Empowering Long-tail Item Recommendation through Cross Decoupling Network (CDN)

Yin Zhang
yinzh@google.com
Google Research, Brain Team, USA

Ruoxi Wang
ruoxi@google.com
Google Research, Brain Team, USA

Derek Zhiyuan Cheng
zcheng@google.com
Google Research, Brain Team, USA

Tiansheng Yao
tyao@google.com
Google Research, Brain Team, USA

Xinyang Yi
xinyang@google.com
Google Research, Brain Team, USA

Lichan Hong
lichan@google.com
Google Research, Brain Team, USA

James Caverlee
caverlee@cse.tamu.edu
Texas A&M University, USA

Ed H. Chi
edchi@google.com
Google Research, Brain Team, USA

ABSTRACT

Industry recommender systems usually suffer from highly-skewed long-tail item distributions where a small fraction of the items receives most of the user feedback. This skew hurts recommender quality especially for the item slices without much user feedback. While there have been many research advances made in academia, deploying these methods in production is very difficult and very few improvements have been made in industry. One challenge is that these methods often hurt overall performance; additionally, they could be complex and expensive to train and serve.

In this work, we aim to improve tail item recommendations while maintaining the overall performance with less training and serving cost. We first find that the predictions of user preferences are biased under long-tail distributions. The bias comes from the differences between training and serving data in two perspectives: 1) the item distributions, and 2) user's preference given an item. Most existing methods mainly attempt to reduce the bias from the item distribution perspective, ignoring the discrepancy from user preference given an item. This leads to a severe forgetting issue and results in sub-optimal performance.

To address the problem, we design a novel Cross Decoupling Network (CDN) to reduce the two differences. Specifically, CDN (i) decouples the learning process of memorization and generalization on the item side through a mixture-of-expert architecture; (ii) decouples the user samples from different distributions through a regularized bilateral branch network. Finally, a new adapter is introduced to aggregate the decoupled vectors, and softly shift the training attention to tail items. Extensive experimental results show that CDN significantly outperforms state-of-the-art approaches on popular benchmark datasets. We also demonstrate its effectiveness by a case study of CDN in a large-scale recommendation system at Google.

KDD '23, August 6–10, 2023, Long Beach, CA, USA

ACM ISBN 979-8-4007-0103-0/23/08.
https://doi.org/10.1145/3580305.3599814

CCS CONCEPTS

• **Information systems** → **Information retrieval**.

KEYWORDS

decoupling, recommendation, memorization and generalization

ACM Reference Format:
Yin Zhang, Ruoxi Wang, Derek Zhiyuan Cheng, Tiansheng Yao, Xinyang Yi, Lichan Hong, James Caverlee, and Ed H. Chi. 2023. Empowering Long-tail Item Recommendation through Cross Decoupling Network (CDN). In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3580305.3599814

1 INTRODUCTION

In industry recommender systems, user feedback towards items usually exhibits server long-tail distributions. That is, a small fraction of items (*head items*) are extremely popular and receive most of the user feedback, while rest of the items (*tail items*) have very little if any user feedback. Recommender models that are trained based on the long-tail data usually amplify head items, enable the "rich get richer" effect while hurting long-term user satisfaction. Models trained on highly skewed data distribution may lead to even worse skewness in real-world applications. Hence, it's critical to address the long-tail distribution problem in industry recommenders.

There have been some methods that are successfully deployed in production models to alleviate the long-tail distribution influence, for example, logQ corrections [18, 28] and re-sampling. However, further improvements in this area for industry models are very limited, despite many research advances made in academia [32]. There are several challenges which make putting these research in production models difficult. First, many work targeting long-tail performance would hurt head or overall performance, directly impacting top line business metrics. Second, production models have very strict latency requirements for real-time inference, while many existing techniques are complex and expensive to serve. Third, production models prefer simplicity for easy adoption and maintenance. Many research (*e.g.*, meta-learning [15], transfer learning [30]) are difficult to be productionized due to these challenges.

In our work, we aim to improve tail item recommendations while maintaining the overall performance with less training and serving

cost. We draw our inspiration from recent work with great success in computer vision [14]. Its core idea is that representation learning and classification learning requires different data distributions, where traditionally models don't consider such decoupling for model parameters. Specifically, they propose a two-stage decoupling training strategy, where the first stage trains on the original long-tail distribution for item representation learning, and the second stage trains on the re-balanced data to improve the predictions of tail items. However, in recommendation application, we empirically observe that these methods suffer from a severe *forgetting issue* [22]. This means that the learned knowledge of certain parts of items (*e.g.* head) in the first training stage are easily forgotten when the learning focus is shifted to other items (*e.g.* tail) in the second training stage, leading to a degradation in overall model quality (as shown in Figure 1). Moreover, in large-scale production systems, two-stage training is much more complex to achieve and maintain than co-training scheme. In light of the pros and cons of this method, we target on developing a model that improves the decoupling technique to accommodate web-scale recommenders.

Like many other methods tackling cold-start problems, the decoupling methods potentially hurt the overall recommendation performance. We attempt to understand this from a theoretical point of view. In particular, we found that the prediction of user preference towards an item is biased. The bias comes from the differences between training and serving data in two perspectives: 1) the item distributions, and 2) user's preference given an item. Most existing methods mainly attempt to reduce the bias from the item distribution perspective, ignoring the discrepancy from user preference given an item.

Motivated by the theoretical findings, we propose a novel Cross Decoupling Network (CDN) framework to mitigate the forgetting issue, by considering decoupling from both item and user sides. "Decoupling" means we treat the corresponding learning as two independent processes. In more details:

- On the item side, the amount of user feedback received by head items and tail items vary significantly. This variations may cause the model to forget the learned knowledge of head (more memorization), when its attention is shifted to tail (more generalization) [1]. Hence, we propose to decouple the memorization and generalization for the item representation learning. In particular, we first group features into memorization and generalization related features, and feed them separately into a memorization-focused expert and a generalization-focused expert, which are then aggregated through a frequency-based gating. This mixture-of-expert structure allows us to dynamically balance memorization and generalization abilities for head and tail items.
- On the user side, we leverage a regularized bilateral branch network to decouple user samples from two distributions. The network consists of two branches: a "main" branch that trains on the original distribution for a high-quality representation learning; and a new "regularizer" branch that trains on the re-balanced distribution to add more tail information to the model. These two branches share some hidden layers and are jointly trained to mitigate the forgetting issue. Shared tower on the user side are used for scalability.

[1]Memorization is to learn and exploit the existing knowledge of visited training data. Generalization, on the other hand, is to explore new knowledge that has not occurred in the training dataset based on the transitivity (*e.g.* data correlation) [5].

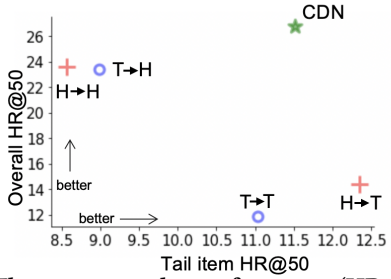

Figure 1: The recommender performance (HR@50) of different methods on tail items (x-axis) and overall items (y-axis). Dots and plus signals represents four two-stage decoupling methods. 'H' means focusing on head items, 'T' means focusing on tail items, and → means switching from the 1st to the 2nd stage. When tail (T) is focused in the second stage, the performance on tail items improves; however the overall performance significantly degrades. Our model CDN achieves excellent performance for both overall and tail item performances.

