Improving Out-of-Vocabulary Hashing in Recommendation Systems

William Shiao*
wshia002@ucr.edu
University of California, Riverside
Riverside, CA, USA

Mingxuan Ju
mju@snap.com
Snap Inc.
Bellevue, WA, USA

Zhichun Guo
zguo5@nd.edu
University of Notre Dame
Notre Dame, IN, USA

Xin Chen
xin.chen@snap.com
Snap Inc.
Palo Alto, CA, USA

Evangelos E. Papalexakis
epapalex@cs.ucr.edu
University of California, Riverside
Riverside, CA, USA

Tong Zhao
tong@snap.com
Snap Inc.
Bellevue, WA, USA

Neil Shah
nshah@snap.com
Snap Inc.
Bellevue, WA, USA

Yozen Liu
yliu2@snap.com
Snap Inc.
Santa Monica, CA, USA

Abstract

Recommendation systems (RS) are an increasingly relevant area for both academic and industry researchers, given their widespread impact on the daily online experiences of billions of users. In real applications, one common challenge is recommending new users and items unseen (out-of-vocabulary, or OOV) at training time, i.e. the *inductive* setting. Additionally, modern RS also faces challenges in ID embedding table size. To handle large cardinality user/item embeddings, memory intensive embedding tables are required. The size of OOV user/item IDs are often large and varies. As a result of both issues, existing solutions applied in practice are often naïve, such as assigning OOV or hashed users/items to a fix set of random buckets. In this work, we tackle the *cold-start* OOV and ID embedding memory problem and propose approaches that better leverage available user/item features and memory-efficient hashing at the embedding table level. We discuss plug-and-play approaches that are easily applicable to RS models and improve inductive performance without negatively impacting transductive performance. Through our extensive evaluation, we find that proposed methods that exploit feature similarity using LSH consistently outperform alternatives on a majority of model-dataset combinations, with the best one showing a mean improvement of 3.74% over the industry standard baseline in recommendation performance. We release our code and hope our work helps practitioners make more informed decisions for efficiently hashing OOV in their RS and further inspires academic research into improving OOV support in RS.

CCS Concepts

• **Information systems** → **Recommender systems**.

Keywords

recommendation systems, cold-start, out-of-vocabulary, hashing

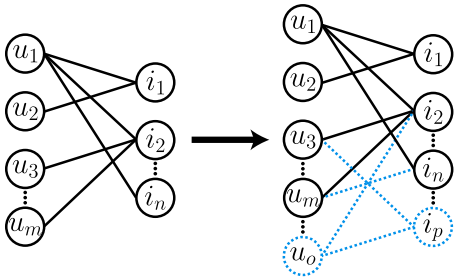


Figure 1: Comparison between transductive (left) and inductive (right) settings. In the transductive setting, RS are evaluated on interactions between users and items observed during training time (i.e., bold links). Whereas in the inductive setting, besides transductive interactions, RS are also evaluated on interactions related to users and items unseen during the training (i.e., both bold and dash links).

1 Introduction

Recommendation systems (RS) suggest items to users and have found wide adoption across a variety of domains. For example, they have been used to recommend advertisements [52, 53], movies [14], friends [40, 44], and products [7, 28] to users. These methods are studied in both academia and industry, but many aspects often differ between academic and industrial recommendation systems [42]. One such difference is their evaluation methodology.

RS research in academia primarily focuses on the *transductive* setting [45, 50], where a portion of interactions are masked out for validation and testing. Such a setting assumes that all users and items in the dataset are seen during training. However, in industrial RS environments, there is often a constant influx of new, or out-of-vocabulary (OOV), users and items that were not seen at training time, i.e., the *inductive setting* which correspond to new users and/or items showing up at validation and testing. Almost all production models are deployed to be utilized in an (at least partially) inductive setting, but a recent survey [42] found that

*Work done during first author's internship at Snap Inc.

only about 10%[1] of 88 recent RS papers evaluated their models in the fully inductive setting, in which OOV users and items are considered. Similarly, existing state-of-the-art models [3, 53, 59] also use embedding tables for sparse ID features, which face similar issues when encountering values unseen at training time since a model would not have an existing row in the embedding table for unseen values.

Modern RS also face challenges with their large memory footprint in user/item ID embedding tables. Large cardinality in practice of the user/item IDs leads to slower training time, increase in cost and hardware requirement or infeasibility during serving. To tackle this problem, hashing is a common way to improve memory efficiency of embedding tables by reducing the large ID space to a fix number of buckets [5, 10, 30, 57, 61].

As a result, industrial practitioners often use primitive methods such as random hashing to a fixed number of OOV buckets whose values are updated during training [1] [2]. While simple to implement, these primitive methods can easily map two very different OOV users/items to the same embedding bucket, i.e. embedding collision, and can greatly affect the recommendation performance [29, 60]. In Figure 2, we show that there exists a clear gap between inductive and transductive performance for all datasets with random OOV bucket assignment. This demonstrates the importance of properly hashing OOV values and leads us to the following question: **Can we re-imagine how we hash OOV users and items to improve the inductive capability of RS?**

While many methods in literature could potentially be used to solve this problem, we constrain our search to methods that meet criteria important to industry practitioners:

- *Efficient*: the OOV embedding method should run in sub-linear time with respect to the total number of users/items.
- *Maintains Transductive Performance*: active users and popular items are often the platform's main income sources. Hence, the OOV embedding method should not sacrifice the base model's performance on non-OOV items.
- *Model-Agnostic*: the OOV embedding method should be applicable to RS with different model architectures.

Given the above criteria, this work explores existing and proposes new OOV embedding methods. These methods range from simply using a zero vector to feature-similarity-based methods. In our experiments, we thoroughly evaluate nine different OOV embedding methods (detailed in Section 3) to provide a broad empirical understanding of the performance of different OOV strategies. Among the 9 OOV methods, inspired by feature-based cold-start work [24], we propose using several feature-based methods, which utilize feature information to compensate for OOV values. In particular, we propose two locality-sensitive hashing (LSH) [11] based methods that exploit feature-similarity consistently outperform other feature or non-feature-based methods in most models-dataset combinations, with the best method showing a mean improvement of 3.74% over the industry-standard random bucket assignment method.

[1]based on an estimate from Figure 1 of [42].

[2]Some examples of industry usage are https://engineering.linkedin.com/blog/2023/enhancing-homepage-feed-relevance-by-harnessing-the-power-of-lar, https://blog.taboola.com/preparing-for-the-unexpected/ and in the Monolith source code [29].

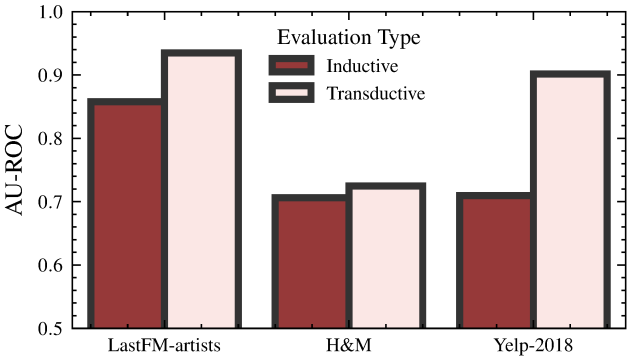


Figure 2: Comparison of inductive vs transductive performance with Wide & Deep models, where OOV (inductive) values are hashed with trained random buckets. We see a clear gap in inductive performance and transductive performance, showing the importance of properly handling OOV values. It is worth noting that even a difference of 0.01 is often significant for large datasets.

