes from a statistical perspective. Next, we elaborate the non-stationary LinUCB-based **DESS** method (shown in Algorithm 1) to conduct the dynamic embedding size search, and provide the corresponding theoretical guarantee analysis. Finally, we introduce the structure of our streaming recommendation model—an embedding size adaptive neural network.

4.1 Embedding Size Search as Bandits

The target of the embedding size search in streaming scenarios is to optimize the average/cumulative model performance by selecting appropriate embedding sizes at different timesteps. From the temporal view, the search process is, in nature, a sequence of size

value decisions according to data characteristics. To solve this sequential decision-making problem, Multi-armed Bandits (MAB) are a promising approach where a fixed limited set of resources must be allocated between competing choices in a way that maximizes the sum of rewards earned through a sequence of lever pulls [27]. Therefore, to consider the model's recommendation performance at each timestep, we model the dynamic embedding size search as a bandit problem, and the objective of the search policy π_{SE} is to minimize the expected dynamic **pseudo-regret** defined as:

$$\bar{R}_T = \max_{\pi:C\to 1,...,K} \mathbb{E}[\sum_{t=1}^{T} \mathcal{L}_t^{te}(\pi_{SE}) - \sum_{t=1}^{T} \mathcal{L}_t^{te}(\pi)], \qquad (1)$$

where $\mathcal{L}_t^{te}(\pi_{SE})$ is the batch loss of M with embedding sizes determined by search policy π_{SE} on test data D_t^{te}. $\mathcal{L}_t^{te}(\pi)$ is the model test loss received from pulling the arm that an arbitrary policy π recommends at the current state. C is the set of context information that can help the policy π select the best arm from the arm candidates $1, ..., K$ at different timesteps. Note that the ideal $\bar{R}_T$ is obtained when π is the optimal one. Thus, the pseudo regret for π_{SE} is the difference between the actual loss it incurs and the loss incurred by the best possible embedding size search policy [27].

Since the test data is actually inaccessible in the training phase, we utilize the validation data to interact with the bandit directly and update the search policy. Especially, in the streaming recommendation scenario, the union of training data and test data at the last timestep $t-1$ (D_{t-1}) can be regarded as the validation data for timestep t [29, 51]. Then, the corresponding dynamic **regret** is:

$$R_T = \max_{\pi:C\to 1,...,K} \sum_{t=1}^{T} \sum_{j=1}^{|D_{t-1}|} r_{t,j}^{va}(\pi(\mathbf{x})) - \sum_{t=1}^{T} \sum_{j=1}^{|D_{t-1}|} r_{t,j}^{va}(\pi_{SE}(\mathbf{x})), \quad (2)$$

where $r_{t,j}^{va}(\pi(\mathbf{x}))$ is the reward received from pulling the arm that the π recommends in the validation phase. $r_{t,j}^{va}(\pi_{SE}(\mathbf{x}))$ is the reward actually received by our π_{SE} on validation data. The $\mathbf{x}$ indicates the context as the input of the bandit model, whose details will be provided in Section 4.2. Here, we further explain the reward $r_{t,j}$, noted as $r_l (l = (t-1) \times |D_t| + j, 1 \le l \le L)$ and the arm a.

Reward. The reward r_l is defined based on the performance $\mathcal{L}_l^{new}$ of the temporarily updated embedding structure by π_{SE} and the previous structure's performance $\mathcal{L}_l^{old}$ on l-th interaction of the validation data stream. To fairly compare the effectiveness of such two embedding structures with different sizes, we temporarily tune their parameters with the l-th interaction. r_l here is designed as a binary variable and can only be 0 or 1. The formula is following:

$$r_l = \begin{cases} 1, & if \quad \mathcal{L}_l^{old} - \mathcal{L}_l^{new} > threshold \\ 0, & if \quad \mathcal{L}_l^{old} - \mathcal{L}_l^{new} < threshold \end{cases} \qquad (3)$$

In the real-world application, r_l can be designed as a continuous real number $\mathcal{L}_l^{old} - \mathcal{L}_l^{new}$ for performance optimization.

Arm. The arms a here are actually different embedding size candidates or other embedding size adjustment operations, like increasing or decreasing embedding sizes.

4.2 Embedding Size Indicator

To effectively determine the sizes of embeddings, the indicator to increase or decrease embedding sizes is worth exploring because only

Figure 1: Illustration of the dynamic embedding size search process. At each timestep, the embedding layer outputs embedding vectors with sizes determined by the π_{SE}. After transformation, they will be input into Neural CF layers for further inference.

browsing frequency is not sufficient to prompt the search policy to make the correct decision. According to [2, 14, 22], a larger data dispersion degree indicates a greater amount of information the data contains. Thus, when the data dispersion degree is large, the size of the embedding should be large to represent the whole historical information. Motivated by this, we follow a similar fashion to quantify the amount of information in historical data by leveraging the explicit item features independent of the user-item interactions. Assume we have a set of raw feature vectors $\mathbf{F}_1, \mathbf{F}_2, .., \mathbf{F}_N$ corresponding to each item. Till l-th interaction of the data stream, let user u have rated a subset of the items indexed by $i_1, i_2, ..., i_H$, then the interest diversity of user u can be formulated by the centroid diameter distance:

$$IND_l^u = \frac{1}{H} \sum_{h=1}^{H} \sqrt{(\mathbf{F}_{i_h} - \mathbf{Q}_l^u)(\mathbf{F}_{i_h} - \mathbf{Q}_l^u)^\top}, \qquad (4)$$

where $\mathbf{Q}_l^u$ is the mean vector of $\mathbf{F}_{i_1}, \mathbf{F}_{i_2}, ..., \mathbf{F}_{i_H}$ and represents the user u's mean interest. For item i, assume it has been rated by users $u_1, u_2, .., u_H$ till l-th interaction, the diversity of its property is:

$$POD_l^i = \frac{1}{H} \sum_{h=1}^{H} \sqrt{(\mathbf{Q}_l^{u_h} - \mathbf{P}_l^i)(\mathbf{Q}_l^{u_h} - \mathbf{P}_l^i)^\top}, \qquad (5)$$

where $\mathbf{P}_l^i$ is the mean vector of $\mathbf{Q}_l^{u_1}, \mathbf{Q}_l^{u_2}, ..., \mathbf{Q}_l^{u_H}$ and represents item i's mean property. In this way, we define the context $\mathbf{x}_l$ for the user embedding size search policy π_{SE}^u as the combination of frequency and information diversity (FRE_l^u, IND_l^u), where FRE_l^u is the occurrence number of user u in historical data. The context for the item embedding size search policy π_{SE}^i is defined as (FRE_l^i, POD_l^i) similarly, where FRE_l^i is the occurrence of item i in historical data.

4.3 Non-stationary LinUCB-based Search Policy

Despite the effectiveness of linear MAB, some recent works [25, 37, 47, 50] focus on a more general setting: the constraint that requires fixed optimal regression parameters $\boldsymbol{\theta}^*$ is relaxed, which is

more suitable for our scenario. To better balance the exploration and exploitation in such a setting, we design our non-stationary LinUCB-based **DESS** algorithm for dynamic embedding size search. Its effectiveness on memory cost and time efficiency will be elaborated in Section 5. Due to the fact that the accumulated historical data for each user and item can only become richer and richer as data streams in, we simplify the embedding size search as a binary selection problem, where π_{SE} only needs to decide if increasing the embedding size to the subsequent larger size candidate.

