SelfCF: A Simple Framework for Self-supervised Collaborative Filtering

XIN ZHOU, Nanyang Technological University, Singapore
AIXIN SUN, Nanyang Technological University, Singapore
YONG LIU, Nanyang Technological University, Singapore
JIE ZHANG, Nanyang Technological University, Singapore
CHUNYAN MIAO, Nanyang Technological University, Singapore

Collaborative filtering (CF) is widely used to learn informative latent representations of users and items from observed interactions. Existing CF-based methods commonly adopt negative sampling to discriminate different items. That is, observed user-item pairs are treated as positive instances; unobserved pairs are considered as negative instances and are sampled under a defined distribution for training. Training with negative sampling on large datasets is computationally expensive. Further, negative items should be carefully sampled under the defined distribution, in order to avoid selecting an observed positive item in the training dataset. Unavoidably, some negative items sampled from the training dataset could be positive in the test set. Recently, self-supervised learning (SSL) has emerged as a powerful tool to learn a model without negative samples. In this paper, we propose a self-supervised collaborative filtering framework (SelfCF), that is specially designed for recommender scenario with implicit feedback. The proposed SelfCF framework simplifies Siamese networks and can be easily applied to existing deep-learning based CF models, which we refer to as backbone networks. The main idea of SelfCF is to augment the latent embeddings generated by backbone networks instead of the raw input of user/item ids. We propose and study three embedding perturbation techniques that can be applied to different types of backbone networks including both traditional CF models and graph-based models. The framework enables learning informative representations of users and items without negative samples, and is agnostic to the encapsulated backbones. We conduct experimental comparisons on four datasets, one self-supervised framework and eight baselines to show that our framework may achieve even better recommendation accuracy than the encapsulated supervised counterpart with a 2×–4× faster training speed. The results also demonstrate that SelfCF can boost up the accuracy of a self-supervised framework BUIR by 17.79% on average and shows competitive performance with baselines.

CCS Concepts: • Information systems → Recommender systems.

Additional Key Words and Phrases: Collaborative Filtering, Self-supervised Learning, Recommender Systems, Siamese Networks

1 INTRODUCTION

Recommender systems aim to provide users with personalized products or services. They help to handle the increasing information overload problem and improve customer relationship management. In Fig. 1, we present an illustration of recommendation under implicit feedback. Recommender systems are designed to infer the missing values of the matrix (right) transformed from the user-item interactions (left). In top-$K$ scenario, the inferred values are further ranked with each user for personalized recommendation. Collaborative Filtering (CF)
Fig. 1. An illustration of the recommendation scenario under implicit feedback. Only positive samples can be captured for training recommender systems.

is a canonical recommendation technique, which predicts interests of a user by aggregating information from similar users or items. In detail, existing CF-based methods \cite{20, 21, 26, 38} learn latent representations of users and items, by first factorizing the observed interaction matrix, then predicting the potential interests of user-item pairs based on the dot-product of learned embeddings. However, existing CF models rely heavily on negative sampling techniques to discriminate against different items, because negative samples are not naturally available.

Nevertheless, the negative sampling techniques suffer from a few limitations. Firstly, they introduce additional computation and memory costs. In existing CF-based methods, the negative sampling algorithm need be carefully designed in order to not select the observed positive user-item pairs. Specifically, to sample one negative user-item pair for a specific user, the algorithm checks its conflicts with all the observed positive items interacted with this user. As a result, much computation is needed for users who have a large number of interactions. Secondly, even if non-conflicted negative samples are selected for a user, the samples may fall into future positive items of the user. The reason is that the unobserved user-item pairs can be either true negative instances (i.e., the user is not interested in these items) or missing values (e.g., interaction pairs not observed in the training set but the test set) \cite{28, 36}. We denote the sampled pairs that fall in the test set as false negative samples \cite{6}. Although another line of work \cite{7–9} has get rid of negative sampling and takes the full unobserved interactions as negative samples, they may still treat a future positive sample as negative.

To uncover the negative sampling problem in current models, we employ uniform sampling (UniS) and Dynamic Negative Sampling (DNS) \cite{57} in LightGCN \cite{20} to study the aforementioned issues. Uniform sampling is a widely used and classical solution in the item recommendation domain with implicit feedback \cite{6}. DNS improves uniform sampling by selecting a set of negative candidates and ranking the candidates based on learned user/item embeddings. The top-ranked item is used as a hard instance. As a result, DNS is model-sensitive. We plot the percentage of sampled false negative pairs in the test set along with the training progress of LightGCN under uniform sampling and DNS in Fig. 2. We test the negative sampling methods on two diverse datasets, MOOC and Amazon Video Games (Games). MOOC contains 458,453 interactions collected from 82,535 learners on 1,302 courses. Games has 50,677 users, 16,897 items and 454,529 interactions. The sparsity of MOOC and Games are 99.4039% and 99.9469%, respectively. From Fig. 2, we observe the percentage of false negative pairs sampled by DNS is over 50% when LightGCN early stops on the MOOC dataset. Here, we use the original early
SelfCF: A Simple Framework for Self-supervised Collaborative Filtering

Stopping setting in LightGCN[20]. The sparse dataset, Games, has a relatively small number of sampled false negative instances under 10%. However, the overhead to sample a negative instance increases with the number of candidates, as shown in Table 1. Although DNS can sample hard negative instances, its overhead on sampling is 2-3 times of uniform sampling in Table 1. From the above observations, it is promising to train the model without negative sampling.