To properly evaluate OOV methods under inductive settings, we also create appropriate inductive datasets, as existing public datasets are (1) transductive and (2) lack user/item features. Such limitations directly contradict the setting faced in industrial recommendation systems, where we usually have rich feature information for both users and items and many OOV values. Therefore, we augment three existing open-source datasets and perform a time-based split such that unseen items naturally appear during evaluation. Furthermore, we also created a proprietary industrial dataset from a large social media company containing rich feature information to evaluate OOV methods properly under real applications.

Our contributions can be summarized as the following,

- To the best of our knowledge, our work is the first to provide a comprehensive empirical understanding of the performances of various OOV methods for RS.
- We demonstrate that a class of proposed feature-aware, efficient locality-sensitive hashing-based OOV embedders that exploit feature-similarity consistently outperform existing approaches in inductive performance.
- We provide realistic inductive datasets by augmenting and splitting three open-source datasets, enabling experiments on inductive performance and OOV methods of RS, which will be publicly available upon the release of this manuscript.
- We will open-source our evaluation framework, a major extension of the popular RecBole [64] RS library that adds inductive and OOV support to encourage future research in this area.

2 Preliminaries

In this section, we formally define OOV values and the user/item recommendation system problem. We detail the difference between the two classes of RS model setups that we study, delineated by the use of contextual features: context-free vs. context-aware models since OOV handling behaves differently for each class of models.

Notation. We denote the set of users as $\mathcal{U}$ and the set of items as $\mathcal{I}$. We denote the set of interactions as $\mathcal{R} \subseteq \mathcal{U} \times \mathcal{I}$. For flexibility, let

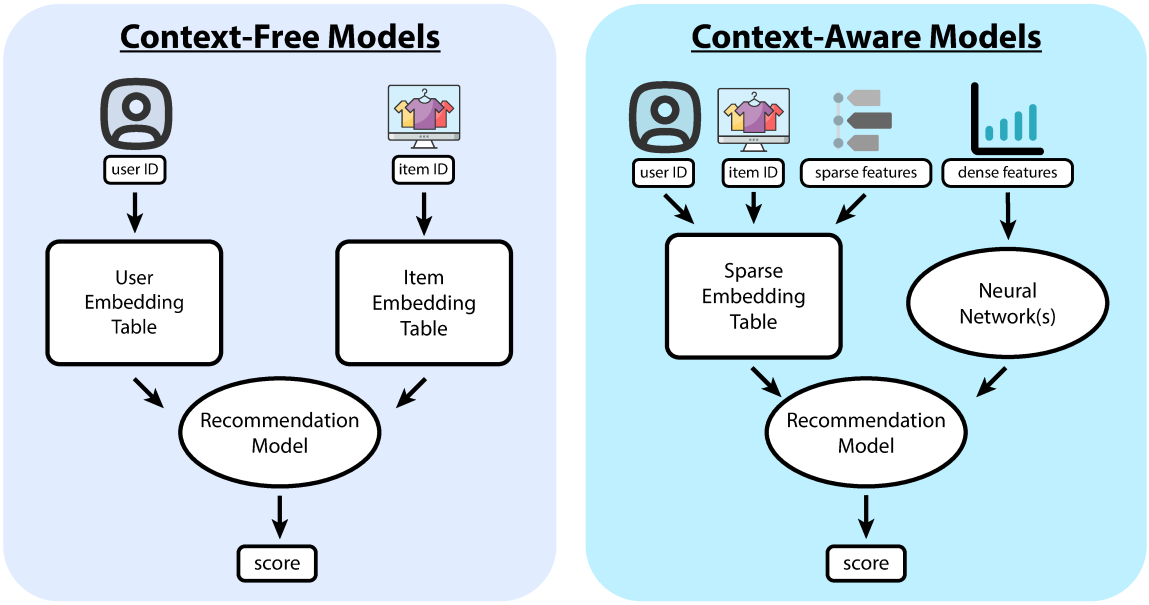


Figure 3: Typical structure of context-aware and context-free recommendation models.

$\boldsymbol{R}$ be the interaction matrix such that $\boldsymbol{R}_{u,i} = 1 \iff (u,i) \in \mathcal{R}$. Let $m = |\mathcal{U}|$ and $n = |\mathcal{I}|$ be the number of users and items, respectively. Let $\boldsymbol{U} \in \mathbb{R}^{m \times d}$ and $\boldsymbol{I} \in \mathbb{R}^{n \times d}$ be the user/item feature matrices. We assume[3] that both feature matrices are of dimension d. For a given user $u \in \mathcal{U}$, we have the associated contextual features $\boldsymbol{U}_u$. Similarly, for a given item $i \in \mathcal{I}$, we have the associated features $\boldsymbol{I}_i$. $\text{cmean}(\cdot) : \mathbb{R}^{n \times d} \to \mathbb{R}^d$ is the column-wise mean of a matrix.

We split the set of users and items based on a time t. All users/items appearing before time t are considered a part of the training set, users $\mathcal{U}_{\text{train}} \subseteq \mathcal{U}$ and items $\mathcal{I}_{\text{train}} \subseteq \mathcal{I}$. The set of training interactions $\mathcal{R}_{\text{train}} \subseteq \mathcal{R}$ is also similarly created.

OOV Values. We consider a value Out-Of-Vocabulary (OOV) if it is a categorical value that does not exist at training time but appears at inference time. Formally, a user u is OOV if and only if $u \notin \mathcal{U}_{\text{train}} \wedge u \in \mathcal{U}$ and an item i is OOV if and only if $i \notin \mathcal{I}_{\text{train}} \wedge i \in \mathcal{I}$. We abbreviate non-OOV values as IV (In-Vocabulary).

Transductive vs. Inductive Settings. In the transductive setting, RS models are evaluated on interactions between users and items that are observed during the model training (i.e., $\mathcal{U}_{\text{eval}} \subseteq \mathcal{U}_{\text{train}}$ and $\mathcal{I}_{\text{eval}} \subseteq \mathcal{I}_{\text{train}}$). Whereas in the inductive setting, besides transductive interactions, RS models are also evaluated on interactions between users and items that do not appear during the model training (i.e., $\mathcal{U}_{\text{eval}} \cup \mathcal{U}_{\text{train}} \neq \mathcal{U}_{\text{train}}$ and $\mathcal{I}_{\text{eval}} \cup \mathcal{I}_{\text{train}} \neq \mathcal{I}_{\text{train}}$).