The algorithm details are described in Algorithm 1. Generally, the whole algorithm is separated to two parts: *Updating Non-stationary LinUCB-based Search Policy π_{SE}* and *Updating recommendation model*, which are executed one after the other at each timestep t. Different arms a in our method share the common context information x_l about the frequency and interest/property diversity: $\mathbf{x}_{l,1} = \mathbf{x}_{l,2} = ... = \mathbf{x}_{l,K} = \mathbf{x}_l$. Based on the assumption that the expected payoff r_l is linear to its context $\mathbf{x}_l \in \mathbb{R}^d$, we set disjoint linear reward models $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, ..., \boldsymbol{\theta}_K$ for corresponding arms $1, 2..., K$ to estimate rewards when selecting different arms. Note that in this paper, $\boldsymbol{\theta}_{l,a}$ indicates the parameter of $\boldsymbol{\theta}_a$ at the l-th user-item interaction. For such disjoint linear models, we use ridge regression [35] to solve them. And the objective is to minimize the regularized weighted residual sum of squares, thus the $\hat{\boldsymbol{\theta}}_{l,a}$ is defined as follows:

$$\underset{\boldsymbol{\theta} \in \mathbb{R}^d}{argmin} (\sum_{s=1}^{l} \mathbb{1}(a_s = a)\gamma^{l-s}(r_s - \langle \mathbf{x}_{s,a}, \boldsymbol{\theta} \rangle)^2 + \lambda \|\boldsymbol{\theta}\|_2^2), \qquad (6)$$

where $\langle , \rangle$ indicates the inner product operation, and a_s is the arm selected by the π_{SE} at s-th interaction $(1 \leq s \leq l)$. Eq. 6 is actually the regularized weighted least-squared estimator of $\boldsymbol{\theta}_a^*$ at l-th interaction. The conduct of the weighted forgetting mechanism (discount factor γ) is to reduce the interference from the outdated data [3, 37, 47] and help π_{SE} pay more attention to the recent

user/item behaviors. Furthermore, we have the solution for Eq. 6:

$$\hat{\theta}_{l,a} = \mathbf{V}_{l,a}^{-1} \sum_{s=1}^{l} \mathbb{1}(a_s = a)\gamma^{l-s}\mathbf{x}_{s,a}r_s,$$

$$where \quad \mathbf{V}_{l,a} = \sum_{s=1}^{l} \mathbb{1}(a_s = a)\gamma^{l-s}\mathbf{x}_{s,a}\mathbf{x}_{s,a}^{\top} + \lambda\mathbf{I}_d, \tag{7}$$

and $\mathbf{I}_d$ denotes the d-dimensional identity matrix. Here, similar to $\mathbf{V}_{l,a}^{-1}$, we define a matrix $\widetilde{\mathbf{V}}_{l,a}$ as an intermediate variable in our algorithm to help obtain the confidence ellipsoid:

$$\widetilde{\mathbf{V}}_{l,a} = \sum_{s=1}^{l} \mathbb{1}(a_s = a)\gamma^{2(l-s)}\mathbf{x}_{s,a}\mathbf{x}_{s,a}^{\top} + \lambda\mathbf{I}_d, \tag{8}$$

which is strongly connected to the variance of the estimator $\hat{\theta}_{l,a}$. Applying the online version of ridge regression [25, 37, 50], the update formulations of $\mathbf{V}_{a_l}, \widetilde{\mathbf{V}}_{a_l}, \hat{\theta}_{a_l}$ are shown as follows:

$$\mathbf{V}_{a_l} = \gamma\mathbf{V}_{a_l} + \mathbf{x}_{l,a_l}\mathbf{x}_{l,a_l}^{\top} + (1-\gamma)\lambda\mathbf{I}_d,$$
$$\widetilde{\mathbf{V}}_{a_l} = \gamma^2\widetilde{\mathbf{V}}_{a_l} + \mathbf{x}_{l,a_l}\mathbf{x}_{l,a_l}^{\top} + (1-\gamma^2)\lambda\mathbf{I}_d, \tag{9}$$
$$\mathbf{b}_{a_l} = \gamma\mathbf{b}_{a_l} + r_l\mathbf{x}_{l,a_l}, \hat{\theta}_{a_l} = \mathbf{V}_{a_l}^{-1}\mathbf{b}_{a_l},$$

where $\mathbf{b}$ is an intermediate variable to help compute $\hat{\theta}$. The initialization of $\mathbf{V}_a, \widetilde{\mathbf{V}}_a, \hat{\theta}_a$ for each arm a is provided in the **Initialize** part of Algorithm 1. During the algorithm execution, we use Eq. 9 to update such variables for the selected arm at each interaction.

Finally, we introduce how our non-stationary LinUCB-based policy π_{SE} selects the most promising arm. Following related works [25, 37, 47, 50], we first obtain the confidence value β_l (coefficient of confidence ellipsoid) which controls the exploration level:

$$\beta_l = \sqrt{\lambda}S + \sigma\sqrt{2\log(\frac{1}{\delta}) + d\log(1 + \frac{U^2(1-\gamma^{2l})}{\lambda d(1-\gamma^2)})}, \tag{10}$$

where S is the upper bound for parameters ($\forall l, a, \|\theta_{l,a}^*\|_2 \leq S$), U is the upper bound for contexts ($\forall l, a, \|\mathbf{x}_{l,a}\|_2 \leq U$), σ is the subgaussian constant, and δ is a pre-designated probability. Based on this, we can derive the upper confidence bound (UCB) which considers both the estimated reward and the uncertainty (confidence ellipsoid) of the reward estimation, in such a way to better balance the exploration and exploitation for arm selection:

$$UCB(a) = \mathbf{x}_{l,a}^{\top}\hat{\theta}_a + \beta_l\sqrt{\mathbf{x}_{l,a}^{\top}\mathbf{V}_a^{-1}\widetilde{\mathbf{V}}_a\mathbf{V}_a^{-1}\mathbf{x}_{l,a}}. \tag{11}$$

After computing $UCB(a)$ for each arm, π_{SE} will select the arm with the highest UCB score as the embedding size control command.

4.4 Theoretical Analysis

We provide the upper regret bound analysis for **DESS** in this section. As far as we know, this is the first theoretical regret bound for the non-stationary contextual linear bandit with disjoint arm-associated parameter vectors θ_a.