Self-supervised learning (SSL) models [13, 17, 51] provide us a possible solution to tackle the aforementioned limitations. SSL enables training a model by iteratively updating network parameters without using negative samples. Research in various domains ranging from Computer Vision (CV) to Natural Language Processing (NLP), has shown that SSL is possible to achieve competitive or even better results than supervised learning [12, 17, 51]. The underlying idea is to maximize the similarity of representations obtained from different distorted versions of a sample using a variant of Siamese networks [18]. Siamese networks usually include two symmetric networks (i.e., online network and target network) for inputs comparing. The problem with only positive sampling in model training is that, the Siamese networks collapse to a trivial constant solution [13]. Thus, in recent work, BYOL [17] and SimSiam [13] introduce asymmetry to the network architecture by adding parameter update technique. Specifically, in the network architecture, an additional “predictor” network is stacked onto the online encoder. For parameter update, a special “stop gradient” operation is highlighted to prevent solution collapsing. SimSiam simplifies BYOL by removing its “momentum update”, which updates the parameters of target networks based on online networks. We will illustrate the architectures in detail in the related work section.

To the best of our knowledge, BUIR [28] is the only recommendation framework to learn user and item latent representations without negative samples. BUIR is derived from BYOL [17]. Similar to BYOL, BUIR employs two distinct encoder networks (i.e., online and target networks) to address the recurring of trivial constant solutions in SSL. In BUIR, the parameters of the online network are optimized towards that of the target network. At the same time, parameters of the target network are updated based on momentum-based moving average [17, 19, 42] to slowly approximate the online network [28]. As BUIR is built upon BYOL, which stems from vision domain, its architecture is redundant and suffers from slow convergence, because of the design of the momentum-based parameter updating. The SimSiam network is originally proposed in vision domain as well. The input is an image, and techniques for data augmentation on images are relatively mature [39], such as random cropping, resizing,

Fig. 2. Percentage of false negative pairs sampled with different sampling methods of LightGCN. The percentage is calculated by dividing the number of sampled false negative pairs by the number of instances in the test set and multiplying the result by 100.

Table 1. Negative sampling time under various sampling methods and datasets. Overhead is calculated as the percentage of sampling time over training time per epoch. UniS denotes uniform sampling. DNS denotes dynamic negative sampling.

| Dataset | Method | Time (s) Sampling | Time (s) Training | Overhead |
|---------|--------|-------------------|------------------|----------|
| MOOC    | UniS   | 2.32              | 8.89             | 26.1%    |
| MOOC    | DNS    | 32.38             | 40.33            | 80.3%    |
| Games   | UniS   | 3.47              | 8.22             | 42.2%    |
| Games   | DNS    | 39.15             | 43.68            | 89.6%    |

ACM Trans. Recomm. Syst.
horizontal flipping, color jittering, converting to grayscale, Gaussian blurring, and solarization. As for a pair of user id and item id that is observed in implicit feedback, there is no standard solution on how to distort it while keeping its representation invariant.

The learning paradigm of SSL without negative samples differs slightly from existing paradigms that use negative samples to learn representations. SSL without negative samples intends to learn an encoder with augmentation-invariant representation [13, 17]. That is, they minimize the representation distance between two positive samples based on a Siamese network architecture [3]. By using negative samples in SSL, solutions are prevented from collapsing because of their repulsivity. Our proposed framework can achieve competitive performance without harnessing this repulsive force.

In this paper, we propose a Self-supervised Collaborative Filtering (SelfCF) framework, which performs posterior perturbation on user and item latent embeddings, to obtain a contrastive pair. On architecture design, our framework uses only one encoder instead of two encoders, which simplifies BYOL and SimSiam. Besides, instead of perturbing inputs ahead of encoding, we generate different but invariant contrastive views with posterior embedding perturbations. An additional benefit of posterior embedding perturbation is that the framework can take the internal implementation of the encapsulated backbones as black-box. Conversely, BUIR adds momentum-based parameter updating to encoders in order to generate different views. Our experiments on four real-world datasets validate that the proposed SelfCF framework is able to learn informative representation solely based on positive user-item pairs. In our experiments, we encapsulate two popular CF-based models into the framework, and the results on Top-K item recommendation are competitive or even better than their supervised counterparts.

We summarize our contributions as follows:

- We propose a novel framework, SelfCF, that learns latent representations of users/items solely based on positively observed interactions. The framework uses posterior output perturbation to generate different augmented views of the same user/item embeddings for contrastive learning.
- We design three output perturbation techniques: historical embedding, embedding dropout, and edge pruning to distort the output of the backbone. The techniques are applicable to all existing CF-based models as long as their outputs are embedding-like.
- We investigate the underlying mechanisms of the framework by performing ablation study on each component. We find the presentations of user/item can be learnt even without the “stop gradient” operator, which shows different behaviors from previous SSL frameworks (e.g., BYOL [17] and SimSiam [13]).
- Finally, we conduct experiments on four public datasets by encapsulating two popular backbones. Results show SelfCF is competitive or better than their supervised counterpart and outperforms existing SSL framework by up to 17.79% on average.