Finally, a new adapter (called γ-adapter) is introduced to aggregate the learned vectors from the user and item sides. By adjusting the hyperparameter γ in the adapter, we are able to shift the training attention to tail items in a soft and flexible way based on different long-tail distributions.

The resulting model CDN mitigates the forgetting issue of existing models, where it not only improves tail item performance, but also preserves or even improves the overall performance. Further, it adopts a co-training scheme that is easy to maintain. These all make CDN suitable for industrial caliber applications.

The contributions of this paper are 4-fold:

- We provide a theoretical understanding on how the long-tail distribution influences recommendation performance from both item and user perspectives.
- We propose a novel cross decoupling network that decouples the learning process of memorization and generalization, and the sampling strategies. A γ-adapter is utilized to aggregate the learning from the two sides.
- Extensive experimental results on public dataset show CDN significantly outperforms the SOTA method, improving performance on both overall and tail item recommendations.
- We further provide a case study of applying CDN to a large-scale recommender system at Google. We show that CDN is easy to adapt to real setting and achieves significant quality improvements both offline and online.

2 LONG-TAIL DISTRIBUTION IN RECOMMENDATION AND MOTIVATION

Problem Settings. Our goal is to predict user engagement (e.g. clicks, installs) of candidates (e.g. videos, apps) that are in long-tail distributions. We start with formulating the problem: Given a set of users $\mathcal{U} = \{1, 2, \ldots, m\}$, a set of items $\mathcal{I} = \{1, 2, \ldots, n\}$ and their content information (*e.g.* item tags, categories). Let $\hat{d}(u, i)$

Table 1: Notations.

Notation	Explanation
$\mathcal{U}$, $\mathcal{I}$	user set, item set
$\mathbf{u}$, $\mathbf{i}$	user u (item i) feature vector
$\hat{d}(u,i)$, $d(u,i)$	user feedback in the traning/test set
$\hat{p}(i\|u)$, $p(i\|u)$	given user u, the estimated (by training data)/true probability of picking item i from candidate pool $\mathcal{I}$
$\hat{p}(i)$, $p(i)$	estimated/true item prior probability
$\hat{p}(i\|u)$, $p(u\|i)$	estimated/true conditional probability of user click when item i is known
Ω_m, Ω_r	training dataset in main/regularizer branch
$\mathbf{x}_m$, $\mathbf{y}_m$	user/item representation in main branch
$\mathbf{x}_r$, $\mathbf{y}_r$	user/item representation in regularizer branch
α	learning adapter

be the user u's feedback (e.g. click, install) towards item i in the training set: $\hat{d}(u,i) = 1$ (considered a positive) when u provides a click/install to item i, otherwise it is 0 (considered a negative). Let $d(u,i)$ be the user feedback towards items in the test set, which we aim to predict.

In the training set, the frequency of the user's positive feedback towards item i could be described as $\sum_{u\in\mathcal{U}} \hat{d}(u,i)$, which follows a highly skewed long-tail distribution across items [20]. This means a few head items receive most of the users' feedback, while most items receive little user feedback (tail items). Such imbalance is often quantified by the imbalance factor IF [13, 15] $max_{i,j\in\mathcal{I}} \sum_{u\in\mathcal{U}} \hat{d}(u,i) / \sum_{u\in\mathcal{U}} \hat{d}(u,j)$.

In industry, IF of item distribution could be very large (*e.g.*, >100000 in production datasets), making it extremely challenging. Furthermore, it also requires the overall performance should be neutral/improvement for production usage. Hence, we target improving recommendation performance for tail items, while maintaining or even improving the overall performance. Key notations are listed in Table 1.

Theoretical Analysis and Model Design Motivations. By Bayes' theorem, we theoretically show the long-tail distribution influence comes from two perspectives:

- The item distribution discrepancy between training and serving data: the item distribution between the training and serving datasets are often different (a.k.a., training / serving skew), and we refer to this as the discrepancy of the item *prior probability* $p(i)$. This causes the model to have a large bias on serving data, even if the model fits the training data well.
- The discrepancy of user preference given items between training and serving data: the estimated user preference under the long-tail distribution settings could be biased, and we refer to this as the discrepancy of the *conditional probability* $p(u|i)$. This is especially true for tail items, where the model underfits due to a lack of training examples. This underfitting leads to a large bias even for training data, let alone serving data.

Concretely, for a given user u, recommenders aim to accurately approximate the user's preference towards item i. The probability of such preference is commonly estimated by the softmax function in recommendation [28]:

$$p(i|u) = \frac{e^{s(u,i)}}{\sum_{j\in\mathcal{I}} e^{s(u,j)}}, \tag{1}$$

where $s(u,i)$ is the true user u preference towards the item i, and could be computed through an inner product between user and item embeddings [28].

Using Bayes' theorem, we find the estimated $\hat{p}(i|u)$ based on the training set could be described as:

$$\hat{p}(i|u) = \frac{e^{s(u,i) - \log\frac{p(i)}{\hat{p}(i)} - \log\frac{p(u|i)}{\hat{p}(u|i)}}}{\sum_{j\in\mathcal{I}} e^{s(u,j) - \log\frac{p(j)}{\hat{p}(j)} - \log\frac{p(u|j)}{\hat{p}(u|j)}}}, \tag{2}$$

where $\hat{p}(i)$ and $\hat{p}(u|i)$ are the estimated probabilities learned from the long-tail training dataset. We see that in order for $\hat{p}(i|u)$ to closely approximate $p(i|u)$ (comparing Eq. (1) and Eq. (2)), we need both of the estimated prior and conditional probability to well approximate the true probabilities, *i.e.*, $log\frac{p(i)}{\hat{p}(i)} \rightarrow 0$ and $log\frac{p(u|i)}{\hat{p}(u|i)} \rightarrow 0$. Motivated by this, in the following we propose a new model to address the long-tail distribution problem from these two perspectives.

3 CROSS DECOUPLING NETWORK

Based on the analysis, we propose a scalable cross decoupling network (CDN) that addresses the two discrepancies on the item and user sides. The main architecture is depicted in Figure 2.

- On the item side, we propose to decouple the memorization and generalization for head and tail item representation learning. To do so, we utilize a gated mixture of experts (MOE) architecture. In our version of MOE, we feed memorization related features into expert sub networks that focus on memorization. Similarly, we feed content related features into expert sub networks that dedicate for generalization. A gate (*i.e.*, a learnable function) is introduced to decide how much weight the model should put on for memorization and generalization for an item representation. The enhanced item representation learning can mitigate the item distribution discrepancy [13].
- On the user side, we decouple the user sampling strategies through a regularized bilateral branch network to reduce the user preference discrepancy. The network consists of a main branch for a universal user preference learning, and a regularizer branch to compensate for the sparsity of user feedback towards tail items. A shared tower between two branches are used for scaling up to production usage.

Finally, we cross combine the user and item learning to learn the users' diverse preference towards head and tail items in the long-tail distributions with a γ-adapter.

3.1 Item Memorization and Generalization Decoupling

We introduce the concepts of memorization features and generalization features, and then describe our method to decouple them through a gated mixture-of-expert architecture.