DEFINITION 1. *Parameter Variation Budget. For the search policy, the true oracle parameters $\{\theta_{l,a}^*\}_{l=1}^{L}$ are actually unknown. And their shifts can be quantified by the variation budget which*

Algorithm 1 Dynamic Embedding Size Search (DESS)

Input:
η (learning rate for recommender model), probability δ, subgaussianity constant σ, context dimension d, regularization λ, upper bound for contexts U, upper bound for parameters S, discount factor γ
Initialize: initial recommender model M_0,
$\mathbf{b}_a = \mathbf{0}_{\mathbb{R}^d}, \mathbf{V}_a = \lambda\mathbf{I}_d, \widetilde{\mathbf{V}}_a = \lambda\mathbf{I}_d, \hat{\theta}_a = \mathbf{0}_{\mathbb{R}^d}$ for each arm a,
user-item interaction data stream $\{(D_1^{tr}, D_1^{te}), ..., (D_T^{tr}, D_T^{te})\}$
which contains T segments of data in chronological order
Process:
1: **for** each $t = 1, 2, ..., T$ **do**
2:    /∗ Update Non-stationary LinUCB-based Policy π_{SE} ∗/
3:    Collect the last segment of data $(D_{t-1}^{tr}, D_{t-1}^{te})$
4:    **for** each interaction in $(D_{t-1}^{tr} \cup D_{t-1}^{te})$ **do**
5:       Receive context $\mathbf{x}_{l,a}$ for each arm a
6:       $\beta_l = \sqrt{\lambda}S + \sigma\sqrt{2\log(\frac{1}{\delta}) + d\log(1 + \frac{U^2(1-\gamma^{2l})}{\lambda d(1-\gamma^2)})}$
7:       $UCB(a) = \mathbf{x}_{l,a}^{\top}\hat{\theta}_a + \beta_l\sqrt{\mathbf{x}_{l,a}^{\top}\mathbf{V}_a^{-1}\widetilde{\mathbf{V}}_a\mathbf{V}_a^{-1}\mathbf{x}_{l,a}}$ for each a
8:       $a_l = argmax_{a\in\mathcal{A}}(UCB(a))$
9:       Temporarily change embedding sizes according to a_l
10:      Temporarily tune embedding parameters
11:      Input interaction into M_{t-1} and receive reward r_l
12:      **Updating:** $\mathbf{V}_{a_l} = \gamma\mathbf{V}_{a_l} + \mathbf{x}_{l,a_l}\mathbf{x}_{l,a_l}^{\top} + (1-\gamma)\lambda\mathbf{I}_d, \widetilde{\mathbf{V}}_{a_l} = \gamma^2\widetilde{\mathbf{V}}_{a_l} + \mathbf{x}_{l,a_l}\mathbf{x}_{l,a_l}^{\top} + (1-\gamma^2)\lambda\mathbf{I}_d, \mathbf{b}_{a_l} = \gamma\mathbf{b}_{a_l} + r_l\mathbf{x}_{l,a_l}, \hat{\theta}_{a_l} = \mathbf{V}_{a_l}^{-1}\mathbf{b}_{a_l}$
13:    **end for**
14:
15:    /∗ Update Recommendation Model M ∗/
16:    Collect the current segment of data (D_t^{tr}, D_t^{te})
17:    Output actions a_t from π_{SE}
18:    Permanently change the embedding sizes for user-item pairs in D_t^{tr} according to a_t
19:    Input D_t^{tr} to M_{t-1} and update the model to M_t
20:    Report the accuracy and test loss of M_t on test data D_t^{te}
21: **end for**

---

measures the magnitude of non-stationarity in the dynamical data stream. This can be defined as:

$$B_L^* := \sum_{l=1}^{L-1} \max_a \|\theta_{l+1,a}^* - \theta_{l,a}^*\|_2. \tag{12}$$

ASSUMPTION 1. *Variation Budget Upper Bound*
We assume that the variation budget is bounded by a known quantity B_L similar to the previous literature [3, 25, 37, 50], that is $B_L^* \leq B_L$.

ASSUMPTION 2. *Bounded Reward*
We assume that the reward r_l is bounded by the subgaussianity constant 2σ, $0 \leq r_l \leq 2\sigma$ which can be easily satisfied. For example, in the above algorithm description part, the σ can be set as 0.5.

LEMMA 1. *Let prediction error $Er(\mathbf{x}_{l,a}, \theta_{l,a}) = |\langle\mathbf{x}_{l,a}, \theta_{l,a}^*\rangle - \langle\mathbf{x}_{l,a}, \theta_{l,a}\rangle|$, $k = \sup_{\mathbf{x},\theta}\langle\mathbf{x}, \theta\rangle, c = \inf_{\mathbf{x},\theta}\langle\mathbf{x}, \theta\rangle$, and $D \in \mathbb{N}^*$. With probability at least $1 - \delta$: for all $a \geq 1$, the following holds:*

$$Er(\mathbf{x}_{l,a}, \theta_{l,a}) \le \frac{2k}{c}\beta_l \|\mathbf{x}_{l,a}\|_{V_{l,a}^{-1}} + \frac{2kU}{c}\sqrt{1 + \frac{L^2}{\lambda(1-\gamma)}}\left(\frac{2kSU^2}{\lambda}\frac{\gamma^D}{1-\gamma} + k\sqrt{\frac{d}{\lambda(1-\gamma)}}\sum_{s=l-D}^{l-1}\|\theta_{s,a}^* - \theta_{s+1,a}^*\|_2\right).$$

LEMMA 2. *Similar to [37], let* $\{\mathbf{x}_{s,a_l^*}\}_{s=1}^l$ *a sequence in* $\mathbb{R}^d$ *such that* $\|\mathbf{x}_{s,a_l^*}\|_2 \le U$ *for all* $s \in \mathbb{N}^*$. a_l^* *is the optimal arm that* π_{SE} *should select at l-th interaction. For* $l \ge 1$, *define* $V_{l,a_l^*} := \sum_{s=1}^l \mathbb{1}(a_s = a_l^*)\gamma^{l-s}\mathbf{x}_{s,a_l^*}\mathbf{x}_{s,a_l^*}^\top + \lambda\mathbf{I}_d$. *Given* $\lambda \ge 0$, *the following inequality holds:*

$$\sum_{l=1}^L \|\mathbf{x}_{l,a_l^*}\|_{V_{l,a_l^*}^{-1}} \le 2\max(1, L^2/\lambda)(dL\log(\frac{1}{\gamma}) + d\log(1 + \frac{U^2(1-\gamma^L)}{\lambda d(1-\gamma)})).$$

THEOREM 1. *Under the assumptions above, the regret of the **DESS** algorithm is bounded for all* $\gamma \in (0, 1)$ *and integer* $D \ge 1$, *with the probability at least* $1 - \delta$, *by*

$$R_L \le \sqrt{32\max(1, U^2/\lambda)}\frac{k}{c}\beta_L\sqrt{dL}\sqrt{L\log(\frac{1}{\gamma}) + \log(1 + \frac{U^2(1-\gamma^L)}{\lambda d(1-\gamma)})} + \frac{8k^2SU^3\gamma^D}{c\lambda(1-\gamma)}L + \frac{8k^2SU^4\gamma^D}{c\lambda^{\frac{3}{2}}(1-\gamma)^{\frac{3}{2}}}L + \frac{4k^2UD}{c\sqrt{\lambda}}\sqrt{\frac{d}{1-\gamma}}B_L + \frac{4k^2U^2D}{c\lambda}\frac{\sqrt{d}}{1-\gamma}B_L.$$

COROLLARY 1. *By choosing discount factor* $\gamma = 1 - (B_L/(\sqrt{d}L))^{2/5}$, *the regret of **DESS** algorithm is asymptotically upper bounded with high probability by a term* $O(d^{9/10}B_L^{1/5}L^{4/5})$ *when* $L \to \infty$.

The detailed proofs of theorem 1 and corollary 1 are provided in Appendix B.1 and B.2, respectively.