2 RELATED WORK

In this section, we first review the CF technique, then summarize the current progress of SSL.

2.1 Collaborative Filtering

CF is a typical and prevalent technique adopted in modern recommender systems [48]. The core concept is that similar users tend to have similar tastes on items. To tackle the data sparsity and scalability of CF, more advanced method, Matrix Factorization (MF), decomposes the original sparse matrix to low-dimensional matrices with latent factors/features and less sparsity. To learn informative and compressed latent features, deep learning based models are further proposed for recommendation [21, 43, 56].

With the emerge of graph convolutional networks (GCNs), which generalize convolutional neural networks (CNNs) on graph-structured data [31, 54, 63], GCN-based CF is widely researched recently [2, 45, 48, 60–62]. The user-item interaction matrix naturally can be treated as a bipartite graph. GCN-based CF takes advantage of fusing
both high-order information and the inherent graph structure. GCNs are used to propagate information using the normalized adjacency matrix and aggregate information from neighbors via the nonlinear activation and linear transformation layers. He et al. [20] simplify the GCNs architecture by removing the feature transformation as well as nonlinear activation layers as they impose negative effect on recommendation performance. In [11], the authors add a residual preference learning on GCN and obtain better recommendation performance.

2.2 Self-supervised Learning

SSL has achieved competitive results on various tasks in vision and natural language processing domains. We review two lines of work on SSL.

**Contrastive learning.** Contrastive approaches learn representations by attracting the positive sample pairs and repulsing the negative sample pairs [18]. A line of work [12, 19, 22, 23, 47, 53, 55] is developed based on this concept. These work benefits from a large number of negative samples, which require a memory bank [47] or a queue [19] to store negative samples. In [46], the authors integrate supplemental signal into supervised baselines for contrastive learning, and show that it performs better than their baselines. Following the line, Yu et al. propose a graph-augmentation free recommendation model [49] to enforce the learning of uniform representations for users and items. The uniform representations can mitigate the popularity bias and achieve better recommendation accuracy. Liu et al. summarize the contrastive learning applied on a broad field, e.g., NLP, Computer Vision, in [32].

**Siamese networks.** Siamese networks [3] are general models for comparing entities. BYOL [17] and SimSiam [13] are two specializations of the Siamese network that achieve remarkable results by only using positive samples. BYOL proposes two coupled networks (i.e., online and target networks) that are optimized and updated iteratively. In detail, the online network is optimized towards the target network, while the target network is updated with a moving average of the online network to avoid collapse. On the contrary, SimSiam verifies that a “stop gradient” operator is crucial in preventing collapse. As a result, it removes the dashed “momentum update” line in Fig. 3a.

Derived from BYOL, the recently proposed self-supervised framework, BUIR, learns the representation of users and items solely on positive interactions. It introduces different views by differentiating parameters of online and target networks. However, the framework modifies the underlying logic of the encapsulated graph-based CF models for the sake of introducing contrastive user-item pairs. In our solution, we choose to augment the output of encoder \( f \) to generate two different but related embeddings for representation learning. For comparison, we present our proposed framework specialized for CF, SelfCF, in Fig. 3b. The framework shares the same encoder between online and target networks, thus reduces the unnecessary memory and computational resources for storing and executing an additional encoder of the target network. We elaborate our framework in the following section.

3 THE SELFCF FRAMEWORK

Our framework (shown in Fig. 3b) partially inherits the Siamese network architecture of SimSiam, as shown in Fig. 3. In our framework, SelfCF, the goal is to learn informative representations of users and items based on positive user-item interactions only. The latent embeddings of users and items are learnt from the online network. Analogous to convolutions [27], which is a successful inductive bias via weight-sharing for modeling translation-invariance, the weight-sharing Siamese networks can model invariance with regard to more complicated transformations (e.g., data augmentations) [13]. The online and target networks in SelfCF use a same copy of the parameters as well as the backbone for modeling representation invariance. In addition, we drop the momentum encoder as used in BYOL and BUIR. As a result, with the same input, the online and target networks
will generate the same output which makes the loss totally vanished. We will discuss how to tackle this issue in the following section.

When considering data augmentations of input in CF, it is not a trivial task to distort the positive samples. In vision domain, where SSL is popularly applied, images can be easily distorted under a wide range of transformations. However, positive user-item pairs are difficult to be distorted while preserving their representation invariance. We use the following embedding perturbation techniques to achieve the same effect. For reasons of clarity, we denote bold value $E$ as the embedding matrix of users and items within a batch, and differentiate the embedding matrix of users with $E_u$, vice versa. The value $e$ in lowercase denotes the embedding of a user or item, specified as $e_u$ or $e_i$. 

**ACM Trans. Recomm. Syst.**
3.1 Data Augmentation via Output Perturbation

In vision, researchers use image transformations to augment input data and generate two different but relative reviews. Instead, our framework augments the output embeddings of users and items to generate two contrastive views. We propose three methods to introduce embedding perturbation in our framework, shown in Fig. 4. The historical embedding and embedding dropout are general techniques for output augmentation in our framework, while the edge pruning is specially designed for graph-based CF models.