2.1 Context-Free Models

Context-free models are the ones that do not use any additional feature information other than the IDs of users or items. They are also known as latent factor models [43] and are typically based on matrix factorization (MF) [22, 23], with the goal of approximating the training interaction matrix $\boldsymbol{R}_{\text{train}} \in \mathbb{R}^{m \times n}$. Typically, they factor $\boldsymbol{R}_{\text{train}}$ into two matrices $\boldsymbol{A} \in \mathbb{R}^{m \times d}$ and $\boldsymbol{B} \in \mathbb{R}^{n \times d}$ such that $\boldsymbol{R}_{\text{train}} \approx \boldsymbol{A}\boldsymbol{B}^\top$. The rows of $\boldsymbol{A}$ and $\boldsymbol{B}$ are the user and item embeddings, respectively.

These embeddings can be learned in a variety of ways. For example, the Non-negative Matrix Factorization (NMF) [25] of the interaction matrix can be computed via non-negative least squares or gradient descent. In this work, we focus on two popular context-free models: Bayesian Personalized Ranking (BPR) [38] and DirectAU [50]. BPR is a pairwise ranking model that learns user and item embeddings by maximizing the likelihood of observed interactions. Unlike BPR which utilizes negative sampling for training, DirectAU [50] is a loss function that instead directly optimizes for alignment and uniformity — factors that have been shown to be important for representation quality [54]. It is worth noting that these are often used as retrieval models in production [44], which is why we evaluate them as such in our experiments.

2.2 Context-Aware Models

Context-aware models utilize complimentary contextual features in addition to the user or item IDs. They are often based on the two-tower architecture [17], where each tower is responsible for embedding the user and item features, respectively. The two towers output embeddings of the same dimensionality, allowing them to be directly compared to produce a score for each user-item pair.

However, these models are very dependent on the quality of the input contextual features. In production, practitioners often produce cross-features [4] that capture the interactions between features. As such, we focus on 3 context-aware models that incorporate these cross-features: Wide & Deep [4], eXtreme Deep Factorization

[3]We assume users/item features to have the same dimension d for simplicity, but this can be enforced in practice with a projection layer if user/item features have different dimensions d_u and d_i respectively.

Machine (xDeepFM) [26], and Deep & Cross Networks V2 (DCN-V2) [53]. We focus on these models as they are three of the most popular context-aware models in practice. The models are often used during the ranking or re-ranking stage in production pipelines. Hence, we evaluate them using ranking metrics in our experiments.

The features used in context-aware models are typically categorized into two categories: sparse and dense. Sparse features are categorical features that are typically one-hot or multi-hot encoded. Dense features are continuous features. For example, in the case of social media content recommendation, a user's country could be a sparse feature, and their mean daily app usage could be a dense feature. Sparse features are typically embedded using an embedding table where each row represents the embedding for that feature's ID. These tables are typically randomly initialized and gradually updated during training. Dense features are typically either unmodified or passed through neural network layers. In this work, we focus primarily on handling OOV values in sparse features-specifically, the user/item IDs, which are most likely to be OOV in production settings.

3 Towards a General OOV Embedder

The motivation for this work stems from how OOV users/items are typically handled in real-world production settings. In practice, OOV users/items are often assigned to a random bucket within which all values share the same embedding or are simply assigned completely random embeddings[2]. This clearly results in poor performance for any pure ID-based models (e.g., factorization-based) that rely on stored embeddings for users/items seen at training time. However, even for models that use features, this can still result in poor performance since poorly-assigned embeddings simply add additional noise. For example, random bucket assignment for OOV users means that two OOV users have the same chance to share an embedding, regardless of how similar/different they are.

Since our goal is to improve OOV support for most general recommendation systems, regardless of specific model architecture, we limit the scope of our modifications to a component that is used in almost all production recommendation systems: the embedding table. In this work, we focus primarily on OOV support for unseen user/item IDs, but the same ideas can also be easily extended to improve support for unseen categorical values in other features. This leads to the following formal definition of an OOV embedder:

OOV Embedders. A user OOV embedder $f_{\text{user}} : \mathcal{U} \setminus \mathcal{U}_{\text{train}} \to \mathbb{R}^d$ maps an OOV user to a real-valued embedding. An item OOV embedder does the same: $f_{\text{item}} : \mathcal{I} \setminus \mathcal{I}_{\text{train}} \to \mathbb{R}^d$.

For the sake of simplicity, we describe all the following OOV embedders in terms of OOV users, but we use them for both OOV users and items during evaluation. They can be easily converted to item OOV embedders by substituting the appropriate variables.

3.1 Heuristic-based Embedders

In this work, we first introduce several heuristic-based OOV embedding models that do not require additional trainable parameters. These are straightforward to apply in practice due to their speed and ease of implementation.

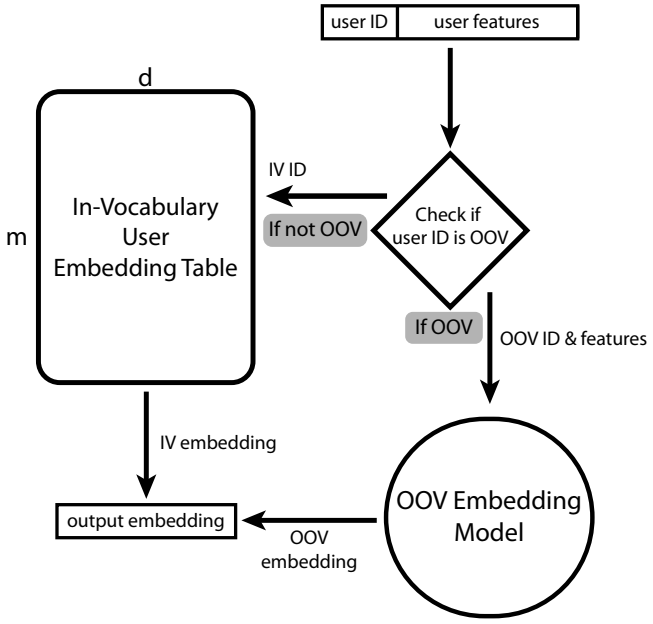


Figure 4: How IV/OOV user IDs are handled under our framework. Item IDs are handled the same way.

Zero Embedder. `zero` simply uses the zero vector for all OOV inputs. This is a simple solution sometimes used in Natural Language Processing (NLP) for OOV words [31, 34]. Formally, $f_{\text{zero}}(\cdot) = \{0\}^d$. For context-free models, all new users will randomly select items (we ensure that items with the same score will be randomly selected without bias towards their ID). For context-aware models, all predictions will depend entirely on a user's/item's contextual information.

Mean Embedder. `mean` uses the column-wise mean of the embedding matrix for all OOV IDs. Note that users and items use their respective means. Formally, for users, $f_{\text{mean}}(\cdot) = \text{cmean}(\boldsymbol{U})$. For context-free models, this means that the RS model will recommend the same popular items to all new users and that all new items will have the same probability of being recommended.