4.5 Embedding Size Adaptive Neural Network

4.5.1 Model Inference. As mentioned in Section 2, based on the NCF model [19, 23, 29], we design an *embedding size adaptive neural network* shown in Figure 2 as the streaming recommendation model M in Algorithm 1. Different from the conventional design that assigns one or a set of embedding sizes for each user or item in advance [19, 28, 29, 45, 49], the embedding size of each user or item can be selected flexibly from a group of size candidates at each timestep in our proposed structure. Suppose that the embedding size group for users and items are both $size = [s_0, s_1, .., s_n](s_0 < s_1... < s_n)$ for simplicity. Actually, the candidates for each user or item can be different. The initial size for each ID is set as the minimum value s_0 of the candidate group. $\{\mathbf{W}_{01}, \mathbf{b}_0, \mathbf{W}_{12}, \mathbf{b}_1, ..., \mathbf{W}_{n-1n}, \mathbf{b}_n\}$ is a sequence of linear transformation parameters that will unify the embedding size to s_n before inputting it to the *representation learning* part. Assume that the current embedding size for user u is s_i and the embedding vector is $\mathbf{E}_i^u$, the forward propagation process in *embedding input* part is: $\widehat{\mathbf{E}}_{i+1}^u = \mathbf{W}_{ii+1}\mathbf{E}_i^u + \mathbf{b}_i, .., \widehat{\mathbf{E}}_n^u = \mathbf{W}_{n-1n}\widehat{\mathbf{E}}_{n-1}^u + \mathbf{b}_{n-1}$. The embedding vector will be transformed into the s_n-dimensional space $\mathbb{R}^{s_n}$. Then, an additional batch normalization with Tanh activation is necessary to tackle the magnitude differences between inner-batch transformed embeddings $\widehat{\mathbf{E}}_n^u$ if processing a mini-batch:$\widehat{\mathbf{E}}_n^u = tanh\left(\frac{\widehat{\mathbf{E}}_n^u - \mu_B}{\sqrt{(\sigma_B^2)^2 + \epsilon}}\right)$, where μ_B is the mini-batch mean and σ_B^2 is the mini-batch variance. The batch size can be set as 1 when inferring a single sample. After executing a similar transformation on item i, we obtain the transformed user embedding $\widehat{\mathbf{E}}_n^u$ and item embedding $\widehat{\mathbf{E}}_n^i$ with the same dimension s_n. The following *representation learning* part is a sequence of BatchNorm, Linear,

and Tanh activation layers: $\mathbf{h}_1 = BatchNorm(cat(\widehat{\mathbf{E}}_n^u, \widehat{\mathbf{E}}_n^i)), \mathbf{h}_2 = BatchNorm(Linear(\mathbf{h}_1)), \hat{\mathbf{y}}_{ui} = Linear(Tanh(\mathbf{h}_2))$.

4.5.2 Embedding Warm Initialization. When receiving the increasing embedding size command from π_{SE}, there are two intuitive ways to initialize the embedding vector with the new embedding size: 1) zero initialization/random initialization, and 2) initialization with the information from previous embedding vectors. We take the second type of initialization and name it as *embedding warm initialization* (EWI). We perform a linear transformation sharing the parameters with above on the previous embedding $\mathbf{E}_i \in \mathbb{R}^{s_i}$ and obtain $\mathbf{E}_{i+1}$ in the s_{i+1}-dimensional space $\mathbb{R}^{s_{i+1}}$: $\mathbf{E}_{i+1} = \mathbf{W}_{ii+1}\mathbf{E}_i + \mathbf{b}_i$.

After the embedding warm initialization, the model inference starts from $\mathbf{E}_{i+1}$ and follows the forward propagation introduced in Section 4.5.1.

5 EXPERIMENTS

In this section, to comprehensively demonstrate the effectiveness of our method, we mainly focus on the following questions:

- **RQ1:** Does our method achieve better recommendation accuracy than the state-of-art methods along the timeline?
- **RQ2:** Does our method get sublinear regret on selecting embedding sizes, outperforming previous methods?
- **RQ3:** Whether our method consumes less computer memory compared with baseline methods?
- **RQ4:** Whether our method is more time-efficient than baselines for streaming recommendation?
- **RQ5:** Whether our method is applicable to different base recommendation models, like matrix factorization-based model and distance-based model?
- **RQ6:** Whether *Embedding Warm Initialization* technique contributes to the model performance improvement?

5.1 Experiment Setting

5.1.1 Datasets. We evaluate our method on four public recommender system datasets. The data and code will be released soon.

- **ml-20m [16]:** This dataset describes 5-star rating and free-text tagging activity from MovieLens, a movie recommendation service. It contains 20,000,263 ratings created by 138,493 users over 27,278 movies between January 09, 1995 and March 31, 2015.
- **ml-latest [16]:** This is a recently released dataset from MovieLens and contains 27,753,444 ratings by 283,228 users over 58,098 movies between January 09, 1995 and September 26, 2018.
- **Amazon-Books [18]:** This dataset contains book reviews from Amazon, including 22,507,154 ratings spanning May 1996 - July 2014. We download the ratings-only dataset and preprocess it like [40]. In the filtered dataset, each user has reviewed at least 20 books and each book has been reviewed by at least 20 users.
- **Amazon-CDs [18]:** This is a CD review dataset also from the website above, including 3,749,003 ratings covering the same time span. A similar preprocessing operation is executed and the filtering threshold is set as 10.

5.1.2 Tasks. We adopt the top-k recommendation and rating score prediction tasks to evaluate the effectiveness of our method.

- **Top-k Recommendation:** This is one of the most common recommendation tasks to evaluate the model's ability on inferring

Figure 2: The illustration of *embedding size adaptive neural network* structure. Only the user *embedding input* is shown in detail above. The item *embedding input* structure is similar. Different colors indicate the model inference process at different timesteps. The dashed rectangles represent the tensors that do not participate in the current forward propagation. The hollow rectangles represent intermediate tensors that will not be saved after the neural network forward propagation.

users' intentions. In detail, the model needs to recommend a list of items with the length k to each user according to their historical interaction records. The accuracy is measured with metrics Recall@k and NDCG@k. Recall@k indicates what percentage of a user's rated items can emerge in the list. NDCG@k is the normalized discounted cumulative gain at k, which takes the position of correctly recommended items into consideration. Here, we take k to be 10.

- **Rating Score Prediction:** Similar to previous works [28, 29, 51], we also utilize the following two rating score prediction subtasks as the benchmark: **binary classification** and **multiclass classification**. The former can be understood as predicting if a user likes an item. And the latter can be used to estimate the discretized interest degree of users. For binary classification task, when pre-processing the raw data, we set the rating scores greater than the threshold 3.5 to 1.0 and others to 0.0. The model performance is measured with classification accuracy and mean-squared-error loss [29, 51]. For multiclass classification task, we regard the 5-star rating scores as 5 classes. The model performance is measured with classification accuracy and cross-entropy loss [29, 51].

5.1.3 ***Baselines****.* Following methods serve as the baselines:

- **Fixed**: The base Neural Collaborative Filtering [19] model, where the embedding sizes for users and items are both identical and fixed. To fairly compare experimental results, , we set the embedding sizes to 128 and 222, regarding the two versions **Fixed-128** and **Fixed-222** of the model, respectively.
- **DARTS** [28]: A type of *soft-selection* algorithm developed from the neural architecture search. The weight vectors regarding embedding sizes are trained directly with gradient backpropagation.