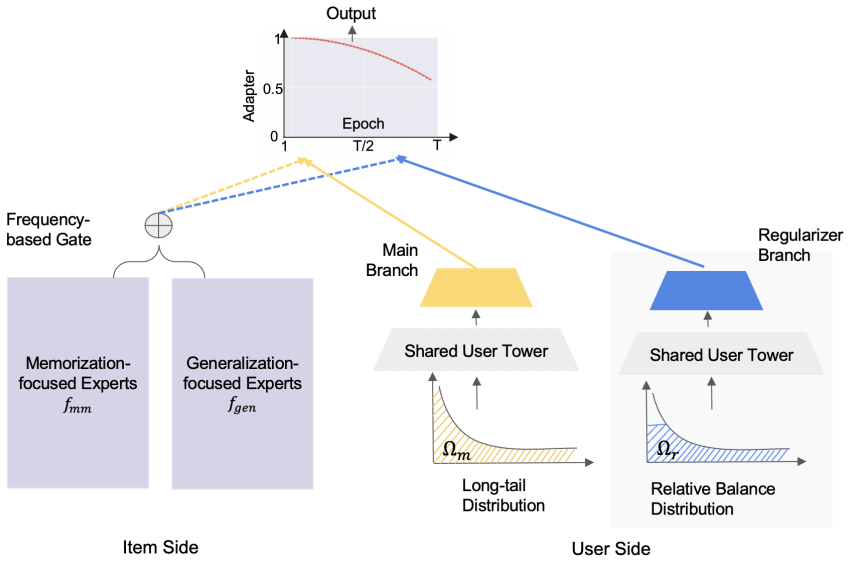

Figure 2: Cross Decoupling Network (CDN).

3.1.1 Features for memorization and generalization. Industry recommender systems usually consider hundreds of features as model inputs. Instead of encoding those features in the same way, we begin by splitting those features into two groups: memorization features and generalization features.

Memorization features. They help memorize the interactions between user and item (collaborative signal) in the training data, such as the item ID. Formally, these features are often categorical features that satisfy:

- **Uniqueness**: for its feature space $\mathcal{V}$, $\exists\ f_{in}$ that satisfies f_{in} is an injective function and $f_{in} : \mathcal{I} \rightarrow \mathcal{V}$, and;
- **Independence**: for $\forall\ v_1, v_2 \in \mathcal{V}$, the change of v_1 does not influence v_2;

In production recommenders, these features are typically represented by embeddings. Their embedding parameters can only be updated by the corresponding item (uniqueness), and do not share any information with other items (independence). Hence, they only memorize the information for a specific item, and cannot generalize to other existing or unseen items. In the meantime, these features also demonstrate a long-tail distribution due to the uniqueness. Therefore, for features that correspond to head items, their embeddings are updated more often, leading to a remarkable memorization effect. While for features corresponding to tail items, their embeddings may be more noisy due to a lack of gradient update.

Generalization features. Generalization features are those that can learn the correlations between user preference and item features, and that can be generalized to other items. These features are either shared across different items (*e.g.*, item categories, tags), or are continuous features. Hence, they can generalize to other existing or unseen items, and are important in improving tail item representation learning.

3.1.2 Item representation learning. We adopt the mixture-of-expert (MoE) structure with a frequency-based gating to decouple the memorization features and generalization features. The structure is depicted on the Item (left) side of Figure 2.

That is, for a training example $(\mathbf{u}, \mathbf{i})$, the item embedding is represented as:

$$\mathbf{y} = \sum_{k=1}^{n_1} G(\mathbf{i})_k E_k^{mm}(\mathbf{i}_{mm}) + \sum_{k=n_1+1}^{n_1+n_2} G(\mathbf{i})_k E_k^{gen}(\mathbf{i}_{gen}) \tag{3}$$

where $E_k^{mm}(\cdot)$ represent the memorization-focused experts which take all memorization features $\mathbf{i}_{mm}$ (*e.g.*, item ID) by concatenating their embeddings as input ; $E_k^{gen}(\cdot)$ represent generalization-focused experts which take all generalization features $\mathbf{i}_{gen}$ (*e.g.*, item categories) by concatenating their embeddings as input; and $G(\cdot)$ is the gating function with $G(\mathbf{i})_k$ being its k-th element, and $\sum_{k=1}^{n_1+n_2} G(\mathbf{i}) = 1$. The gating here is critical for dynamically balancing memorization and generalization for head and tail items. Intuitively, the gate could take item frequency as the input, and transform it through a non-linear layer: $g(\mathbf{i}) = softmax(\mathbf{W}\mathbf{i}_{freq})$, where $\mathbf{W}$ is a learnable weight matrix. It could also take input from other features, and we found feeding the item popularity as input to work pretty well empirically.

The proposed mechanism uncovers the differences for items in the long-tail distribution in a simple and elegant way for enhanced item representation learning. By decoupling memorization and generalization, the head items can achieve better memorization ability and tail items can be more generalized in the same time. As indicated by [13], the enhanced item representation learning can help compensate the inconsistency of conditional distribution between $p(u|i)$ and $\hat{p}(u|i)$. Additionally, by decoupling the memorization and generalization through experts with frequency-based gates, we are able to mitigate the forgetting issue when the learning attention is shifted towards tail items. That is, with the decoupling, when the training attention shifts to tail items, the gradients (knowledge) from tail items will mainly update the model parameters in the generalization-focused expert, while maintaining the well-learned memorization expert from head items.

3.2 User Sample Decoupling

As shown on the user (right) side of Figure 2, inspired by [14, 21, 30], we propose a regularized bilateral branch network that consists of two branches: A "main" branch that trains on the original highly skewed long-tail distribution Ω_m; and a novel "regularizer" branch that trains on a relatively balanced data distribution Ω_r. Ω_m includes all the user feedback from both head and tail items. Ω_r, on the other hand, includes all the user feedback for the tail items, while down-sampling the user feedback for head items to be as frequent as the most popular tail items. A shared tower between two branches is used for scalability. The method can softly up-weight the learning of user preference towards tail items. So this corrects the under-estimation of user's preference towards tail items, and mitigates the popularity bias of user preference estimation.

In each step, a training example $(\mathbf{u}_m, \mathbf{i}_m) \in \Omega_m$ and $(\mathbf{u}_r, \mathbf{i}_r) \in \Omega_r$, is fed to the main branch and the regularizer branch respectively. Then the user representation vectors are calculated by:

$$\mathbf{x}_m = h_m(f(\mathbf{u}_m)), \quad \mathbf{x}_r = h_r(f(\mathbf{u}_r)). \tag{4}$$

where $f(\cdot)$ is a sub-network shared for both branches, $h_m(\cdot)$ and $h_r(\cdot)$ are the branch-specific sub-networks. The shared network helps to communicate the learned knowledge from both distributions, and largely reduces the computational complexity. The branch-specific subnetwork learns the unique knowledge for each data distribution (head and tail items). Therefore, the Ω_m and Ω_r are jointly learned to approximate the $\hat{p}(i)$ to $p(i)$, reduce the inconsistency of prior perspective.

The main branch aims to learn high-quality user representations that retain the characteristics of the original distribution, acting as the cornerstone to support the further learning in regularizer branch. As indicated by [14], training on original distribution can learn the best and most generalizable representations. The regularizer branch is designed to (1) add more tail information to the model and alleviate the high IF influence on tail items; (2) prevent over-fitting for tail items through a regularized adapter (the adapter details are discussed in Section 3.3).

In the application to production, the two branches can be trained simultaneously. So the extra training cost would be limited. Note that at inference time, only the main branch is used, so there would be no extra serving cost.

3.3 Cross Learning

To bridge the gap between head and tail items, we cross the learning of the decoupled information from user and item side through a γ-adapter.