Fixed Random Embedder. `rand` returns a random floating point vector for all OOV IDs. There are b fixed random vectors for each ID type (e.g., user ID, item ID). This ensures that the model's output is deterministic for users/items. Formally, for a set of random vectors $\mathcal{V} = \{\boldsymbol{v} \in \mathbb{R}^d\}$, we have $f_{\text{rand}}(\cdot) = \mathcal{V}_{g(u)}$ where g is a random hash function[5] $\mathbb{Z} \to \{1, 2, \ldots b\}$. This approach is similar to generating a random vector, except that (a) the output for a given ID is deterministic, and (b) the maximum amount of memory used is bounded by b.

KNN Embedder. `knn` returns the mean of the k nearest neighbors of a given point, as measured by the inner product of the features. Formally, for a user u, we have $f_{\text{knn}}(u) = \frac{1}{k} \sum_{a \in \text{K-NEAREST}(u)} \boldsymbol{U}_a$. With $k = 2$, this is similar to the double-hashing performed by Zhang et al. [60], except that we use feature similarity instead of

[5]We use the three-round integer hash function from [56].

Table 1: Comparison of the different OOV embedders evaluated in this work. For applicable methods, θ refers to the number of parameters in the neural network, b refers to the number of buckets, and n is the number of input items. *Features* refer to non-ID features. We assume the embedding dimensionality is constant for the complexity analysis.

Embedder	zero	mean	rand	r-bucket	knn	dhe	fdhe	dnn	m-lsh	s-lsh
Requires training	✗	✗	✗	✓	✗	✓	✓	✓	✓	✓
Uses user/item ID	✗	✗	✓	✓	✗	✓	✓	✗	✗	✗
Uses trainble OOV buckets	✗	✗	✗	✓	✗	✗	✗	✗	✓	✓
Uses features	✗	✗	✗	✗	✓	✗	✓	✓	✓	✓
Same features → same embedding	✗	✓	✗	✗	✓	✗	✗	✓	✓	✓
Requires pre-processing	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗
Complexity	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$<O(n)^4$	$O(\theta)$	$O(\theta)$	$O(\theta)$	$O(b)$	$O(b)$
Potential unique embeddings	1	1	1	b	$> n$	$> n$	$> n$	$> n$	$> n$	b

random hashing to select rows. In order to meet the efficiency criteria mentioned in Section 1, we use approximate nearest neighbor search through libraries like FAISS [19] and SCaNN [12]. Each training ID's k-nearest neighbors can optionally be pre-computed and stored to prevent additional overhead during training.

3.2 Learning-based Embedders

We also consider a set of trained embedders that are optimized during the training of the base model. As mentioned in Section 3.3, we freeze the non-OOV parameters of the base model to avoid affecting its transductive performance. Some of these methods use an embedding table with b rows, where each row corresponds to an OOV *bucket*. This value can be tuned depending on the expected number of OOV values. In the following paragraphs, we introduce several different learning-based OOV embedders. We describe how these embedders are optimized in Section 3.3.

Random Buckets. `r-bucket` randomly assigns an embedding (denoted as a bucket) to a given OOV ID. This mapping is done with a deterministic hash function[5] to ensure that the bucket mappings remain consistent. The chance of any bucket being selected is uniform. Given o OOV IDs and b buckets, each bucket's expected number of OOV IDs is o/b. This is similar to `rand`, except that the values in each bucket are optimized during training. This is TensorFlow's [1] default approach for handling OOV values.

DHE. Deep Hash Embedding (DHE) [21] substitutes a deep neural network for the embedding table. To ensure determinism for a given ID, they first compute many hashes on that ID and use those as inputs to the neural network. We use SipHash [2] with different key values as the hash functions for our implementation. DHE was originally created as a drop-in replacement for the main embedding table in context-free methods, but we use it as an OOV embedded (only on OOV IDs) since it naturally works in this case.

F-DHE. Kang et al. [21] mentions that DHE can also incorporate user features. `fdhe` uses the concatenation of the user/item feature vector with the hash inputs (as with DHE) for the input to a deep neural network. This incorporates user/item features into the OOV embedding. However, compared to `dnn`, it also assigns a unique embedding for each user/item ID, even if they share the same features.

Algorithm 1: PyTorch-style pseudocode for the `m-lsh` OOV embedder.

```
# row_features: vector of the user/item features.
# oov_table: OOV embedding table.
# Each row of the table is an OOV bucket
def lsh_embed(row_features, oov_table):
    # lsh_hash is a binary vector
    lsh_hash = random_projection(row_features)
    # get col-wise mean of rows where vec is 1
    return oov_table[lsh_hash].mean(axis=1)
    # oov_table is updated via backpropagation
```

DNN. `dnn` is a simple feed-forward deep neural network that takes in the user/item features as input and outputs a real-valued vector. This embedder can be viewed as a modification to `fdhe` that omits the hash-related features. As a result, users/items with the same features will share the same embedding.

Mean LSH. `m-lsh` is a locality-sensitive hashing (LSH) [6] based OOV embedder. It uses a random projection matrix to map a user/item ID to a binary vector. It then uses this binary vector to index into the OOV embedding table and returns the column-wise mean of the rows where the binary vector is 1. This helps ensure that similar users/items have similar embeddings, even if their LSH vector is not exactly the same. The projection matrix remains constant, but the OOV embedding table values are updated during training. PyTorch-style pseudocode for this embedder can be found below in Algorithm 1.

Single LSH. `s-lsh` is similar to `m-lsh` but instead treats the binary vector as a single index into the OOV embedding table. This means similar users/items with the same LSH vector will have the same embedding. Conversely, users/items with different LSH hashes will have completely different embeddings. As with `m-lsh`, the projection matrix remains constant, but the OOV embedding table values are updated during training.

For all of the embedders, we implement *per-feature* normalization — we normalize each feature vector individually before concatenating them together. This is done to ensure that the distance between two users/items is not dominated by a single feature. Otherwise, long, dense features (like content embeddings) or lists of categorical

features (like watch history) could dominate the similarity computations for OOV embedding methods like KNN.

3.3 OOV Embedder Training

The training procedure does not need to be modified for the untrained OOV embedders (Section 3.1) — they can be applied to a pre-trained model. However, trained embedders (Section 3.2) add additional parameters that need to be optimized over OOV users/items. With a time-based inductive dataset split (details in Section 4), our training set contains only IV values, and the test set contains OOV values. OOV embedders are only used on OOV values so there is no training data for their parameters if only use the training set. As such, there are two main ways to generate OOV data in training for optimizing our OOV embedders: (1) withhold training data and use it as OOV samples or (2) generate synthetic OOV samples from the training data.

Withholding Data. Withholding training data to use as OOV samples is the simplest method, but it also reduces the amount of data available for training. This also complicates evaluation when benchmarking trained embedders against untrained embedders since the untrained embedders do not have access to the withheld data. Reducing the amount of data available for transductive training worsens transductive performance, violating the criteria defined in Section 1. For this reason, we choose to use synthetic OOV samples. However, the withholding data approach may be useful in production settings where we often cannot afford to maintain a unique embedding table entry for every user/item and may treat low-frequency IDs as OOV values.