- **AutoEmb** [51]: Another type of *soft-selection* algorithm similar to DARTS. However, the weight vectors are the outputs of controller neural networks independent of the recommendation model.
- **ESAPN** [29]: A type of *hard-selection* algorithm using the REIN-FORCE algorithm [46] as the controller to dynamically choose the suitable embedding size for corresponding users and items.

Due to the fact that Amazon datasets cannot provide the raw item feature vectors, we run both **DESS-CV** (with (FRE_l^u, IND_l^u) and (FRE_l^i, POD_l^i) as embedding size search policies input) and **DESS-FRE** (with only historical frequency as search policies input) on ml-20m and ml-latest datasets while only **DESS-FRE** on Amazon-Books and Amazon-CDs datasets. The comparison between DESS-CV and DESS-FRE can be regarded as the ablation study to the embedding size indicators proposed in Section 4.2. All the results below are the average of five trials with different random seeds.

5.2 Results and Analysis

5.2.1 ***Recommendation Accuracy (RQ1)****.* In Tables 1, 2 and 3, we report the average performance of recommendation models on test data sequence $\{D_1^{te}, ..., D_T^{te}\}$ of each task. First, we observe that DESS performs significantly better than the *fixed* methods and the *soft-selection* methods. This proves that *hard-selection*, generally speaking, is a more effective way to search embedding sizes. Second, we notice that our DESS-FRE and DESS-CV achieve better results compared with the state-of-the-art *hard-selection* algorithm ESAPN on all tasks and datasets. This demonstrates the effectiveness of DESS algorithm in improving the streaming recommendation. Third, by comparing the results of DESS-FRE with ESAPN and

Table 1: The top-k recommendation performance on all four datasets. Our results are statistically significant (t-test, $p <= 0.01$).

Methods	ml-20m		ml-latest		Amazon-Books		Amazon-CDs	
	Recall@10	NDCG@10	Recall@10	NDCG@10	Recall@10	NDCG@10	Recall@10	NDCG@10
Fixed-128	0.0774	0.0785	0.0779	0.0783	0.0568	0.0404	0.0759	0.0402
Fixed-222	0.0769	0.0779	0.0785	0.0791	0.0573	0.0416	0.0742	0.0397
DARTS	0.0786	0.0791	0.0784	0.0795	0.0597	0.0435	0.0778	0.0432
AutoEmb	0.0771	0.0782	0.0783	0.0792	0.0571	0.0412	0.0780	0.0429
ESAPN	0.0831	0.0842	0.0825	0.0837	0.0632	0.0475	0.0816	0.0468
DESS-FRE (w/o EWI)	**0.0858**	**0.0867**	**0.0866**	**0.0869**	**0.0658**	**0.0494**	**0.0843**	**0.0487**
DESS-FRE	**0.0866**	**0.0876**	**0.0874**	**0.0879**	**0.0672**	**0.0507**	**0.0852**	**0.0493**
DESS-CV (w/o EWI)	**0.0875**	**0.0888**	**0.0879**	**0.0887**	N/A			
DESS-CV	**0.0898**	**0.0891**	**0.0884**	**0.0895**				

Table 2: The performance of both rating score binary classification and rating score multiclass classification tasks on ml-20m dataset and ml-latest dataset. Our results are statistically significant (t-test, $p <= 0.01$).

Methods	ml-20m				ml-latest			
	Binary Classification Task		Multi Classification Task		Binary Classification Task		Multiclass Classification Task	
	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss
Fixed-128	70.93%	0.1904	47.88%	1.1870	70.98%	0.1900	48.28%	1.1804
Fixed-222	71.09%	0.1896	48.19%	1.1799	71.13%	0.1892	48.62%	1.1727
DARTS	71.07%	0.1897	48.24%	1.1781	71.12%	0.1892	48.73%	1.1702
AutoEmb	70.54%	0.1917	47.99%	1.1820	71.00%	0.1892	48.61%	1.1718
ESAPN	71.62%	0.1861	49.24%	1.1539	71.40%	0.1870	49.52%	1.1510
DESS-FRE (w/o EWI)	**71.89%**	**0.1843**	**49.55%**	**1.1463**	**71.77%**	**0.1849**	**49.89%**	**1.1435**
DESS-FRE	**71.93%**	**0.1837**	**49.67%**	**1.1438**	**71.98%**	**0.1838**	**50.05%**	**1.1383**
DESS-CV (w/o EWI)	**72.29%**	**0.1823**	**49.98%**	**1.1358**	**72.35%**	**0.1819**	**50.24%**	**1.1327**
DESS-CV	**73.28%**	**0.1768**	**51.19%**	**1.1103**	**73.07%**	**0.1779**	**50.73%**	**1.1216**

Table 3: The performance of both rating score binary classification and rating score multiclass classification tasks on Amazon-Books dataset and Amazon-CDs dataset. Our results are statistically significant (t-test, $p <= 0.01$).

Methods	Amazon-Books				Amazon-CDs			
	Binary Classification Task		Multi Classification Task		Binary Classification Task		Multiclass Classification Task	
	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss
Fixed-128	80.75%	0.1459	54.31%	1.1225	80.28%	0.1545	57.17%	1.1638
Fixed-222	80.87%	0.1433	54.85%	1.1026	80.09%	0.1540	57.24%	1.1536
DARTS	81.02%	0.1415	55.22%	1.0895	80.50%	0.1498	57.74%	1.1305
AutoEmb	80.60%	0.1456	54.77%	1.0910	80.48%	0.1504	57.89%	1.1232
ESAPN	81.51%	0.1357	56.84%	1.0437	81.44%	0.1399	59.50%	1.0616
DESS-FRE (w/o EWI)	**81.62%**	**0.1342**	**57.14%**	**1.0372**	**81.82%**	**0.1367**	**60.02%**	**1.0453**
DESS-FRE	**81.76%**	**0.1336**	**57.35%**	**1.0329**	**81.93%**	**0.1361**	**60.15%**	**1.0491**

DESS-CV with DESS-FRE, respectively, we can find that the non-stationary LinUCB-based search policy π_{SE} and the two indicators (*IND* and *POD*) both contribute to the performance gains.

*5.2.2 **Embedding Size Selection Regret (RQ2)**.* We report the regret in terms of users due to the space limitation. The item side has a similar trend. Fixed embedding size methods and *soft-selection* methods have no regret because they actually use the embeddings of all sizes at the same time. We illustrate the regret curves of two rating score prediction tasks on each dataset in Figure 3. First, the observed decline in regret connects the aforementioned improvement in accuracy, justifying the advantage of modeling the

dynamic embedding size search as bandits. Second, the regret can be reduced several times to dozens of times compared with ESAPN. The regret of DESS-CV is a bit lower than that of DESS-FRE. These demonstrate the effectiveness of π_{SE} and also the two indicators. Third, from the regret curves, we observe the sublinear increase phenomena of DESS-FRE and DESS-CV, which is also in line with the regret upper bound guarantee given in Section 4.4.

*5.2.3 **Model Memory Cost (RQ3)**.* In Figure 4, we calculate the average number of embedding parameters for each algorithm on each task. From this figure, we observe that the embedding parameter quantity of DESS-FRE is much less than that of other

Figure 3: The regret curves of three *hard-selection* methods: ESAPN, DESS-FRE, DESS-CV across binary classification and multiclass classification tasks. DESS-CV cannot be evaluated on the Amazon datasets due to the lack of raw item features.