**Historical embedding.** We introduce embedding perturbation by utilizing historical embeddings \([10, 15]\) from prior training iterations. Specifically, we use a momentum update to generate the contrastive embeddings in the target network. Suppose \(E'\) is the embeddings generated by a backbone encoder \(f\) in a batch \(B'\). The perturbed embeddings \(\tilde{E}'\) is calculated by combining of the output embeddings \(E'\) with the historical embedding \(E'_{t-1}\):

\[
\tilde{E}' = E'_{t-1} \tau + E'(1 - \tau)
\]

where \(\tau\) is a parameter controls the proportion of information preserved from a prior iteration.

**Embedding dropout.** We apply the embedding dropout scheme to perturb the embeddings of users and items from the target network. In classical CF models, the parameters are not modified until the loss is backpropagated. With the same input, to avoid null loss resulted from these models, our framework adopts embedding dropout on the resulted users’ and items’ vectors, which is analogous to node dropout \([40]\). In this way, the framework is able to generate two different but related views on the output, which are then feed into the loss function for optimization. The resulted embeddings under a dropout ratio \(p\) is calculated as:

\[
\tilde{E}' = E' \cdot \text{Bernoulli}(p)
\]

**Edge pruning.** As for graph-based CF models, the edge pruning method used in \([34, 37]\) provides an alternative way to augment the output embeddings. With the user-item bipartite graph, we randomly prune a certain proportion of edges from the graph in each batch. The output embeddings are updated by aggregating the embeddings of neighbors. With the same positive user-item pair, the output is distorted with different adjacency matrix (neighbors). Let \(A_{\text{pruned}}\) be the pruned adjacency matrix, then the resulted embeddings with edge pruning denote as:

\[
\tilde{E}' = E' \cdot A_{\text{pruned}}
\]

Note that, in implementation, edge pruning would require to calculate the adjacency matrix of users and items, which is more expensive in computation than the embedding dropout technique.

To summarize, our framework augments the output via embedding perturbation in the target network instead of distorting the input directly as commonly used in vision domain. It is worth noting that the historical embedding perturbation performs on embeddings from prior and current iteration, the embedding dropout perturbs the current embedding with noise, and the edge pruning method operates on future embeddings generated by stacking one more convolutional layer on current embeddings. Both historical embedding perturbation and embedding dropout perturbation remove the requirements of auxiliary graphs to generate a contrastive view as in \([28, 46, 49]\). We will discuss their performance with regard to this perspective in experiments section.

3.2 The Loss Function

Our framework, as shown in Fig. 3b, takes a positive user-item pair \((u, i)\) as input. The \((u, i)\) pair is initially processed by an encoder network \(f\) in a backbone \((e.g., \text{LightGCN}\ [20])\). The output of the encoder \(f\) is then copied to the target network for embedding perturbation. Formally, we denote the output of the encoder from the online network as \((e_u, e_i) = f(u, i)\). Finally, the linear predictor in our framework transforms the output
With the stop gradient operator, the target network receives no gradient from
\((\hat{e}_u, \hat{e}_i) = h(e_u, e_i)\) and matches it to the perturbed embeddings \((\tilde{e}_u, \tilde{e}_i) = g(e_u, e_i)\) in other view like in BYOL [17] and SimSiam [13].

We define a symmetrized loss function as the negative cosine similarity between \((\hat{e}_u, \hat{e}_i)\) and \((\tilde{e}_u, \tilde{e}_i)\):

\[
\mathcal{L} = \frac{1}{2} C(\hat{e}_u, \hat{e}_i) + \frac{1}{2} C(\tilde{e}_u, \tilde{e}_i)
\]  

(4)

Function \(C(\cdot, \cdot)\) in the above equation is defined as:

\[
C(e_u, e_i) = -\frac{(e_u)^T e_i}{||e_u||_2 ||e_i||_2}
\]

(5)

where \(|| \cdot ||_2\) is \(l_2\)-norm. The total loss is averaged over all user-item pairs in a batch. The intuition behind this is that we intend to maximize the prediction of the perturbed item \(i\) given a user \(u\), and vice versa. The minimized possible value for this loss is \(-1\).

Finally, we stop gradient on the target network and force the backpropagation of loss over the online network only. We follow the stop gradient (sg) operator as in [13, 17], and implement the operator by updating Equation 4 as:

\[
\mathcal{L} = \frac{1}{2} (C(\hat{e}_u, sg(\hat{e}_i)) + C(sg(\tilde{e}_u), \tilde{e}_i)).
\]

(6)

With the stop gradient operator, the target network receives no gradient from \((\hat{e}_u, \hat{e}_i)\). However, the encoder \(f\) in the online network receives gradients from user-item pair \((\hat{e}_u, \hat{e}_i)\), and optimizes its parameters towards the global optimum. Conversely, the removal of this operator can cause instability in online network learning, which we will verify this through ablation study. The reason is that the online and target networks simulate the student-teacher-like network [42] in which only the online network is optimized to predict the positively interacted item (user) presented by the target network. Additionally, we add regularization penalty on the online embeddings \((i.e., e_u\) and \(e_i\)) and the predictor \(h\). The final loss function is:

\[
\mathcal{L} = \frac{1}{2} \left( C(\hat{e}_u, sg(\hat{e}_i)) + C(sg(\tilde{e}_u), \tilde{e}_i)) + \lambda_1 \cdot (||e_u||_2^2 + ||e_i||_2^2) + \lambda_2 \cdot (||h||_1^2) \right),
\]

(7)

where \(|| \cdot ||_1\) is \(l_1\)-norm. The pseudo-code of SelfCF is in Algorithm 1.