The γ-adapter is designed to fuse the learned representations and softly shifts the learning focus towards tail items. Specifically, for $\mathbf{x}_m$ and $\mathbf{x}_r$ which are the learned user representations from main and regularizer branch, and $\mathbf{y}_m$ and $\mathbf{y}_r$ which are corresponding learned item representations. The predicted logit is formalized as:

$$s(i_m, i_r) = \alpha_t \mathbf{y}_m^T \mathbf{x}_m + (1 - \alpha_t)\mathbf{y}_r^T \mathbf{x}_r, \tag{5}$$

where α_t is the γ-adapter, and a function of the training epoch t:

$$\alpha_t = 1 - (\frac{t}{\gamma \times T})^2, \quad where\ \gamma > 1. \tag{6}$$

Here T is the total number of epochs, and γ is the regularizer rate. We see that α_t decays as the training progresses (as t increases), and this shifts the model's learning focus from the original distribution to the rebalanced data distribution. In this way, we learn the universal patterns first and then gradually shift towards tail items to improve their performance. This order is critical in obtaining a high-quality representation learning, which can further facilitate the learning of the regularizier branch, as indicated by [33]. The constrain $\gamma > 1$ is also important in the recommender setting to mitigate the forgetting issue: It ensures that the main focus always stays on the main branch throughout the training. This is a desirable feature for long-tail distributions with different imbalanced factor IF, when IF is high, a larger γ is preferred. In fact, we empirically found this γ-adapter significantly benefits the learning in the highly-skewed long-tail distribution.

With the logit $s(i_m, i_r)$, we calculate the probability of user u's preference towards different items through a softmax function:

$$p(i|u) = \frac{e^{s(i_m, i_r)}}{\sum_{j \in \mathcal{I}} e^{s(j_m, j_r)}}. \tag{7}$$

In the industry application, the batch softmax [28] is used for scalability. Then, the loss function can be formulated as:

$$L = -\sum_{u \in \mathcal{U}, i \in \mathcal{I}} \alpha_t \hat{d}(u_m, i_m) \log p(i|u) + (1 - \alpha_t)\hat{d}(u_r, i_r) \log p(i|u), \tag{8}$$

where $\hat{d}(u_m, i_m)$ and $\hat{d}(u_r, i_r)$, respectively, is the user feedback from the main and the regularizer branch. They help to learn high preference scores for items that user engaged with in the two branches.

For the inference, to predict a user's preference towards an item, we only use the main branch, and calculate the preference score as:

$$s(u, i) = \mathbf{y}_m^T \mathbf{x}_m. \tag{9}$$

to obtain the logits in softmax. The regularizer branch functions as a regularizer for the training. At prediction time, the test data is also long-tail, and adding the regularizer branch would introduce another layer of mismatch in distributions.

Training and serving cost: CDN has negligible extra cost for training, compared with the standard two tower model. At serving time, the user side only uses the main branch, and the item side has the same total number of parameters/ FLOPs as in the two tower setting. At training time, the extra cost on the user side is on the head for the regularizer branch, which is negligible.

Discussion: An intuitive question is why we decouple the user and item sides from different aspects? In this work, we consider the long-tail distributions from the item side (*i.e.* the long-tail item distribution), which treats users as samples in the long-tail distributions. If we want to consider the long-tail distribution from the user side, then a direct approach would be switching the decoupling methods between users and items as [30]. However, we argue that the long-tail user distributions should be modeled differently since the IF for the user side is usually substantially smaller than the item side. Additionally, the long-tail distributions from both sides are highly correlated and they can influence the IF in each side. This is a nontrivial problem and we leave it for future exploration.

4 EXPERIMENT

In this section, we conduct extensive experiments to address the following key questions:

RQ1: How well does the cross decoupling network CDN perform compared to the state-of-the-art methods, especially the traditional decoupling methods?

RQ2: How does the cross decoupling work in user and item sides?

RQ3: How would the expert design influence the CDN performance? How does the model balance between tail/head items by gating mechanisms?

RQ4: What's the influence of γ-adapter for recommendations on head and tail items?

RQ5: Are we learning better representations for tail items?

RQ6: How does the CDN work on real-world recommenders?

4.1 Experimental Setup

Datasets. We use both public benchmark datasets and production dataset at Google which contain rich user and item content information. The public datasets are MovieLens 1M [2] and BookCrossing [3]. The production dataset is from user logs in Google app ads pInstall. Items in three datasets follow the highly-skewed long-tail distribution, which suggests that they are well-suited for studying the targeted long-tail problem. For most experiments, we report the averaged metrics from 5 independent trials followed by the standard error of the mean after the ± sign.

Evaluation Criteria. We measure the model performance using the widely used metrics [25, 30] – Hit Ratio at top K (HR@K) and NDCG at top K (NDCG@K). HR@K measures the percentage of whether the ground-truth item is retrieved in the top K ranked items. NDCG@K further considers the ranked positions of the ground-truth items by assigning higher scores if ground-truth items are correctly recommended and have lower ranked positions.

We report both metrics evaluated on the tail, head slices and overall items, to provide different perspectives of recommendation performance. Similar as [30], based on the Pareto Principle [2, 20], we consider the top 20% most frequent items in MovieLens1M and 0.1% items in BookCrossing [4] as head items, and rest as tail items.

Our goal is to build a single model that can improve the recommendation performance on tail items, while maintaining or even improving the overall performance that can be easily applied for production usage with good scalability and efficiency.

Baselines. We use the widely adopted production model Two-tower model as the backbone model. Baselines are choosen based on the production usage and similar training/serving time.

- *Two-tower Model* [28]: The Two-tower model is a popular architecture for modeling different types of content information. Due to its scalability and efficiency, it is widely adopted in real-world applications [17, 27, 30].

Re-balance strategies:

- *ClassBalance* [6]: It calculates an effective number of samples for each imbalanced class and adopts it in a re-weighting term in loss function. We adapt the method to the recommendation task similar to [30].
- *LogQ* [18, 28]: The method is widely applied in production usage. It constructs an item frequency-related term to adjust the softmax logit in the loss function, in order to re-weight head and tail items differently in the learning process.

Decoupling:

- *NDP* [14]: The method is adapted from a recent state-of-the-art method for long-tail distribution problem in computer vision. Instead of jointly learning image representation and classification, it separately considers the two learning process in a two-stage training and shows great success in image classification. We adapt the method to recommendation, and treat users as representations and items as classes as [28].
- *BBN* [33]: BBN utilizes an uniform sampler and a reversed sampler to create a Bilateral-Branch Network (BBN), and jointly learns the two sampled distributions through an adaptor. In this work, BBN is applied to the image side to improve its representation learning. Hence, we follow a similar strategy where we treat the user as the representation learning and item as the classifier. For cumulative learning, $\alpha_t = 1 - (\frac{t}{T})^2$ is used.

4.2 Recommendation Performance (RQ1)

We first evaluate CDN and baselines on public datasets. All the methods have similar training time. Table 2 and Table 3 show the recommendation performance of CDN and baselines on the MovieLens1M and BookCrossing datasets.

The proposed CDN achieves the best performance – it brings significant improvements not only for tail items, but also for head items and overall recommendation performance. It is worth to notice that most of the baselines only improve on tail items, while the performance of head items significantly degrades (we find inverse propensity score IPS has similar performance as LogQ here). This shows that when we focus the learning attention on tail items, it is very challenging to maintain the performance for head items.