Figure 4: The parameter comparison among different methods. In the table, "BC" refers to the binary classification task and "MC' denotes the multiclass classification task.

Figure 5: The average training time and inference time of different embedding size search methods in two rating score prediction tasks on ml-latest dataset and ml-20m dataset.

four methods. As for ml-20m, the memory consumption of Fixed-128 and two *soft selection* approaches are 1.93 times, 3.35 times, and 3.45 times of DESS-FRE. In the recommendation tasks on ml-latest dataset, our method only consumes 50.2%, 28.9%, 28.9% and 28.2% memory compared with the Fixed-128, Fixed-222, DARTS, and AutoEmb, respectively. Moreover, the embedding memory cost brought by DESS-CV is even less than that of DESS-FRE. Similar memory overhead savings can also be observed on Amazon datasets. These all certify that our proposed methods can reduce the memory cost effectively and contribute to the more efficient algorithm DESS.

5.2.4 ***Time Efficiency Analysis (RQ4)***. Time efficiency is also of great significance when deploying streaming recommendation models in real world. In the experiments, we count the average training time and average inference time of our method and embedding size-changeable baselines for the above two classification tasks on ml-latest and ml-20m datasets. The results are shown in Figure 5. From the figure, we can observe that the average training time of our **DESS** algorithms is much less that of other *soft-selection* and *hard-selection* methods. More precisely, the training time of DESS-FRE is no more than 20% ∼ 30% of Darts and is even no more than 10% of AutoEmb. We can also observe a similar trend in average inference time histogram. Thus, we can speculate that our

DESS algorithm holds obvious time efficiency advantage compared with previous methods. This is actually because that the bandit inherently has more lighter model and faster decision-making process compared with deep neural work-based and reinforcement learning-based embedding size selection policies. Besides, we can find that both the average training time and average inference time of DESS-FRE are less than DESS-CV to some extent. This is due to the lower input dimension and the smaller linear matrix in the bandit's reward model, which finally lead to much faster computation.

5.2.5 ***Method Applicability Analysis (RQ5)***. We also explore the effects of our method when choosing the matrix factorization-based model and distance-based model as the base streaming recommendation models, respectively. Different dynamic embedding size search methods are evaluated with the rating score binary classification task on ml-20m and ml-latest datasets. From the accuracy reported in Table 4, it can be first noticed that the recommendation models with fixed embedding sizes (Fixed-128 and Fixed-222) are much worse than that with dynamic embedding sizes. This also confirms the necessity of the dynamic embedding size search in the streaming recommendation. Second, we observe that DESS-CV

Table 4: The binary classification accuracy when adopting matrix factorization-based model and distance-based model. Our results are statistically significant (t-test, $p <= 0.01$).

Methods	Matrix Factorization-based		Distance-based	
	ml-20m	ml-latest	ml-20m	ml-latest
Fixed-128	50.03%	50.08%	54.87%	55.36%
Fixed-222	50.07%	50.04%	52.02%	52.59%
DARTS	70.74%	70.92%	70.17%	70.17%
AutoEmb	69.77%	70.32%	70.30%	70.44%
ESAPN	70.21%	71.86%	**71.55%**	71.03%
DESS-FRE	**72.75%**	**72.61%**	71.46%	**71.84%**
DESS-CV	**73.84%**	**73.25%**	**73.02%**	**72.49%**

outperforms all the baselines regardless of datasets and recommendation models. Even the DESS-FRE without the support from *IND* and *POD* is still superior to all the previous methods in most cases. Such experimental results demonstrate that our algorithm can be a general approach towards more effective dynamic embedding size search in streaming recommendation.

5.2.6 **Ablation Study (RQ6)**. To validate the effectiveness of *Embedding Warm Initialization* technique proposed in Section 4.5, we also conduct the experiments of DESS-FRE (w/o EWI) and DESS-CV (w/o EWI) on four datasets, whose results are shown in Tables 1, 2, and 3. We can observe that, DESS-FRE and DESS-CV both achieve stable performance gain over DESS-FRE (w/o EWI) and DESS-CV (w/o EWI), respectively, especially on ml-20m and ml-latest datasets. The limited improvement on Amazon datasets may be due to the bottleneck of the base recommender model itself. These results demonstrate that initializing the embeddings with previous information via a simple linear transformation can effectively benefit the recommendation performance along the timeline.

6 CONCLUSION

In this work, we first rethink the streaming model update process and then model the dynamic embedding size as a bandit problem. Based on the embedding size indicator analysis, we provide the DESS algorithm and obtain a sublinear dynamic regret upper bound as the theoretical guarantee. The results of recommendation accuracy, memory cost, and time consumption across various recommendation tasks on four open datasets demonstrate the effectiveness of our method. In the future, we plan to explore the automated tuning methods of other hyperparameters like learning rate in the streaming machine learning.

REFERENCES

[1] Bilge Acun, Matthew Murphy, Xiaodong Wang, Jade Nie, Carole-Jean Wu, and Kim Hazelwood. 2021. Understanding training efficiency of deep learning recommendation models at scale. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 802–814.

[2] Samuele Battaglino and Erdem Koyuncu. 2020. A generalization of principal component analysis. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 3607–3611.

[3] Omar Besbes, Yonatan Gur, and Assaf Zeevi. 2014. Stochastic multi-armed-bandit problem with non-stationary rewards. *Advances in neural information processing systems* 27 (2014).

[4] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. 2017. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344* (2017).

[5] Badrish Chandramouli, Justin J Levandoski, Ahmed Eldawy, and Mohamed F Mokbel. 2011. Streamrec: a real-time recommender system. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 1243–1246.

[6] Shiyu Chang, Yang Zhang, Jiliang Tang, Dawei Yin, Yi Chang, Mark A Hasegawa-Johnson, and Thomas S Huang. 2017. Streaming recommender systems. In *Proceedings of the 26th international conference on world wide web*. 381–389.

[7] Chen Chen, Hongzhi Yin, Junjie Yao, and Bin Cui. 2013. Terec: A temporal recommender system over tweet stream. *Proceedings of the VLDB Endowment* 6, 12 (2013), 1254–1257.

[8] Xiong-Hui Chen, Bowei He, Yang Yu, Qingyang Li, Zhiwei Qin, Wenjie Shang, Jieping Ye, and Chen Ma. 2023. Sim2Rec: A Simulator-based Decision-making Approach to Optimize Real-World Long-term User Engagement in Sequential Recommender Systems. *arXiv preprint arXiv:2305.04832* (2023).

[9] Weiyu Cheng, Yanyan Shen, and Linpeng Huang. 2020. Differentiable neural input search for recommender systems. *arXiv preprint arXiv:2006.04466* (2020).

[10] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.

[11] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*. 271–280.

[12] Robin Devooght, Nicolas Kourtellis, and Amin Mantrach. 2015. Dynamic matrix factorization with priors on unknown values. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 189–198.

[13] Ernesto Diaz-Aviles, Lucas Drumond, Lars Schmidt-Thieme, and Wolfgang Nejdl. 2012. Real-time top-n recommendation in social streams. In *Proceedings of the sixth ACM conference on Recommender systems*. 59–66.

[14] George H Dunteman. 1989. *Principal components analysis*. Number 69. Sage.