**Algorithm 1** PyTorch-style pseudo-code for SelfCF.

**Require:** user-item interaction set \(\mathbb{B}\)

**Require:** \(f, h, g\)  \(\triangleright\) encoder, predictor, output perturbation

1. for \(\mathbb{B}^t\) in \(\mathbb{B}\) do  \(\triangleright\) load a batch
2. \((E_{ut}^t, E_{it}^t) = f(\mathbb{B}^t)\)  \(\triangleright\) output of encoder
3. \((\tilde{E}_{ut}^t, \tilde{E}_{it}^t) = h(E_{ut}^t, E_{it}^t)\)  \(\triangleright\) output of predictor
4. \((\tilde{E}_{ut}^t, \tilde{E}_{it}^t) = g(E_{ut}^t, E_{it}^t)\)  \(\triangleright\) output of perturbation
5. \(\mathcal{L} = \frac{1}{2} \left( C(\tilde{E}_{ut}^t, sg(\tilde{E}_{it}^t)) + C(sg(\tilde{E}_{ut}^t), \tilde{E}_{it}^t) \right) + \lambda_1 \cdot (||E_{ut}^t||_2^2 + ||E_{it}^t||_2^2) + \lambda_2 \cdot (||h||_1^2)\)  \(\triangleright\) Eq. 7
6. \(\mathcal{L}.backward()\)  \(\triangleright\) back-propagate
7. update\((f, h)\)  \(\triangleright\) parameters update
8. end for

9. def predict\((e_u, e_i)\):  \(\triangleright\) calculate recommendation score
10. return \(s(e_u, e_i)\)  \(\triangleright\) Eq. 8

ACM Trans. Recomm. Syst.
Table 2. Statistics of the experimented data.

| Dataset | # of Users | # of Items | # of Interactions | Sparsity   |
|---------|------------|------------|-------------------|------------|
| Arts    | 45,624     | 21,104     | 396,556           | 99.9588%  |
| Games   | 50,677     | 16,897     | 454,529           | 99.9469%  |
| Food    | 115,144    | 39,688     | 1,025,169         | 99.9776%  |
| COCO    | 144,773    | 20,969     | 1,204,697         | 99.9603%  |

3.3 Top-K Recommendation

Classical CF methods recommend top-K items by ranking scores of the inner product of a user embedding with all candidate item embeddings. However, in SSL, we minimize the predicted loss between \( u \) and \( i \) for each positive interaction \((u, i)\). Intuitively, we predict the future interaction score based on a cross-prediction task [28]. That is, we both predict the interaction probability of item \( i \) with \( u \) and the probability of user \( u \) with \( i \). Given \((e_u, e_i)\) being the output of the encoder \( f \), the recommendation score is calculated as:

\[
s(e_u, e_i) = h(e_u) \cdot (e_i)^T + e_u \cdot h(e_i)^T
\]  

(8)

It is worth noting that since the encoder \( f \) is shared between both online and target networks, we use the representations obtained from the online network to predict top-K items for each user.

4 EXPERIMENTS

We evaluate the framework on four publicly available datasets and compare its performance with BUIR [28] and eight baselines by encapsulating BPR and LightGCN as our backbone. Our framework is mainly compared with BUIR, as it is the only recommendation framework that works without negative samples. All baselines as well as our frameworks are trained on a single GeForce RTX 2080 Ti (11 GB).

We list the research questions addressed in our evaluation as follows:

**RQ1:** Whether the self-supervised models that only leverage positive user-item interactions can outperform their supervised counterparts?

**RQ2:** How SelfCF shapes the recommendation results for cold-start and loyal users?

**RQ3:** Why SelfCF works, and which component is essential in preventing collapsing?

We address the first research question by evaluating our framework against supervised baselines with four datasets under six evaluation metrics. Next, we dive into the recommendation results of the baselines under both supervised and self-supervised settings and analyze their performance on users with different number of interactions. Finally, to investigate the underlying mechanisms of SelfCF, we perform ablation study on the components of SelfCF, such as the linear predictor, the loss function etc.

4.1 Dataset Description.

We choose the evaluated datasets carefully by considering the following principles in order to introduce as much as diversity.

**Domain:** Interactions within the same domain may exhibit similar patterns across datasets. Hence, we choose evaluation datasets from two different domains ranging from education to e-commerce under different categories.

**Released date:** Existing recommender systems usually evaluated on out-dated datasets nearly collected 10 years ago. With the rapid growth of e-commerce platforms, user behaviors are gradually shaped with online purchasing.
Graph size: The user-item interactions can be viewed as a bipartite graph (Fig. 1), we consider the graph size with the number of nodes ranging from 10K to 100K.