When comparing decoupling methods NDP, BBN and CDN, CDN outperforms the other methods, especially for the overall recommendation. In particular, NDP that adopts the two-stage training improves the recommendation for tail item; however the head item performance degrades. This verifies the forgetting issue of the two-stage training method in recommenders. CDN, on the other hand, is able to mitigate the forgetting issue through the cross decoupling technique, which results in an improvement of not only for tail but also for head items. It is also interesting to note that BBN failed to deliver an improvement either for the tail items or for the head items. One hypothesis is that BBN considers the branch that trains on the balanced data to be as important as the main branch, which does not work well in recommendation where the test distribution is also long tail. This demonstrates the importance to treat the regularizer branch (which takes a balanced distribution) as a regularizer, which softly introduces tail signals to the model while maintaining the main focus on head items.

[2] https://grouplens.org/datasets/movielens/1m/

[3] http://www2.informatik.uni-freiburg.de/~cziegler/BX/

[4] The split rate of 0.1% can better show the performance differences among methods, especially for tail items.

Table 2: The recommendation performance of CDN versus baselines on MovieLens1M. Numbers after ± indicate the standard error of the mean. The last row shows the improvement of CDN over the backbone Two-tower model.

Measure%	Overall		Head		Tail	
	HR@50	NDCG@50	HR@50	NDCG@50	HR@50	NDCG@50
Two-tower	24.68±0.16	7.29±0.06	33.83±0.39	10.06±0.14	10.32±0.50	2.96±0.14
ClassBalance	20.21±0.17	5.91±0.06	30.20±0.20	8.96±0.09	5.19±0.16	1.34±0.03
LogQ	13.37±0.23	3.50±0.07	10.75±0.38	2.62±0.09	17.30±0.15	4.81±0.05
NDP	14.72±0.14	4.28±0.06	14.92±0.22	4.20±0.07	14.41±0.22	4.41±0.07
BBN	16.18±0.38	4.84±0.17	22.55±0.91	6.89±0.36	6.19±0.82	1.61±0.25
CDN	**26.75±0.12**	**7.93±0.04**	**36.46±0.14**	**10.97±0.04**	**11.51±0.18**	**3.15±0.08**
Improv%	8.39%	8.78%	7.77%	9.05%	11.53%	6.42%

Table 3: The recommendation performance of CDN versus baselines on BookCrossing. Numbers after ± indicate the standard error of the mean. The last row shows the improvement of CDN over Two-tower model.

Measure%	Overall		Head		Tail	
	HR@50	NDCG@50	HR@50	NDCG@50	HR@50	NDCG@50
Two-tower	2.32±0.45	0.81±0.17	19.76±4.62	7.58±1.81	0.72±0.08	0.19±0.02
ClassBalance	1.47±0.32	0.47±0.10	8.94±2.46	3.36±0.88	0.79±0.13	0.21±0.04
LogQ	1.27±0.25	0.37±0.07	5.39±1.54	1.76±0.51	0.89±0.14	0.24±0.04
NDP	1.58±0.30	0.42±0.08	7.24±1.78	1.94±0.48	1.06±0.19	0.43±0.09
BBN	0.80±0.20	0.20±0.05	2.62±0.96	0.66±0.26	0.63±0.14	0.16±0.04
CDN	**2.52±0.3**	**0.82±0.15**	**21.18±4.82**	**7.42±1.86**	**0.81±0.05**	**0.22±0.01**
Improv%	8.94%	1.16%	7.21%	-2.09%	13.31%	12.90%

Moreover, CDN is more robust for both tail items and overall performance. The standard error of the mean in CDN is smaller than the Two-tower baseline. One possible reason is that CDN is jointly trained with both the original and the balanced dataset. This achieves a similar effect as the data augmentation technique, which effectively alleviates the over-fitting issues for tail items and results in lower model variances.

4.3 Cross-Decoupling (RQ2)

We further did ablation studies to explore how different design choices affect CDN performance:

- *BDN*: This model considers the main and regularizer branches on the user side to be equally important, as opposed to softly shifting the focus towards regularizer branch and treating it as a regularizer. The preference score is calculated by the averaged outputs of the two branches: $s(i_m, i_r) = 0.5\mathbf{y}_m^T\mathbf{x}_m + 0.5\mathbf{y}_r^T\mathbf{x}_r$.
- *UDN*: This model only decouples the user side by adopting the Bilateral-Branch Network. For the serving, same as CDN, only the main branch is used to calculate the user preference.
- *IDN*: This model only decouples the item side by utilizing the memorization-focused and generalization-focused experts with item frequency based gating. The serving is the same as CDN.

We see from Table 4 that CDN achieves the best performance for both tail items and head items. Concretely, UDN, IDN and CDN outperforms the two tower model and CDN has the largest improvements. It shows the benefits of cross decoupling. BDN outperforms the two-tower model on the tail items; however the head performance significantly degrades. This verifies the importance of considering the balanced branch as a regularizer rather than a component in the model for the heavily skewed long-tail distribution dataset in recommendation.

4.4 Comparison of Expert Design (RQ3)

Is the designed memorization-focused and generalization-focused experts in CDN can well handle the high heterogeneity between head and tail items in recommendation? To answer the question, we further compared our method with other expert design choices: separate the learning of unbalanced/balanced dataset (similar as the user tower), or head/tail items. We then analyze how head and tail items balance between memorization and generalization through experts' gate values.

4.4.1 Recommendation performance. To separately consider the head and tail items for item representation learning, there are some design choices:

- *Unbalanced/Balanced*: Similar as the user side, we use two experts: one trains on the original dataset, and the other trains on the balanced dataset. Item samples from the two distributions are separately fed to each expert for learning.
- *Head/Tail*: We directly let one expert trains over head items, and the other trains over tail items. Then, the two experts are aggregated through a 0-1 gating network [16].
- *Mem/Gen*: This is the design in CDN, where one expert considers the memorization features while the other considers the generalization features.

We only change the expert design while keeping the other settings the same for a fair comparison; for different expert designs, model sizes are kept the same. As shown in Table 5, CDN which

Table 4: Cross decoupling study on MovieLens1M. Numbers after ± indicate the standard error of the mean.

Measure%	Overall		Head		Tail	
	HR@50	NDCG@50	HR@50	NDCG@50	HR@50	NDCG@50
Two-tower	24.68±0.16	7.29±0.06	33.83±0.39	10.06±0.14	10.32±0.50	2.96±0.14
BDN	18.35±0.21	5.27±0.06	22.23±0.37	6.31±0.11	12.26± 0.17	3.64± 0.04
UDN	25.55±0.08	7.53±0.03	34.98±0.11	10.35±0.06	10.77±0.13	3.10±0.03
IDN	25.85±0.15	7.58±0.05	34.82±0.21	10.26±0.08	11.79±0.22	3.37±0.08
CDN	26.75±0.12	7.93±0.04	36.46±0.14	10.97±0.04	11.51±0.18	3.15±0.08

adopts Mem/Gen experts achieves the best performance. Specifically, when compared with UDN (in Table 4) which only decouples the user side, our method that further decouples the item size through Mem/Gen experts shows improvements for both head and tail items; while alternative designs using Unbalanced/Balanced and Head/Tail experts yield a neutral performance for head items. It is worth to note that compared with UDN, the tail performance on Head/Tail relatively degrades. One possible reason is that training tail items alone easily causes over-fitting. This demonstrates the advantages to separately consider the memorization and generalization for item representation learning.

4.4.2 Memorization vs Generalization. We then checked how CDN balances memorization and generalization between head and tail items through the learned gate values in CDN. The results are very inspiring and shows many insights.