[15] AA Ginart, Maxim Naumov, Dheevatsa Mudigere, Jiyan Yang, and James Zou. 2021. Mixed dimension embeddings with application to memory-efficient recommendation systems. In *2021 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2786–2791.

[16] F Maxwell Harper and Joseph A Konstan. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2015), 1–19.

[17] Bowei He, Xu He, Yingxue Zhang, Ruiming Tang, and Chen Ma. 2023. Dynamically Expandable Graph Convolution for Streaming Recommendation. In *Proceedings of the ACM Web Conference 2023*. 1457–1467.

[18] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*. 507–517.

[19] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[20] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. 2017. Collaborative metric learning. In *Proceedings of the 26th international conference on world wide web*. 193–201.

[21] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*. Springer, 507–523.

[22] Arun Jambulapati, Jerry Li, and Kevin Tian. 2020. Robust sub-gaussian principal component analysis and width-independent schatten packing. *Advances in Neural Information Processing Systems* 33 (2020), 15689–15701.

[23] Manas R Joglekar, Cong Li, Mei Chen, Taibai Xu, Xiaoming Wang, Jay K Adams, Pranav Khaitan, Jiahui Liu, and Quoc V Le. 2020. Neural input search for large scale recommendation models. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2387–2397.

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

[25] Baekjin Kim and Ambuj Tewari. 2020. Randomized exploration for non-stationary stochastic linear bandits. In *Conference on Uncertainty in Artificial Intelligence*. PMLR, 71–80.

[26] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[27] Tor Lattimore and Csaba Szepesvári. 2020. *Bandit algorithms*. Cambridge University Press.

[28] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

[29] Haochen Liu, Xiangyu Zhao, Chong Wang, Xiaobing Liu, and Jiliang Tang. 2020. Automated embedding size search in deep recommender systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2307–2316.

[30] Siyi Liu, Chen Gao, Yihong Chen, Depeng Jin, and Yong Li. 2021. Learnable embedding sizes for recommender systems. *arXiv preprint arXiv:2101.07577* (2021).

[31] Siyi Liu, Chen Gao, Yihong Chen, Depeng Jin, and Yong Li. 2021. Learnable Embedding sizes for Recommender Systems. In *ICLR*. OpenReview.net.

[32] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. 2018. Neural architecture optimization. *Advances in neural information processing systems* 31 (2018).

[33] Fuyuan Lyu, Xing Tang, Hong Zhu, Huifeng Guo, Yingxue Zhang, Ruiming Tang, and Xue Liu. 2022. OptEmbed: Learning Optimal Embedding Table for Click-through Rate Prediction. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 1399–1409.

[34] Chen Ma, Liheng Ma, Yingxue Zhang, Ruiming Tang, Xue Liu, and Mark Coates. 2020. Probabilistic metric learning with adaptive margin for top-k recommendation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1036–1044.

[35] Donald W Marquardt and Ronald D Snee. 1975. Ridge regression in practice. *The American Statistician* 29, 1 (1975), 3–20.

[36] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*. PMLR, 4095–4104.

[37] Yoan Russac, Claire Vernade, and Olivier Cappé. 2019. Weighted linear bandits for non-stationary environments. *Advances in Neural Information Processing Systems* 32 (2019).

[38] Yang Song, Ziming Zhuang, Huajing Li, Qiankun Zhao, Jia Li, Wang-Chien Lee, and C Lee Giles. 2008. Real-time automatic tag recommendation. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. 515–522.

[39] Karthik Subbian, Charu Aggarwal, and Kshiteesh Hegde. 2016. Recommendations for streaming data. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 2185–2190.

[40] Jianing Sun, Zhaoyue Cheng, Saba Zuberi, Felipe Pérez, and Maksims Volkovs. 2021. Hgcf: Hyperbolic graph convolution networks for collaborative filtering. In *Proceedings of the Web Conference 2021*. 593–601.

[41] Bruno Veloso, Luciano Caroprese, Matthias König, Sónia Teixeira, Giuseppe Manco, Holger H Hoos, and João Gama. 2021. Hyper-parameter Optimization for Latent Spaces. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 249–264.

[42] Junshan Wang, Guojie Song, Yi Wu, and Liang Wang. 2020. Streaming graph neural networks via continual learning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1515–1524.

[43] Junshan Wang, Wenhao Zhu, Guojie Song, and Liang Wang. 2022. Streaming Graph Neural Networks with Generative Replay. In *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*. 1878–1888.

[44] Weiqing Wang, Hongzhi Yin, Zi Huang, Qinyong Wang, Xingzhong Du, and Quoc Viet Hung Nguyen. 2018. Streaming ranking based recommender systems. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 525–534.

[45] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*. 165–174.

[46] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning* (1992), 5–32.

[47] Qingyun Wu, Naveen Iyer, and Hongning Wang. 2018. Learning contextual bandits in a non-stationary environment. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 495–504.

[48] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. 2018. SNAS: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926* (2018).

[49] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 353–362.

[50] Peng Zhao, Lijun Zhang, Yuan Jiang, and Zhi-Hua Zhou. 2020. A simple approach for non-stationary linear bandits. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 746–755.

[51] Xiangyu Zhao, Haochen Liu, Wenqi Fan, Hui Liu, Jiliang Tang, Chong Wang, Ming Chen, Xudong Zheng, Xiaobing Liu, and Xiwang Yang. 2021. Autoemb: Automated embedding dimensionality search in streaming recommendations. In *2021 IEEE International Conference on Data Mining (ICDM)*. IEEE, 896–905.

[52] Xiangyu Zhao, Haochen Liu, Hui Liu, Jiliang Tang, Weiwei Guo, Jun Shi, Sida Wang, Huiji Gao, and Bo Long. 2021. Autodim: Field-aware embedding dimension

searchin recommender systems. In *Proceedings of the Web Conference 2021*. 3015–3022.

A NOTATIONS

We summarize the main notations used in this paper in Table 5.

Table 5: Major notations.

K	The number of arms (embedding size candidates)
T	The total number of time steps
L	The length of the whole data stream
C	The set of contexts for non-stationary LinUCB
U	The upper bound of contexts for non-stationary LinUCB
S	The upper bound of parameters in reward models of bandit
γ	The discount factor in non-stationary LinUCB bandit
d	The dimension of context vectors for bandit
π^u_{SE}	The dynamic embedding size search policy for users
π^l_{SE}	The dynamic embedding size search policy for items
r_l	The reward received at l-th user-item interaction ($l \le L$)
$\theta_{l,a}$	The updated parameter of the reward model for arm a at l-th user-item interaction
$\theta^*_{l,a}$	The parameter of the oracle reward model for arm a at l-th user-item interaction
M_t	The updated recommendation model at time step t ($t \le T$)
$\mathbf{F}^i$	The raw feature vector for item i