We describe each dataset with regard to the above selection principles.

- Amazon Video Games (Games): This is a newly released version of the Amazon-Video-Games review dataset in 2018. We select the rating only version for evaluation. Dataset is available from [35] ¹.
- Amazon Arts, Crafts and Sewing (Arts): This dataset is similar to the Amazon Video Games dataset under a different genre.
- Amazon Grocery and Gourmet Food (Food): This dataset has a large-scale interaction graph with more than 100K users.
- COCO: A large-scale dataset from education domain. The raw dataset includes over 43K online courses and 2.5M learners [14].

All raw datasets are preprocessed with a 5-core setting on both items and users and the filtered results are presented in Table 2.

4.2 Encapsulated Baselines and Framework BUIR.

To compare the performance of our proposed framework, we first consider the following baselines that adopt negative sampling for supervised learning except the popularity algorithm.

- Pop: Popularity algorithm recommends the most popular items to each user.
- BPR [36]: A matrix factorization model optimized by a pairwise ranking loss in a Bayesian way.
- MultiVAE [30]: It is a generative model that adopts variational auto-encoder (VAE) for item-based CF. It uses a multinomial likelihood to fit the distribution of data and adopts Bayesian inference for parameter estimation.
- EHCF [9]: This is an efficient recommendation model that learns the representations of users and items by reconstructing the interaction matrix without negative sampling. It takes all unobserved user-item pairs as negative samples.
- NGCF [45]: This model explicitly injects collaborative signal from high-order connectivity of user-item graph into the embedding learning process.
- LR-GCCF [11]: The model first simplifies the vanilla GCN by removing nonlinear function, then it uses a residual preference learning process for prediction.
- LightGCN [20]: This is a simplified graph convolution network that only performs linear propagation and aggregation between neighbors. The hidden layer embeddings are averaged to calculate the final user/item embeddings for prediction.
- SimGCL [49]: This self-supervised model injects uniform noises into the latent embeddings to generate contrastive views.

We also consider the following self-supervised frameworks that learn the representations of users and items without negative samples. Our framework is mainly compared with BUIR [28], a self-supervised framework that is derived from BYOL [17]. Its architecture follows the Siamese network in Fig. 3a. To compare the performance of our proposed framework, we encapsulate two state-of-the-art models, BPR and LightGCN, into the frameworks. That is, we substitute the encoder $f$ in Fig. 3b with BPR and LightGCN, respectively.

- BUIR [28]: This framework uses asymmetric network architecture to update its backbone network parameters.
- SelfCF_{he}: Our proposed framework that uses historical embedding for data augmentation.
- SelfCF_{ed}: Our proposed framework that uses embedding dropout for data augmentation.

¹https://nijianmo.github.io/amazon/index.html
SelfCF: A Simple Framework for Self-supervised Collaborative Filtering

Table 3. Hyper-parameter exploration.

| Framework   | Backbone model | Para.      | Tuning Range          |
|-------------|----------------|------------|-----------------------|
| SelfCFhe & SelfCFep | LightGCN | layers      | [1, 2, 3, 4]          |
|              |               | momentum   | [0.1, 0.2, 0.5]       |
|              |               | \( \lambda_1 \) | [0.0, 1e-01, 1e-02, 1e-03, 1e-04, 1e-05] |
|              |               | \( \lambda_2 \) | [0.0]           |
| BPR         | LightGCN      | dropout    | [0.05]                |
| SelfCFed    | LightGCN      | layers     | [1, 2]                |
|              |               | dropout    | [0.1, 0.2, 0.5]       |
|              |               | \( \lambda_1 \) | [0.0, 1e-01, 1e-02, 1e-03, 1e-04, 1e-05] |
|              |               | \( \lambda_2 \) | [0.0]           |

- **SelfCFep**: Our proposed framework that uses edge pruning for data augmentation.

To demonstrate the generalization of our framework, we consider two backbone networks for BUIR and SelfCFed, the classic BPR and recently graph-based LightGCN. Other frameworks will only use LightGCN as their backbone network because LightGCN always shows better performance against BPR.

### 4.3 Evaluation Metrics.

We use \( \text{Recall}@K \) and \( \text{NDCG}@K \) computed by the all-ranking protocol as the evaluation metrics for recommendation accuracy comparison. In the recommendation phase, all items that have not been interacted with a specific user are regarded as candidates. That is, we do not use sampled evaluation.

Formally, we define \( I_u^i(i) \) as the \( i \)-th ranked item recommended for \( u \), \( I[\cdot] \) is the indicator function, and \( I_u^i \) is the set of items that user \( u \) interacted in the testing data.

Recall@K for users \( u \) is:

\[
\text{Recall}@K(u) = \frac{\sum_{i=1}^{K} \mathbb{I}[I_u^i(i) \in I_u^i]}{|I_u^i|} \tag{9}
\]

The Discounted Cumulative Gain (DCG@K) is:

\[
\text{DCG}@K(u) = \sum_{i=1}^{K} \frac{2^{|I_u^i(i)\in I_u^i|} - 1}{\log(i+1)} \tag{10}
\]

\( \text{NDCG}@K \) is normalized to \([0, 1]\) with \( \text{NDCG}@K = \frac{\text{DCG}@K}{\text{IDCG}@K} \), where \( \text{IDCG}@K \) is calculated by sorting the interacted items in the testing data at top and then use the formula for \( \text{DCG}@K \). We set \( K = 10 \), \( K = 20 \) and \( K = 50 \) in our experimental comparison, respectively. For simplicity, we denote \( \text{Recall}@K \) and \( \text{NDCG}@K \) with \( R@K \) and \( N@K \) in the following sections.