Synthetic Data. A simple way to train OOV embedders without affecting existing performance is to generate synthetic OOV samples. For each user/item, we create an OOV version of it that has the same interactions. We then select a subset with ratio α of the OOV samples each epoch to use for training. We then perform feature masking, a common augmentation for self-supervised learning [47, 66], with mask rate β on the features of the OOV samples. This ensures that generated samples do not have the exact same features as the input samples. There are three types of OOV interactions: (IV user) $\rightarrow$ (OOV item), (OOV user) $\rightarrow$ (IV item), and (OOV user) $\rightarrow$ (OOV item). We generate each type with equal probability — although, in practice, this can be tuned to match the expected distribution of OOV interactions in production.

Maintaining Transductive Performance. When training our OOV embedder, our aim is to maintain the performance of the transductive portion of the model. For example, with synthetic training, interactions that only involve one OOV user/item will result in undesirable updates to the main embedding table. To avoid this, we split each epoch into two training steps. In the first step, we train the model on the original training data — as we normally would in transductive training. There are no OOV values at this point, so it does not affect any trainable parameters in the OOV embedder. In the second step, we freeze the main embedding table weights and train the model on the synthetic OOV samples. The only parameters that can be updated at this step are those of the OOV embedder. We also checkpoint and restore the optimizer state before and after the second step. This ensures that the OOV training does not affect the transductive portion of the model.

4 Datasets

As mentioned in Section 2, following suggestions from recent works [18, 46], we split the datasets based on a time t. We select t for each dataset by computing the first time each user/item appeared. We then select a time t such that 20% of the users/items are OOV. Formally, select t such that $|\mathcal{U}_{\text{train}}| + |\mathcal{I}_{\text{train}}| \approx 0.8(n+m)$. This results in a naturally different distribution of OOV users compared to OOV items for each of the four datasets. Plots of the relative user/item distributions can be seen in Figure 5. These dataset splits will help facilitate future benchmarking in the inductive setting.

We benchmark various transductive recommendation system methods across four different datasets. Below, we briefly describe how we processed each of the datasets. Representative statistics for each dataset can be found in Table 2.

Yelp. The `Yelp-2018` dataset consists of user reviews of businesses on Yelp from the 2018 Yelp Dataset Challenge[6]. We start with the version of the dataset provided by RecBole [64]. We then sample 75% of the users/items and perform 5-core filtering. We also clean up each feature by removing invalid values, normalizing floating point values, and imputing missing values with scikit-learn [36]. We also remove low-frequency values in categorical features and normalize all strings. Finally, we add text vectors for each business name. We use 300-dimensional GloVe [37] vectors for this purpose.

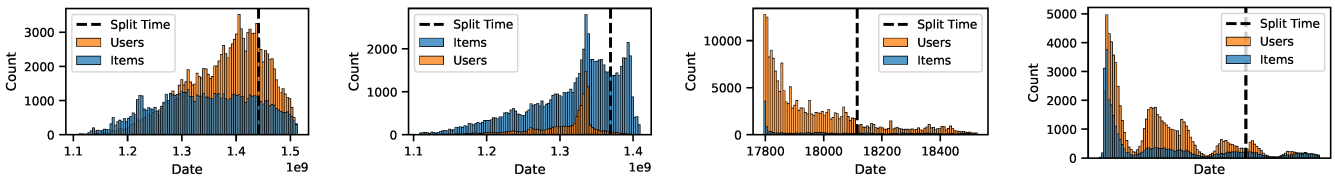
LastFM. The `LastFM-artists` dataset [41] consists of user/artist interactions on LastFM gathered in 2014. We start with the LastFM-1b version of the dataset provided by RecBole [64] and sample 10% of the users and items. We perform the feature cleaning as with the `Yelp-2018` dataset and add GloVe vectors for each artist's name.

H&M. The `H&M` dataset consists of user/item purchases on the H&M website. The raw dataset is taken from the H&M Kaggle competition[7] and we sample 30% of the users/items. We compute GloVe vectors for each item's name and use a pre-trained Vision Transformer [8] to extract features from each item's image. We also perform the same feature cleaning as with the `Yelp-2018` dataset.

Content. The `Content` dataset is a proprietary user-item interaction dataset from a large social platform serving hundreds of millions of daily active users. The data is gathered from 5 days of production traffic over users sampled from a single country. We only collect users who are 18 years old and above. The `Content` dataset has rich user/item features as with many production recommendation systems. Unfortunately, due to the large number of features, we were unable to train any context-aware models with our RecBole-based [64] evaluation framework.

Table 2: Statistics for each of the datasets used in this work. The number of float/dense features counts the number of distinct dense vectors, not the total number of floating point values (e.g., text embeddings count as a single float feature).

Dataset	IV Users / Items	OOV Users / Items	Mean User / Item Deg.	# User / Item Cat. Feat.	# User / Item Float Feat.
Yelp-2018	126,379 / 79,238	28,140 / 13,078	13.74 / 21.92	3 / 7	18 / 6
LastFM-artists	11,962 / 76,152	342 / 17,190	53.69 / 8.39	1 / 2	45 / 1
H&M	200,749 / 18,871	36,961 / 7,024	12.76 / 135.71	7 / 12	0 / 3
Content	74,700 / 30,757	6,610 / 3,941	24.47 / 59.42	57 / 339	172 / 899



(a) Yelp-2018 (b) LastFM-artists (c) H&M (d) Content

Figure 5: Visualization of where the inductive split occurs on the datasets. The x-axis is the time that the user/item first appeared. Everything to the left of the split time is used for training and validation. The remainder is used for evaluation.

Table 3: OOV user AUC of context-aware methods with different OOV embedding methods. Higher is better. The best-performing method in each column is bolded, and the second-best is underlined. Rows are sorted from lowest mean rank to highest mean rank. Reported results are the best in a hyperparameter grid search across 5 runs.

	H&M			LastFM-artists			Yelp-2018		
OOV Method	DCNV2	WideDeep	xDeepFM	DCNV2	WideDeep	xDeepFM	DCNV2	WideDeep	xDeepFM
fdhe	66.07	68.51	<u>72.1</u>	85.53	83.25	75.88	71.24	75.87	68.86
dhe	69.09	68.55	**74.12**	86.15	84.97	59.74	70.57	67.15	71.48
zero	<u>71.06</u>	71.21	69.06	81.57	86.57	84.75	71.16	72.43	76.04
knn	63.71	63.13	63.61	83.9	84.79	83.2	72.73	73.42	75.04
rand	55.17	70.49	66.76	82.14	<u>86.64</u>	85.52	78.95	74.75	73.68
r-bucket	65.48	70.61	68.37	**87.13**	85.79	84.24	<u>79.88</u>	70.97	76.04
dnn	63.87	<u>71.7</u>	71.09	86.65	84.71	<u>86.19</u>	76.23	<u>77.89</u>	<u>82.26</u>
s-lsh	**73.03**	**72.61**	70.64	86.37	79.71	85.86	78.52	76.03	80.68
mean	67.72	70.72	66.12	86.49	85.79	85.16	74.9	**79.82**	73.88
m-lsh	70.69	71.65	71.3	<u>86.93</u>	**86.67**	**86.78**	**79.96**	76.35	**82.48**

5 Experimental Evaluation

5.1 Evaluation Details

As mentioned in Section 2, we evaluate our OOV embedding methods on 5 different models: 2 context-free and 3 context-aware. We intentionally select popular, well-established industry-standard base models to allow us to draw widely applicable conclusions.