Concretely, Figure 3 shows the gate values of head items (blue bars) and tail items (orange bars) on memorization-focused expert (left) and generalization-focused expert (right). Comparing between values learned from generalization-focused expert (right) and memorization focused expert (left), we see both head and tail items put a higher weight on the generalization-focused expert. It is reasonable since both datasets are very sparse and we utilize rich content information which are better at generalization. More importantly, when comparing between head and tail items, we see that head items put more weights on memorization-focused experts while tail items put more weights on generalization-focused experts. This is consistent with our assumption that head items prefer more memorization abilities as their user feedback information is already very rich, while tail items prefer more generalization abilities due to training data sparsity. This healthy balance of memorization and generalization in CDN is important in improving the performance of both head and tail items.

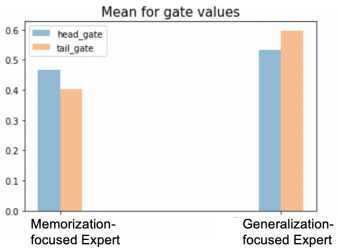

Figure 3: Gate value of memorization-focused and generalization-focused experts for head (blue) and tail (orange) items. In CDN, head items put more weights on memorization-focused experts while tail items put more weights on generalization-focused experts

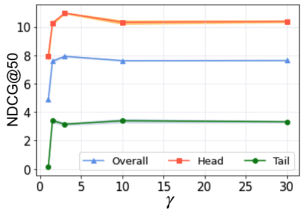

Figure 4: The NDCG@50 of head, tail and overall performance w.r.t different γ.

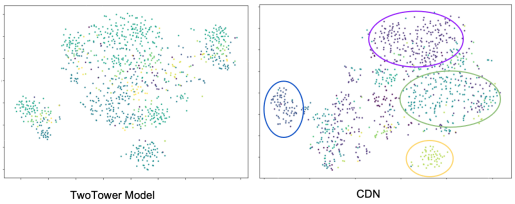

Figure 5: Embedding visualization comparison by using t-SNE [23]. Each genre is represented by a different color.

4.5 γ-Adapter (RQ4)

The γ-adapter controls the decay speed of the adaptor function as well as its lower bound. In this section, we check how it influence the recommendation performance on head and tail items.

Figure 4 shows how NDCG@50 changes as we increase γ (similar for HR). When γ becomes larger, it puts more weights on the main branch in most epochs. We see in the region where γ is small (*e.g.*, $\gamma < 5$ in x-axis), increasing γ yields a significant improvement. This suggests the importance of maintaining a focus on the main branch. That is, treating the regularizer branch as a regularization term. When γ is large (*e.g.*, $\gamma > 5$), we see that the performance saturated when we further increase γ. Furthermore, for tail items (green line), larger γ could hurt its performance (e.g. when $\gamma = 4$). This shows the regularizer branch benefits tail item to avoid overfitting.

4.6 Representation Visualization (RQ5)

We visualize the learned embeddings from CDN and two-tower model to analyze semantics of the learnt representations, as shown in Figure 5. We see that compared to the two-tower model, CDN's embeddings are more coherently clustered (movies in the same genre tend to cluster together). This suggests that CDN is able to learn more semantic meaningful representations for tail items, which also aligns with the findings in Section 4.4.2 where tail slice items put more weights on the generalization experts.

Table 5: Expert design study on MovieLens1M. Numbers after ± indicate the standard error of the mean.

Measure%	Overall		Head		Tail	
	HR@50	NDCG@50	HR@50	NDCG@50	HR@50	NDCG@50
Unbalanced/Balanced	25.89±0.08	7.59±0.02	34.69±0.23	10.16±0.06	12.09±0.32	3.57±0.12
Head/Tail	25.25±0.19	7.48±0.06	34.64±0.34	10.36±0.11	10.52±0.41	2.97±0.14
Mem/Gen	26.75±0.12	7.93±0.04	36.46±0.14	10.97±0.04	11.51±0.18	3.15±0.08

Table 6: Relative improvements of CDN performance.

Metrics	Overall	Dense	Sparse
(Offline) AUCLoss improvement	+0.27%	+0.26%	+0.66%
(Online) KPI online	+1.43%	+0.73%	+3.75%

4.7 Experiments in a Google System (RQ6)

We conduct a study to showcase CDN framework's performance in a real-world recommender system at Google. Results show that CDN is easy to adapt with industry scale and constraints, and delivers significant offline and online performance through live A/B experiments.

Concretely, given queries and candidates, the task is to predict the likelihood of a user installing an app. The training data is sampled from user logs and consists of hundreds of billions of training examples. Rich content features of query and candidates are used as inputs in the model. We use a binary label of whether a user installed the app, and the positive labels are sparse compared to the negatives. Due to the constraints on the infrastructure side, here we focus on the item side decoupling. And instead of focusing on head and tail items, we adopted different approaches to define dense and sparse data slices based on the production application. Results are shown in table 6.

As shown in the table 6, CDN delivers significant improvements on top of the production model for both offline and online experiments. We use AUCLoss (1 - AUC) as the offline evaluation metric, and see 0.27% overall improvements, 0.66% improvement on the sparse slice, and 0.26% improvement on the dense slice (0.1% is considered significant). In a time frame of 14 days, the online metrics were also highly positive, with 1.43% improvements on key online metric. What is encouraging is that we see even bigger improvements for sparse slice compared to the dense slice, with 3.75% improvements on key online metric. The results further confirm the effectiveness of CDN to improve the sparse slice performance and model generalization.

5 RELATED WORK

Long-tail Problem. Large-scale real world datasets often exhibit a long-tail distribution, that pose critical challenges to many industry recommender systems [11, 19]. Industry solutions usually focus on rebalancing the data distribution through resampling and reweighting [6, 18, 28]. While improving performance for tail classes, they may under-represent the head classes. In fact, recent studies [14, 33] find that the rebalancing methods could hurt the representative capability when the original distribution gets distorted badly.

Many of the progresses mentioned above are achieved in academia [4, 7], especially computer vision area [14, 26, 31]. However, directly adapting those methods to industry-scale recommenders is very challenging. For example, a closely related academia work [30] uses meta-learning to address the long-tail distribution problem. It shows great success in the academia datasets. However, we find the training time of the method to be doubled compared to the production baseline. The increased training time cannot justify the performance gain, making it infeasible in practice. Furthermore, in recommendation, the imbalance factor IF is magnitudes higher, and IF is a core factor that influence the recommendation performance. Moreover, to improve the performance of tail items, most related work in recommendation are focusing on cold-start items by investigating effective ways to incorporate content information [15, 34]. This is a different scenario from the long-tail distribution problem. In our long-tail case, we need to consider the influence from the imbalanced distribution and aim to not only improve the tail performance but also maintain the overall performance.

Memorization and Generalization. This is a key topic that has been widely studied [1, 3, 29]. They directly influence the model's expressiveness and performance on unseen data or new tasks. [9] theoretically shows that memorizing data labels is necessary to achieve a good performance, and many experiments [10] further strengthen the importance of memorization. Recently, generalization receives extensive attention for different model designs, such as meta-learning [8], reinforcement learning [24]. Research has shown that generalization is especially crucial when the data is sparse or insufficient for a given task, *e.g.*, the distribution shift and adversarial attack [35] problems. [29] conducted empirical analysis on the generalization for different models, and shows that a model's generalization ability is closely related to its architecture. In recommendation, the wide-and-deep model [5] explains the importance of considering both memorization and generalization in model design, and is widely adopted in many real world applications.