### 4.4 Hyper-parameters Settings.

Same as other work \([11, 20]\), we fix the embedding size of both users and items to 64 for all models, initialize the embedding parameters with the Xavier method \([16]\), and use Adam \([25]\) as the optimizer. For a fair comparison, we carefully tune the parameters of each model following their published papers. For our proposed frameworks, we perform grid search across all datasets to conform the optimal settings. We summarize the settings in Table 3. We
We conclude our analysis to address research question **RQ1**: Both classical CF and graph-based model CF can benefit from SelfCF. Specially, the supervised counterparts, BPR and LightGCN, can be improved with up to 7.36% and 6.55% across the four datasets under SelfCFed, respectively.

### 4.6 Efficiency of SelfCF

We evaluate the efficiency of SelfCF compared with LightGCN with regard to the number of layers in Table 5. From the results, we observe SelfCFed is on par or better than LightGCN stacked with 4-layer, but requires only one half to one quarter training time of LightGCN.

ACM Trans. Recomm. Syst.
Table 4. Overall performance comparison. We mark the global best results on each dataset under each metric in **boldface**, and the second best underlined. We also calculate the performance improvement by SelfCF on BUIR over each evaluation metric as $\Delta$. "NS" denotes Negative Samples.

|          | Framework | Model   | R@10 | R@20 | R@50 | N@10 | N@20 | N@50 |
|----------|-----------|---------|------|------|------|------|------|------|
| **Arts** | Non-parametric | -       | 0.0091 | 0.0164 | 0.0283 | 0.0072 | 0.0095 | 0.0128 |
| Supervised (with NS) | -       | BPR | 0.0201 | 0.0327 | 0.0589 | 0.0137 | 0.0177 | 0.0245 |
|          |           | MultiVAE | 0.0171 | 0.0268 | 0.0503 | 0.0113 | 0.0145 | 0.0205 |
|          |           | EHCF | 0.0202 | 0.0319 | 0.0567 | 0.0136 | 0.0175 | 0.0240 |
|          |           | NGCF | 0.0205 | 0.0342 | 0.0623 | 0.0142 | 0.0186 | 0.0260 |
|          |           | LR-GCCF | 0.0221 | 0.0365 | 0.0636 | 0.0151 | 0.0197 | 0.0268 |
|          |           | LightGCN | 0.0231 | 0.0371 | 0.0663 | 0.0156 | 0.0201 | 0.0277 |
|          | Self-Supervised (with NS) | SimGCL | 0.0198 | 0.0322 | 0.0558 | 0.0133 | 0.0172 | 0.0234 |
|          | Self-Supervised (without NS) | BUIR | 0.0197 | 0.0309 | 0.0560 | 0.0139 | 0.0174 | 0.0239 |
|          |           | LightGCN | 0.0208 | 0.0334 | 0.0636 | 0.0149 | 0.0190 | 0.0270 |
|          |          | BPR | 0.0236 | 0.0397 | 0.0709 | 0.0157 | 0.0208 | 0.0289 |
|          |          | LightGCN | 13.46% | 18.86% | 11.48% | 5.37% | 94.7% | 7.04% |
|          | SelfCFhe | BPR | 0.0231 | 0.0354 | 0.0632 | 0.0157 | 0.0197 | 0.0269 |
|          |          | LightGCN | 17.26% | 14.56% | 12.86% | 12.95% | 13.22% | 12.55% |
|          |          | LightGCN | 0.0246 | 0.0391 | 0.0708 | 0.0170 | 0.0218 | 0.0300 |
|          |          | LightGCN | 18.27% | 17.07% | 11.32% | 14.09% | 14.74% | 11.11% |
|          | SelfCFed | BPR | 0.0239 | 0.0395 | 0.0714 | 0.0158 | 0.0208 | 0.0290 |
|          |          | LightGCN | 14.90% | 18.26% | 12.26% | 6.04% | 94.7% | 7.41% |
|          | SelfCFep | BPR | 0.0241 | 0.0402 | 0.0744 | 0.0152 | 0.0200 | 0.0285 |
|          |          | LightGCN | 29.96% | 23.18% | 14.69% | 25.52% | 25.62% | 22.55% |
|          |          | LightGCN | 32.60% | 34.64% | 24.17% | 32.17% | 34.95% | 26.98% |
|          | Games | Framework | Model | R@10 | R@20 | R@50 | N@10 | N@20 | N@50 |