B THEOREM PROOF

B.1 Proof of Theorem 1

PROOF. First recall that $a^*_l = \arg max_{a \in \mathcal{A}} \langle \mathbf{x}_l, \theta^*_{l,a} \rangle$ and $\mathbf{x}_{l,1} = \mathbf{x}_{l,2} = ... = \mathbf{x}_{l,K} = \mathbf{x}_{l,a^*_l} = \mathbf{x}_l = \mathbf{x}_{(t,j)} (l = t \times |D_t| + j)$,

$$
\begin{aligned}
R_L &= \sum_{t=1}^{T} \sum_{j=1}^{|D_t^{val}|} \langle \mathbf{x}_{(t,j)}, \theta^*_{(t,j),a^*_{(t,j)}} \rangle - \langle \mathbf{x}_{(t,j)}, \theta^*_{(t,j),a_{(t,j)}} \rangle \\
&= \sum_{l=1}^{L} \langle \mathbf{x}_l, \theta^*_{l,a^*_l} \rangle - \langle \mathbf{x}_l, \theta^*_{l,a_l} \rangle \\
&= \sum_{l=1}^{L} \langle \mathbf{x}_l, \theta^*_{l,a^*_l} \rangle - \langle \mathbf{x}_l, \theta_{l,a^*_l} \rangle + \sum_{l=1}^{L} \langle \mathbf{x}_l, \theta_{l,a^*_l} \rangle - \langle \mathbf{x}_l, \theta_{l,a_l} \rangle \\
&\quad + \sum_{l=1}^{L} \langle \mathbf{x}_l, \theta_{l,a_l} \rangle - \langle \mathbf{x}_l, \theta^*_{l,a_l} \rangle \\
&= \sum_{l=1}^{L} \langle \mathbf{x}_l, \theta_{l,a^*_l} \rangle - \langle \mathbf{x}_l, \theta_{l,a_l} \rangle + \sum_{l=1}^{L} \langle \mathbf{x}_l, \theta^*_{l,a^*_l} \rangle - \langle \mathbf{x}_l, \theta_{l,a^*_l} \rangle \\
&\quad + \sum_{l=1}^{L} \langle \mathbf{x}_l, \theta_{l,a_l} \rangle - \langle \mathbf{x}_l, \theta^*_{l,a_l} \rangle \\
&\le \frac{2k}{c} \sum_{l=1}^{L} \beta_t [\|\mathbf{x}_l\|_{V^{-1}_{l,a^*_l}} - \|\mathbf{x}_l\|_{V^{-1}_{l,a_l}}] + \sum_{l=1}^{L} Er(\mathbf{x}_l, \theta_{l,a^*_l}) \\
&\quad + \sum_{l=1}^{L} Er(\mathbf{x}_l, \theta_{l,a_l}).
\end{aligned}
\tag{13}
$$

Thanks to Lemma 1 and Lemma 2:

$$R_L \leq \underbrace{\sum_{l=1}^{L} \frac{4k}{c} \beta_l \|\mathbf{x}_l\|_{V_{l,a_l^*}^{-1}}}_{R_L^1} + \underbrace{\sum_{l=1}^{T} \frac{4kU}{c} \sqrt{1 + \frac{U^2}{\lambda(1-\gamma)}} \frac{2kSU^2}{\lambda} \frac{\gamma^D}{1-\gamma}}_{R_L^2}$$

$$+ \underbrace{\sum_{l=1}^{L} \frac{4kU}{c} \sqrt{1 + \frac{U^2}{\lambda(1-\gamma)}} \sum_{s=l-D}^{l-1} k \sqrt{\frac{d}{\lambda(1-\gamma)}} max_a \|\theta_{s,a}^* - \theta_{s+1,a}^*\|}_{R_L^3}.$$

(14)

$$R_L^1 \leq \sum_{l=1}^{L} \frac{4k}{c} \beta_L \|\mathbf{x}_l\|_{V_{l,a_l^*}^{-1}}$$

$$\leq \frac{4k}{c} \beta_L \sqrt{L} \sqrt{\sum_{l=1}^{L} \|\mathbf{x}_l\|_{V_{l,a_l^*}^{-1}}} \quad (Cauthy - Schwarz)$$

$$\leq \frac{4k}{c} \beta_L \sqrt{2dL \max(1, \frac{U^2}{\lambda})} \sqrt{L \log(\frac{1}{\gamma}) + \log(1 + \frac{U^2(1-\gamma^L)}{\lambda d(1-\gamma)})} (Lemma\ 2).$$

(15)

Because $\sqrt{1 + \frac{U^2}{\lambda(1-\lambda)}}$ can be upper bounded by $1 + \frac{U}{\sqrt{\lambda(1-\lambda)}}$:

$$R_L^2 \leq \frac{8k^2 SU^3 \gamma^D}{c\lambda(1-\gamma)} L + \frac{8k^2 SU^4 \gamma^D}{c\lambda^{\frac{3}{2}}(1-\gamma)^{\frac{3}{2}}} L$$

$$R_L^3 \leq \frac{4k^2 UD}{c\sqrt{\lambda}} \sqrt{\frac{d}{1-\gamma}} B_L + \frac{4k^2 U^2 D}{c\lambda} \frac{\sqrt{d}}{1-\gamma} B_L.$$

(16)

Add such three components together, we have:

$$R_L \leq R_L^1 + R_L^2 + R_L^3$$

$$\leq \sqrt{32 \max(1, U^2/\lambda)} \frac{k}{c} \beta_L \sqrt{dL} \sqrt{L \log(\frac{1}{\gamma}) + \log(1 + \frac{U^2(1-\gamma^L)}{\lambda d(1-\gamma)})} +$$

$$\frac{8k^2 SU^3 \gamma^D}{c\lambda(1-\gamma)} L + \frac{8k^2 SU^4 \gamma^D}{c\lambda^{\frac{3}{2}}(1-\gamma)^{\frac{3}{2}}} L + \frac{4k^2 UD}{c\sqrt{\lambda}} \sqrt{\frac{d}{1-\gamma}} B_L + \frac{4k^2 U^2 D}{c\lambda} \frac{\sqrt{d}}{1-\gamma} B_L.$$

(17)

Thus, the theorem 1 is proved. □

B.2 Proof of Corollary 1

Proof. By neglecting the logarithmic term,

$$\beta_L \sqrt{dL} \sqrt{L \log(1/\gamma)} \sim dL \frac{B_L^{1/5} d^{-1/10}}{L^{1/5}} = d^{9/10} B_L^{1/5} L^{4/5}$$

$$\gamma^D L/(1-\gamma)^{3/2} \sim e^{-\log L} L(\frac{d^{1/5} L^{2/5}}{B_L^{2/5}})^{3/2} = d^{3/10} B_L^{-3/5} L^{3/5}$$

$$\frac{\sqrt{d}}{1-\gamma} DB_L \sim d^{1/2} B_L (\frac{d^{1/5} L^{2/5}}{B_L^{2/5}})^2 = d^{9/10} B_L^{1/5} L^{4/5}.$$

Thus, with high probability, we have:

$$R_L = O_{L \to \infty}(d^{9/10} B_L^{1/5} L^{4/5}).$$

□

C IMPLEMENTATION DETAILS

For the fair comparison, we set the embedding size candidates of the last three methods and our DESS as $\{2, 4, 8, 16, 64, 128\}$, which is also consistent with previous related researches [28, 29, 51]. All the methods share the embedding size adaptive neural network designed in 4.5 as the streaming recommendation model. We use the Adam optimizer with an initial learning rate as 0.001 and a regularization parameter as 0.001. The hidden layer size of the recommendation model is set to 512. The min-batch size is set as 500. The discount factor γ is set as 0.99. The subguassian constant σ is set as 3.0. Without specifications, the hyper-parameters are set same as the original paper. We implement our algorithm with PyTorch and test it on the NVIDIA Titan-RTX GPU with 24 GB memory.