6 CONCLUSION

In this work, we propose a scalable cross decoupling network (CDN) to address the long-tail distribution in industry recommender systems. CDN decouples the memorization and generalization on the item side, as well as the user sampling from the user side. Then, a new γ-adapter is introduced to softly shift the learning attention to tail items, while smoothly aggregating the learning from both sides. Experiments on public dataset show that CDN delivers significant improvements for both tail and overall recommendations. The case study of applying CDN to a large-scale recommendation system at Google further demonstrates CDN can also improve overall model performance and model generalization in real world scenarios.

ACKNOWLEDGMENTS

We would like thank our amazing collaborators for their collaboration and highly valuable suggestions, including Ting Chen, Evan Ettinger, Andrew Evdokimov, and Samuel Ieong.

REFERENCES

[1] Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. 2017. A closer look at memorization in deep networks. In *ICML*. PMLR, 233–242.

[2] George EP Box and R Daniel Meyer. 1986. An analysis for unreplicated fractional factorials. *Technometrics* 28, 1 (1986).

[3] Gavin Brown, Mark Bun, Vitaly Feldman, Adam Smith, and Kunal Talwar. 2021. When is memorization of irrelevant training data necessary for high-accuracy learning?. In *Proceedings of the 53rd annual ACM SIGACT symposium on theory of computing*. 123–132.

[4] Yixin Cao, Jun Kuang, Ming Gao, Aoying Zhou, Yonggang Wen, and Tat-Seng Chua. 2021. Learning relation prototype from unlabeled texts for long-tail relation extraction. *IEEE Transactions on Knowledge and Data Engineering* (2021).

[5] 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.

[6] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. 2019. Class-balanced loss based on effective number of samples. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 9268–9277.

[7] Kunal Dahiya, Anshul Mittal, Deepak Saini, Kushal Dave, Himanshu Jain, Sumeet Agarwal, and Manik Varma. 2019. DeepXML: Scalable & Accurate Deep Extreme Classification for Matching User Queries to Advertiser Bid Phrases. (2019).

[8] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. 2021. Generalization of Model-Agnostic Meta-Learning Algorithms: Recurring and Unseen Tasks. In *NeurIPS*.

[9] Vitaly Feldman. 2020. Does learning require memorization? a short tale about a long tail. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. 954–959.

[10] Vitaly Feldman and Chiyuan Zhang. 2020. What neural networks memorize and why: Discovering the long tail via influence estimation. *arXiv preprint arXiv:2008.03703* (2020).

[11] Yingqiang Ge, Xiaoting Zhao, Lucia Yu, Saurabh Paul, Diane Hu, Chu-Cheng Hsieh, and Yongfeng Zhang. 2022. Toward Pareto efficient fairness-utility trade-off in recommendation through reinforcement learning. In *Proceedings of the fifteenth ACM international conference on web search and data mining*. 316–324.

[12] 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.

[13] Muhammad Abdullah Jamal, Matthew Brown, Ming-Hsuan Yang, Liqiang Wang, and Boqing Gong. 2020. Rethinking class-balanced methods for long-tailed visual recognition from a domain adaptation perspective. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7610–7619.

[14] Bingyi Kang, Saining Xie, Marcus Rohrbach, Zhicheng Yan, Albert Gordo, Jiashi Feng, and Yannis Kalantidis. 2020. Decoupling representation and classifier for long-tailed recognition. *ICLR* (2020).

[15] Yuanfu Lu, Yuan Fang, and Chuan Shi. 2020. Meta-learning on heterogeneous information networks for cold-start recommendation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1563–1573.

[16] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. 2018. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 1930–1939.

[17] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Ji Yang, Minmin Chen, Jiaxi Tang, Lichan Hong, and Ed H Chi. 2020. Off-policy learning in two-stage recommender systems. In *Proceedings of The Web Conference 2020*. 463–473.

[18] Aditya Krishna Menon, Sadeep Jayasumana, Ankit Singh Rawat, Himanshu Jain, Andreas Veit, and Sanjiv Kumar. 2021. Long-tail learning via logit adjustment. In *ICLR 2021*.

[19] Xichuan Niu, Bofang Li, Chenliang Li, Rong Xiao, Haochuan Sun, Hongbo Deng, and Zhenzhong Chen. 2020. A dual heterogeneous graph attention network to improve long-tail performance for shop search in e-commerce. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3405–3415.

[20] William J Reed. 2001. The Pareto, Zipf and other power laws. *Economics letters* 74, 1 (2001), 15–19.

[21] Jiawei Ren, Cunjun Yu, Shunan Sheng, Xiao Ma, Haiyu Zhao, Shuai Yi, and Hongsheng Li. 2020. Balanced meta-softmax for long-tailed visual recognition. *arXiv preprint arXiv:2007.10740* (2020).

[22] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. 2018. An empirical study of example forgetting during deep neural network learning. *arXiv preprint arXiv:1812.05159* (2018).

[23] Laurens Van Der Maaten. 2014. Accelerating t-SNE using tree-based algorithms. *The Journal of Machine Learning Research* 15, 1 (2014), 3221–3245.

[24] Kaixin Wang, Bingyi Kang, Jie Shao, and Jiashi Feng. 2020. Improving generalization in reinforcement learning with mixture regularization. In *NeurIPS*.

[25] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *SIGIR*. 165–174.

[26] Xudong Wang, Long Lian, Zhongqi Miao, Ziwei Liu, and Stella X Yu. 2020. Long-tailed recognition by routing diverse distribution-aware experts. *arXiv preprint arXiv:2010.01809* (2020).

[27] Tiansheng Yao, Xinyang Yi, Derek Zhiyuan Cheng, Felix Yu, Ting Chen, Aditya Menon, Lichan Hong, Ed H Chi, Steve Tjoa, Jieqi Kang, et al. 2021. Self-supervised learning for large-scale item recommendations. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 4321–4330.

[28] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed Chi. 2019. Sampling-bias-corrected neural modeling for large corpus item recommendations. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 269–277.

[29] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Michael C Mozer, and Yoram Singer. 2020. Identity crisis: Memorization and generalization under extreme overparameterization. In *ICLR*.

[30] Yin Zhang, Derek Zhiyuan Cheng, Tiansheng Yao, Xinyang Yi, Lichan Hong, and Ed H Chi. 2021. A model of two tales: Dual transfer learning framework for improved long-tail item recommendation. In *Proceedings of the web conference 2021*. 2220–2231.

[31] Yifan Zhang, Bryan Hooi, Lanqing Hong, and Jiashi Feng. 2021. Test-agnostic long-tailed recognition by test-time aggregating diverse experts with self-supervision. *arXiv e-prints* (2021), arXiv–2107.

[32] Yu Zheng, Chen Gao, Xiang Li, Xiangnan He, Yong Li, and Depeng Jin. 2021. Disentangling user interest and conformity for recommendation with causal embedding. In *Proceedings of the Web Conference 2021*. 2980–2991.

[33] Boyan Zhou, Quan Cui, Xiu-Shen Wei, and Zhao-Min Chen. 2020. Bbn: Bilateral-branch network with cumulative learning for long-tailed visual recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 9719–9728.

[34] Ke Zhou, Shuang-Hong Yang, and Hongyuan Zha. 2011. Functional matrix factorizations for cold-start recommendation. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. 315–324.

[35] Sicheng Zhu, Bang An, and Furong Huang. 2021. Understanding the Generalization Benefit of Model Invariance from a Data Perspective. In *NeurIPS*.

Table 7: Statistics and used content information in the datasets.