Evaluation Metrics. We evaluate the ranking and retrieval models separately. Following conventions from existing work [38, 52, 53, 58], we use ndcg@k (where k=20) for retrieval models and AUROC for ranking models. In Tables 3 and 4, we report the inductive performance of OOV users. It is worth noting that the transductive performance of IV users to IV items remains the same due to how we train the OOV embedding models (see Section 3.3).

[6]https://www.yelp.com/dataset
[7]https://www.kaggle.com/competitions/h-and-m-personalized-fashion-recommendations

Experimental Details. All models utilize a fork of the RecBole [63, 64] framework for experiments, in which we have made extensive modifications to the framework and models to support OOV values and swap between different OOV embedder types. We also added support for filtered evaluation on a subset of users/items. We were very careful to facilitate the easy addition of OOV support to new models. We run all experiments on Google Cloud Platform (GCP). Experiments are conducted on Google Compute Engine instances with NVIDIA Tesla P100 GPUs. The code, datasets, and hyperparameters for each of our experiments and embedders are available at: https://github.com/snap-research/improving-inductive-oov-recsys.

5.2 Context-Aware Results

The OOV user evaluation results of context-aware models are displayed in Table 3. On average, the best-performing OOV embedding

method is m-lsh and the worst is fdhe. Unfortunately, we were unable to train context-aware models on Content (even in the transductive setting) using our RecBole-based framework due to the large number of features and resulting stability issues. We make the following observations:

Table 4: OOV user NDCG@20 of context-free methods with different OOV embedding methods. Higher is better. The best-performing method in each column is bolded and the second-best is underlined.

Dataset	Yelp-2018		LastFM-artists		H&M		Content
Method	BPR	DAU	BPR	DAU	BPR	DAU	BPR
zero	0.79	0.79	0.93	0.93	1.15	1.15	0.71
fdhe	0.99	1.06	0.34	0.35	1.97	2.02	1.32
dhe	1.12	1.11	0.59	0.40	2.09	1.96	1.39
rand	4.05	0.83	15.09	0.71	2.80	1.89	0.94
s-lsh	9.13	1.05	46.92	0.79	**6.19**	2.02	1.05
r-bucket	9.33	1.22	41.78	0.76	6.02	1.38	1.38
dnn	2.94	**4.43**	0.38	0.44	2.94	**3.36**	1.87
mean	<u>9.40</u>	<u>2.89</u>	**48.38**	0.12	6.15	1.94	<u>3.74</u>
m-lsh	**9.49**	1.49	<u>47.85</u>	<u>1.13</u>	<u>6.16</u>	<u>2.15</u>	2.02
knn	6.95	1.58	45.69	**4.86**	5.23	1.67	**6.00**

Context helps OOV embeddings. From Table 3, we can see that incorporating contextual information generally helps OOV embeddings. 3/4 of the best-performing OOV embedding models utilize context information. This aligns with our intuition: similar users/items should have similar embeddings. This is true for both context-free and context-aware models. In some cases, like with xDeepFM on Yelp-2018, the gap in AU-ROC on OOV users is as large as 6 points — showing that incorporating feature information in OOV handling can drastically improve an RS's ability to generalize to OOV users/items.

LSH-based solutions perform well. Both m-lsh and s-lsh work well for the context-aware models, with one of the two methods performing the best on 6/9 model/dataset combinations, as shown in Table 3. Across the context-aware model experiments, m-lsh and s-lsh show a mean improvement of 3.74% and 2.58% over r-bucket (a common industry standard[2]), respectively. They also perform well compared to the next-best method, mean, showing respective average improvements of 3.45% and 2.25%.

DHE-based solutions perform poorly. dhe and fdhe are the methods with the lowest average rank across the model/dataset combinations shown in Table 3. Surprisingly, we find that zero generally outperforms both dhe and fdhe. This is likely due to the additional noise the multiple hash inputs introduce to DHE-style models — a different ID results in a completely different embedding.

5.3 Context-Free Results

Table 4 shows the NDCG@20 for OOV users of BPR and DirectAU. m-lsh has the highest mean rank of the different OOV embedding methods. Unlike in the context-aware setting, a relatively large gap exists between different base models on the same dataset. Surprisingly, BPR outperforms DirectAU on OOV users across all three datasets. We make the following observations about OOV embedding methods on the context-free models:

Improving context-free OOV performance is difficult. Both BPR and DirectAU exhibit poor performance on most datasets, regardless of OOV embedder choice. This shows that it is difficult to encode feature information from OOV IDs in a useful manner for context-free models.

OOV embedder choice is extremely important. From Table 4, we can observe a large gap between the best-performing models on each dataset and the worst-performing models for context-free models. This is especially true for BPR on LastFM-artists, where there is a 48.04 gap between the best-performing mean embedder and the worst-performing fdhe embedder. fdhe and dhe exhibit similarly poor performance across the four datasets.

5.4 Recommendations for Practitioners

Since the performance of each OOV embedder greatly depends on the dataset and method, there is no silver bullet method. However, based on the results of our experiments in Tables 3 and 4, we make the following recommendations for practitioners aiming to improve their performance on OOV users/items:

(1) **If contextual information (features) is available**, try using m-lsh. Across our experiments, m-lsh generally performs the best. An advantage of m-lsh over s-lsh is that it results in fewer collisions (see Table 1). It can also be trivially computed directly on the GPU and efficiently implemented through data structures like PyTorch's [35] EmbeddingBag.

(2) **If no features are available and collisions are not important**, consider using mean. It is extremely cheap to compute and, based on our experiments, is the best-performing untrained OOV embedder. However, all IDs will receive the same embedding, making it particularly problematic for context-free models.

(3) **If only users *or* items have features**, OOV embedding methods can be mixed. For example, in a dataset with user features but no item features, m-lsh could be used for users and mean for items. This approach can also be used in any case where the user/item ID distributions are significantly different.

6 Related Work

Cold-Start RS Methods. A known issue in recommendation systems is the cold-start problem [27], which is when low-degree users and items receive poorer quality recommendations. In this work, we look specifically at the problem of OOV users/items, which means they occur exactly *zero* times in the training examples. However, cold-start methods often focus on the transductive setting where all the users/items appear at train time (although some may have very few interactions). Vartak et al. [48] focuses on the case of OOV items and proposes a meta-learning approach that uses a classifier based on user history to adjust model parameters. Wang et al. [51] extends Model-Agnostic Meta-Learning (MAML) [9] for improving cold-start recommendation performance. The Aligning Distillation (ALDI) framework [16] applies knowledge distillation by treating warm items as teachers and cold items as students. Zhu

[7]The exact complexity here is difficult to compute since we rely on approximate nearest neighbor search [12, 19].

et al. [67] applies a transformation function consisting of a mixture-of-experts model to transform feature information to collaborative filtering representations. DropoutNet [49] uses input dropout during RS model training to improve the model's generalization to missing features. Lam et al. [24] proposes a probabilistic atpproach to handling OOV users on MovieLens [14].

Cold-Start Graph Methods. Recommendation systems can be formulated as a *link prediction* problem on a bipartite graph [32, 55], where edges represent interactions between users and items. As such, we also briefly discuss existing literature focused on improving cold-start performance on graph-related tasks. These include both training-based [15, 20, 33, 65] and augmentation-based [13, 39, 55, 62] approaches. However, due to model architecture and training differences, they are not straightforward to apply to RS.

7 Conclusion

In this work, we explored the inductive setting in recommendation systems, where we focused on finding the best method to handle previously unseen (OOV) values. We evaluated nine different OOV embedder methods that are efficient, model-agnostic, and guaranteed to maintain transductive performance. To the best of our knowledge, this is the first comprehensive empirical study of the performance of various OOV methods for recommendation systems. Our results show that, of the nine methods, the locality-sensitive-hashing-based methods tend to be the most effective in improving inductive performance. Additionally, we augment and re-release three inductive datasets to facilitate future study of inductive performance and OOV methods in recommendation system problems. Furthermore, we derive a set of four recommendations for industrial practitioners to improve their inductive recommendation systems performance and alleviate pain points in dealing with OOV values. We hope this work encourages both academic and industrial researchers to further explore the inductive and OOV settings, considering their immediate practical impact in real-world, production-scale recommendation systems.

Acknowledgments

UCR coauthors were partly supported by the National Science Foundation under CAREER grant no. IIS 2046086 and were also sponsored by the Combat Capabilities Development Command Army Research Laboratory under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding parties.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Jean-Philippe Aumasson and Daniel J Bernstein. 2012. SipHash: a fast short-input PRF. In *International Conference on Cryptology in India*. Springer, 489–508.

[3] Xuheng Cai, Chao Huang, Lianghao Xia, and Xubin Ren. 2023. LightGCL: Simple Yet Effective Graph Contrastive Learning for Recommendation. *arXiv preprint arXiv:2302.08191* (2023).

[4] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.

[5] Benjamin Coleman, Wang-Cheng Kang, Matthew Fahrbach, Ruoxi Wang, Lichan Hong, Ed H. Chi, and Derek Zhiyuan Cheng. 2023. Unified Embedding: Battle-Tested Feature Representations for Web-Scale ML Systems. In *NeurIPS*.

[6] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*. 253–262.

[7] Ruihai Dong, Michael P O'Mahony, Markus Schaal, Kevin McCarthy, and Barry Smyth. 2016. Combining similarity and sentiment in opinion mining for product recommendation. *Journal of Intelligent Information Systems* 46, 2 (2016), 285–312.

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[9] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*. PMLR, 1126–1135.

[10] Benjamin Ghaemmaghami, Mustafa Ozdal, Rakesh Komuravelli, Dmitriy Korchev, Dheevatsa Mudigere, Krishnakumar Nair, and Maxim Naumov. 2022. Learning to Collide: Recommendation System Model Compression with Learned Hash Functions. *ArXiv* abs/2203.15837 (2022). https://api.semanticscholar.org/CorpusID:247794181

[11] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *Vldb*, Vol. 99. 518–529.

[12] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*. PMLR, PMLR, Cambridge, MA, USA, 3887–3896.

[13] Zhichun Guo, Tong Zhao, Yozen Liu, Kaiwen Dong, William Shiao, Neil Shah, and Nitesh V Chawla. 2024. Node Duplication Improves Cold-start Link Prediction. *arXiv preprint arXiv:2402.09711* (2024).

[14] 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.

[15] Weihua Hu, Kaidi Cao, Kexin Huang, Edward W Huang, Karthik Subbian, and Jure Leskovec. 2022. Tuneup: A training strategy for improving generalization of graph neural networks. *arXiv preprint arXiv:2210.14843* (2022).

[16] Feiran Huang, Zefan Wang, Xiao Huang, Yufeng Qian, Zhetao Li, and Hao Chen. 2023. Aligning distillation for cold-start item recommendation. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1147–1157.

[17] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. 2333–2338.

[18] Yitong Ji, Aixin Sun, Jie Zhang, and Chenliang Li. 2023. A critical study on data leakage in recommender system offline evaluation. *ACM Transactions on Information Systems* 41, 3 (2023), 1–27.

[19] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

[20] Mingxuan Ju, Tong Zhao, Wenhao Yu, Neil Shah, and Yanfang Ye. 2024. GRAPHPATCHER: mitigating degree bias for graph neural networks via test-time augmentation. *Advances in Neural Information Processing Systems* 36 (2024).

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

[22] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 426–434.

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

[24] Xuan Nhat Lam, Thuc Vu, Trong Duc Le, and Anh Duc Duong. 2008. Addressing cold-start problem in recommendation systems. In *Proceedings of the 2nd International Conference on Ubiquitous Information Management and Communication* (Suwon, Korea) *(ICUIMC '08)*. Association for Computing Machinery, New York, NY, USA, 208–211. doi:10.1145/1352793.1352837

[25] Daniel Lee and H Sebastian Seung. 2000. Algorithms for non-negative matrix factorization. *Advances in neural information processing systems* 13 (2000).