	MovieLens1M	BookCrossing
# User	6,040	50,454
# Items	3,706	222,154
# Feedback	1,000,209	1,031,175
Sparsity	95.5316%	99.9908%
IF	3,428	2,001
User Features	IDs, Gender, Occupation, ZipCode, Age	IDs, Location, Age
Item Features	IDs, Title, Genres, Year	IDs, Title, Author, Year, Publisher

A PROOF OF EQUATION (2) IN SECTION 2

Proof. By Bayes' theorem, the user preference towards items can be represented by

$$p(i|u) = \frac{p(u|i)}{p(u)} p(i),$$

then, based on the observed data, the posterior probability is

$$\hat{p}(i|u) = \frac{\hat{p}(u|i)}{\hat{p}(u)} \hat{p}(i),$$

where $\hat{p}(i)$ is usually estimated based on the observed data distribution.

Therefore, the differences between the desired conditional probability and the estimated one is:

$$log\frac{p(i|u)}{\hat{p}(i|u)} = \log \frac{p(i)}{\hat{p}(i)} + \log \frac{p(u|i)}{\hat{p}(u|i)} + \log \frac{\hat{p}(u)}{p(u)}. \tag{10}$$

Suppose $s(u,i) = \log(\frac{p(i|u)}{p(1|u)})$ by using canonical link function [21], where the $p(1|u)$ is when the item equals to 1. then

$$s(u,i) - \log \frac{p(i|u)}{\hat{p}(i|u)} = \log \frac{p(i|u)}{p(1|u)} - \log \frac{p(i|u)}{\hat{p}(i|u)} = \log \frac{\hat{p}(i|u)}{p(1|u)}.$$

Raising both terms to the exponent gives:

$$\hat{p}(i|u) = e^{s(u,i) - \log \frac{p(i|u)}{\hat{p}(i|u)}} p(1|u). \tag{11}$$

Then, summing up the probabilities over all items yields:

$$1 = \sum_{i \in \mathcal{I}} \hat{p}(i|u) = \sum_{i \in \mathcal{I}} e^{s(u,i) - \log \frac{p(i|u)}{\hat{p}(i|u)}} p(1|u). \tag{12}$$

Hence, combining Eq. (11) and Eq. (12), we have

$$\hat{p}(i|u) = \frac{e^{s(u,i) - \log \frac{p(i|u)}{\hat{p}(i|u)}}}{\sum_{j \in \mathcal{I}} e^{s(u,j) - \log \frac{p(j|u)}{\hat{p}(j|u)}}}.$$

Incorporating with Eq. (10), we obtain

$$\hat{p}(i|u) = \frac{e^{s(u,i) - \log \frac{p(i)}{\hat{p}(i)} - \log \frac{p(u|i)}{\hat{p}(u|i)}}}{\sum_{j \in \mathcal{I}} e^{s(u,j) - \log \frac{p(j)}{\hat{p}(j)} - \log \frac{p(u|j)}{\hat{p}(u|j)}}}$$

Hence the proof. □

B EXPERIMENTAL SETUP

B.1 Experimental Setup for Figure 1

We experimentally study the forgetting issue in this section. The experiment setting follows the two-stage training setting in [14], where in each stage the original long-tail data distribution Ω_m and resampled balanced data distribution Ω_r are used. We denote the original long-tail distribution as H (which has more head items), and the resampled balanced data distribution [30] as T (which has more tail items). Then, $H \rightarrow T$ means that the model trains on the long-tail distribution in the first stage, and on a balanced data distribution in the second stage. For simplicity, we only use the ID feature in the item side.

Figure 1 shows the recommendation performance on MovienLens1M dataset. x-axis and y-axis represents the hit ratio at top k (HR@K) for tail items and overall items respectively. The metrics are averaged over 5 independent runs. Consider $H \rightarrow H$ and $H \rightarrow T$ (or $T \rightarrow H$ and $T \rightarrow T$), we see that when a method shifts its focus to tail items, *i.e.*, switching to balanced data distribution in the second stage, its tail performance improves while the overall performance degrades significantly. This suggests that when the learning attention is shifted towards tail items, the already learned knowledge of head items is forgotten. Further, consider $H \rightarrow H$ and $T \rightarrow H$ (or $H \rightarrow T$ and $T \rightarrow T$), we can also see that in the first stage, training on the original long-tail distribution yields a better performance than training on a balanced distribution. This finding is consistent with [14], and suggests that training on the original long-tail distribution results in a better user representation.

The results have shed light on the importance of mitigating the forgetting issue to obtain a good performance of both tail items overall recommendation.

B.2 Offline Dataset

Both MovieLens1M and BookCrossing datasets are widely used benchmark for recommendation with considering content information [15]. For MovieLens1M and BookCrossing datasets, the <user, item> pairs with explicit ratings are considered as positive examples (1s), and the rest of the pairs are negative examples (0s). Pairs with invalid or missing features are filtered. For feature engineering, we follow [30] for consistency. The dataset statistics are shown in Table 7.

The MovieLens1M is splitted to train/validation/test datasets by the leave-one-out evaluation [12]: for each user, the most recent interacted item is for testing, the second most recent interacted item is for validation, and the rest items are used for training. Since the BookCrossing dataset does not have the timestamp information, for each user, we randomly select one item for testing, one for validation, and use the rest of items for training. Statistical details for the two datasets are shown in Table 7.

B.3 Hyper-parameter Settings

All the methods are implemented by Tensorflow. The user and item embedding dimensions for all methods are set to be 64. For methods that utilize the two-tower architecture [28], their hidden layer sizes for both towers are set to [256, 128, 64] for fair comparisons. ReLU is used as the activation function. Adam is used for the optimizer. To reduce the variance, all the results are the average of the 5 experiments. Readers may replicate the results reported in the paper using the random seed of 0,1,2,3,4 to get the average and

Algorithm 1: Training procedure of CDN.

Input : user set $\mathcal{U}$, item set $\mathcal{I}$, feature vector $\mathbf{u}$, $\mathbf{i}$ that maps each user and item input to an embedding space, regularizer rate γ, epoch number T

Initialized model parameters Θ;
Construct sets Ω_m, Ω_r;
for $t = 1$ *to* T **do**
  Calculate α_t based on Equation (6);
  Generate a training example $(u_m, i_m) \in \Omega_m$, and a training example $(u_r, i_r) \in \Omega_r$;
  Decouple the memorization and generalization features for i_m and i_r;
  Calculate item representations $\mathbf{y}_m$, $\mathbf{y}_r$ with the decoupled features for i_m and i_r based on Equation (3);
  Calculate user representations $\mathbf{x}_m$, $\mathbf{x}_r$ for u_m and u_r based on Equation (4);
  $s(i_m, i_r) \leftarrow \alpha_b \mathbf{y}_m^T \mathbf{x}_m + (1 - \alpha_b)\mathbf{y}_r^T \mathbf{x}_r$;
  $p \leftarrow softmax(s(i_m, i_r))$;
  Update the loss function based on Equation (8);
  Update the model parameters;

standard error of the mean. The learning rate, dropout rate and regularization parameters are selected by a grid search in the range of {0.1, 0.01, 0.001, 0.0001}, {0.1, 0.3, 0.5, 0.7, 0.9} and {0.0, 0.1, 0.01, 0.001}, respectively.

For methods with experts, each expert is set as a single-layer network for simplicity with a hidden layer size 64. The number of experts that encode memorization and generalization features is determined by grid search from {1,2,3}. The regularizer rate γ in the adapter is searched from {1.2, 1.5, 2.1, 3, 5, 10, 30}. For logQ, the log weight is searched from {0.1, 1, 10}. For methods that adopt a 2-stage training, the number of training epochs are set to be 20 for each stage. For the other methods, the number of epochs are 40 for a fair comparison.