[26] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD*

international conference on knowledge discovery & data mining. 1754–1763.
[27] Blerina Lika, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. 2014. Facing the cold start problem in recommender systems. *Expert systems with applications* 41, 4 (2014), 2065–2073.
[28] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* 7, 1 (2003), 76–80.
[29] Zhuoran Liu, Leqi Zou, Xuan Zou, Caihua Wang, Biao Zhang, Da Tang, Bolin Zhu, Yijie Zhu, Peng Wu, Ke Wang, et al. 2022. Monolith: real time recommendation system with collisionless embedding table. *arXiv preprint arXiv:2209.07663* (2022).
[30] Zhu-Ping Liu, Leqi Zou, Xuan Zou, Caihua Wang, Biao Zhang, Dandan Tang, Bolin Zhu, Yijie Zhu, Pengfei Wu, K. Wang, and Youlong Cheng. 2022. Monolith: Real Time Recommendation System with Collisionless Embedding Table. *ArXiv* (2022).
[31] Johannes V Lochter, Renato M Silva, and Tiago A Almeida. 2020. Deep learning models for representing out-of-vocabulary words. In *Brazilian Conference on Intelligent Systems*. Springer, 418–434.
[32] Haitao Mao, Juanhui Li, Harry Shomer, Bingheng Li, Wenqi Fan, Yao Ma, Tong Zhao, Neil Shah, and Jiliang Tang. 2023. Revisiting link prediction: A data perspective. *arXiv preprint arXiv:2310.00793* (2023).
[33] Haitao Mao, Juanhui Li, Harry Shomer, Bingheng Li, Wenqi Fan, Yao Ma, Tong Zhao, Neil Shah, and Jiliang Tang. 2024. Revisiting Link Prediction: a data perspective. In *The Twelfth International Conference on Learning Representations*.
[34] Sabrina J Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y Lee, Benoît Sagot, et al. 2021. Between words and characters: a brief history of open-vocabulary modeling and tokenization in nlp. *arXiv preprint arXiv:2112.10508* (2021).
[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019), 1–12.
[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[37] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.
[38] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618* (2012).
[39] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2019. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903* (2019).
[40] Aravind Sankar, Yozen Liu, Jun Yu, and Neil Shah. 2021. Graph neural networks for friend ranking in large-scale social platforms. In *Proceedings of the Web Conference 2021*. 2535–2546.
[41] Markus Schedl. 2016. The lfm-1b dataset for music retrieval and recommendation. In *Proceedings of the 2016 ACM on international conference on multimedia retrieval*. 103–110.
[42] Tobias Schnabel, Mengting Wan, and Longqi Yang. 2022. Situating Recommender Systems in Practice: Towards Inductive Learning and Incremental Updates. *arXiv preprint arXiv:2211.06365* (2022).
[43] Shalin Shah. 2023. A Survey of Latent Factor Models for Recommender Systems and Personalization. *Authorea Preprints* (2023).
[44] Jiahui Shi, Vivek Chaurasiya, Yozen Liu, Shubham Vij, Yan Wu, Satya Kanduri, Neil Shah, Peicheng Yu, Nik Srivastava, Lei Shi, et al. 2023. Embedding Based Retrieval in Friend Recommendation. (2023).
[45] Aixin Sun. 2023. On Challenges of Evaluating Recommender Systems in an Offline Setting. In *Proceedings of the 17th ACM Conference on Recommender Systems*. 1284–1285.
[46] Aixin Sun. 2023. Take a Fresh Look at Recommender Systems from an Evaluation Standpoint. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2629–2638.
[47] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Mehdi Azabou, Eva L. Dyer, Rémi Munos, Petar Velickovic, and Michal Valko. 2022. Large-Scale Representation Learning on Graphs via Bootstrapping. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, Virtual, 1–18. https://openreview.net/forum?id=0UXT6PpRpW
[48] Manasi Vartak, Arvind Thiagarajan, Conrado Miranda, Jeshua Bratman, and Hugo Larochelle. 2017. A meta-learning perspective on cold-start recommendations for items. *Advances in neural information processing systems* 30 (2017).
[49] Maksims Volkovs, Guangwei Yu, and Tomi Poutanen. 2017. Dropoutnet: Addressing cold start in recommender systems. *Advances in neural information processing systems* 30 (2017).
[50] Chenyang Wang, Yuanqing Yu, Weizhi Ma, Min Zhang, Chong Chen, Yiqun Liu, and Shaoping Ma. 2022. Towards representation alignment and uniformity in collaborative filtering. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1816–1825.
[51] Li Wang, Binbin Jin, Zhenya Huang, Hongke Zhao, Defu Lian, Qi Liu, and Enhong Chen. 2021. Preference-Adaptive Meta-Learning for Cold-Start Recommendation.. In *IJCAI*. 1607–1614.
[52] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*. 1–7.
[53] Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. 2021. Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the web conference 2021*. 1785–1797.
[54] Tongzhou Wang and Phillip Isola. 2020. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *International Conference on Machine Learning*. PMLR, 9929–9939.
[55] Yu Wang, Tong Zhao, Yuying Zhao, Yunchao Liu, Xueqi Cheng, Neil Shah, and Tyler Derr. 2024. A topological perspective on demystifying gnn-based link prediction performance. In *ICLR*.
[56] Christopher Wellons. 2018. Hash Function Prospector. https://github.com/skeeto/hash-prospector.
[57] Xinyi Wu, Donald Loveland, Runjin Chen, Yozen Liu, Xin Chen, Leonardo Neves, Ali Jadbabaie, Clark Mingxuan Ju, Neil Shah, and Tong Zhao. 2024. GraphHash: Graph Clustering Enables Parameter Efficiency in Recommender Systems. *arXiv preprint arXiv:2412.17245* (2024).
[58] Liangwei Yang, Zhiwei Liu, Chen Wang, Mingdai Yang, Xiaolong Liu, Jing Ma, and Philip S Yu. 2023. Graph-based Alignment and Uniformity for Recommendation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. 4395–4399.
[59] Junliang Yu, Xin Xia, Tong Chen, Lizhen Cui, Nguyen Quoc Viet Hung, and Hongzhi Yin. 2023. XSimGCL: Towards extremely simple graph contrastive learning for recommendation. *IEEE Transactions on Knowledge and Data Engineering* (2023).
[60] Caojin Zhang, Yicun Liu, Yuanpu Xie, Sofia Ira Ktena, Alykhan Tejani, Akshay Gupta, Pranay Kumar Myana, Deepak Dilipkumar, Suvadip Paul, Ikuhiro Ihara, et al. 2020. Model size reduction using frequency based double hashing for recommender systems. In *Proceedings of the 14th ACM Conference on Recommender Systems*. 521–526.
[61] Caojin Zhang, Yicun Liu, Yuanpu Xie, Sofia Ira Ktena, Alykhan Tejani, Akshay Gupta, Pranay K. Myana, Deepak Dilipkumar, Suvadip Paul, Ikuhiro Ihara, Prasang Upadhyaya, Ferenc Huszár, and Wenzhe Shi. 2020. Model Size Reduction Using Frequency Based Double Hashing for Recommender Systems. In *Proceedings of the 14th ACM Conference on Recommender Systems*.
[62] Tong Zhao, Gang Liu, Daheng Wang, Wenhao Yu, and Meng Jiang. 2022. Learning from counterfactual links for link prediction. In *International Conference on Machine Learning*. PMLR, PMLR, Cambridge, MA, 26911–26926.
[63] Wayne Xin Zhao, Yupeng Hou, Xingyu Pan, Chen Yang, Zeyu Zhang, Zihan Lin, Jingsen Zhang, Shuqing Bian, Jiakai Tang, Wenqi Sun, et al. 2022. RecBole 2.0: towards a more up-to-date recommendation library. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*.
[64] Wayne Xin Zhao, Shanlei Mu, Yupeng Hou, Zihan Lin, Yushuo Chen, Xingyu Pan, Kaiyuan Li, Yujie Lu, Hui Wang, Changxin Tian, et al. 2021. Recbole: Towards a unified, comprehensive and efficient framework for recommendation algorithms. In *proceedings of the 30th acm international conference on information & knowledge management*.
[65] Wenqing Zheng, Edward W Huang, Nikhil Rao, Sumeet Katariya, Zhangyang Wang, and Karthik Subbian. 2021. Cold brew: Distilling graph node representations with incomplete or missing neighborhoods. *arXiv preprint arXiv:2111.04840* (2021).
[66] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2020. Deep Graph Contrastive Representation Learning. *CoRR* abs/2006.04131 (2020), 1–17. arXiv:2006.04131 https://arxiv.org/abs/2006.04131
[67] Ziwei Zhu, Shahin Sefati, Parsa Saadatpanah, and James Caverlee. 2020. Recommendation for new users and new items via randomized training and mixture-of-experts transformation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 1121–1130.