| Non-parametric | -       | Pop | 0.0117 | 0.0175 | 0.0379 | 0.0049 | 0.0067 | 0.0117 |
| Supervised (with NS) | -       | BPR | 0.0210 | 0.0369 | 0.0699 | 0.0135 | 0.0183 | 0.0265 |
|          |           | MultiVAE | 0.0238 | 0.0376 | 0.0718 | 0.0154 | 0.0196 | 0.0280 |
|          |           | EHCF | 0.0278 | 0.0445 | 0.0772 | 0.0175 | 0.0227 | 0.0308 |
|          |           | NGCF | 0.0254 | 0.0425 | 0.0825 | 0.0166 | 0.0217 | 0.0314 |
|          |           | LR-GCCF | 0.0259 | 0.0446 | 0.0824 | 0.0171 | 0.0228 | 0.0320 |
|          |           | LightGCN | 0.0275 | 0.0461 | 0.0841 | 0.0175 | 0.0231 | 0.0326 |
|          | Self-Supervised (with NS) | SimGCL | 0.0310 | 0.0502 | 0.0879 | 0.0194 | 0.0251 | 0.0344 |
|          | Self-Supervised (without NS) | BUIR | 0.0217 | 0.0361 | 0.0674 | 0.0135 | 0.0180 | 0.0257 |
|          |          | LightGCN | 0.0227 | 0.0384 | 0.0749 | 0.0143 | 0.0192 | 0.0282 |
|          |          | LightGCN | 0.0295 | 0.0473 | 0.0859 | 0.0187 | 0.0241 | 0.0336 |
|          |          | LightGCN | 29.96% | 23.18% | 14.69% | 25.52% | 25.62% | 22.55% |
|          |          | LightGCN | 32.60% | 34.64% | 24.17% | 32.17% | 34.95% | 26.98% |
|          | SelfCFhe | BPR | 0.0241 | 0.0402 | 0.0744 | 0.0152 | 0.0200 | 0.0285 |
|          |          | LightGCN | 11.06% | 11.36% | 10.39% | 12.59% | 11.11% | 10.89% |
|          |          | LightGCN | 0.0289 | 0.0485 | 0.0857 | 0.0181 | 0.0371 | 0.0332 |
|          |          | LightGCN | 27.31% | 26.30% | 14.42% | 26.57% | 25.00% | 17.73% |
|          | SelfCFed | BPR | 0.0301 | 0.0517 | 0.0930 | 0.0189 | 0.0255 | 0.0358 |
|          |          | LightGCN | 32.60% | 34.64% | 24.17% | 32.17% | 34.95% | 26.98% |
| Framework Model | R@10 | R@20 | R@50 | N@10 | N@20 | N@50 |
|------------------|------|------|------|------|------|------|
| **Non-parametric** | Pop  | 0.0125 | 0.0189 | 0.0346 | 0.0112 | 0.0133 | 0.0173 |
| **Supervised (with NS)** | | BPR | 0.0138 | 0.0222 | 0.0390 | 0.0097 | 0.0124 | 0.0167 |
| | | MultiVAE | 0.0133 | 0.0208 | 0.0374 | 0.0092 | 0.0116 | 0.0159 |
| | | EHCF | 0.0158 | 0.0243 | 0.0416 | 0.0111 | 0.0137 | 0.0182 |
| | | NGCF | 0.0158 | 0.0254 | 0.0456 | 0.0102 | 0.0132 | 0.0185 |
| | | LR-GCCF | 0.0172 | 0.0277 | 0.0478 | 0.0120 | 0.0154 | 0.0206 |
| | | LightGCN | 0.0184 | 0.0286 | 0.0497 | 0.0125 | 0.0157 | 0.0211 |
| **Self-Supervised (with NS)** | SimGCL | 0.0173 | 0.0265 | 0.0453 | 0.0116 | 0.0147 | 0.0195 |
| **Self-Supervised (without NS)** | BUIR | BPR | 0.0113 | 0.0178 | 0.0313 | 0.0075 | 0.0096 | 0.0130 |
| | | LightGCN | 0.0145 | 0.0236 | 0.0469 | 0.0111 | 0.0141 | 0.0201 |
| | LightGCN | BPR | 0.0165 | 0.0259 | 0.0443 | 0.0111 | 0.0141 | 0.0188 |
| Δ | 34.48% | 26.69% | 10.02% | 18.92% | 17.73% | 9.95% |
| SelfCF | LightGCN | BPR | 0.0198 | 0.0256 | 0.0316 | 0.0075 | 0.0092 | 0.0135 |
| Δ | 15.25% | 15.71% | 12.87% | 13.81% | 14.46% | 13.20% |
| SelfCF | LightGCN | BPR | 0.1210 | 0.1674 | 0.2507 | 0.0724 | 0.0880 | 0.1078 |
| Δ | 36.55% | 33.90% | 18.34% | 21.62% | 22.70% | 16.92% |
| SelfCF | LightGCN | BPR | 0.1126 | 0.1784 | 0.2734 | 0.0754 | 0.0923 | 0.1147 |
| Δ | 1.03% | -0.63% | 1.50% | 2.92% | 1.46% | 2.16% |

| Framework Model | R@10 | R@20 | R@50 | N@10 | N@20 | N@50 |
|------------------|------|------|------|------|------|------|
| **Non-parametric** | Pop  | 0.0574 | 0.0798 | 0.1393 | 0.0318 | 0.0385 | 0.0525 |
| **Supervised (with NS)** | | BPR | 0.1181 | 0.1745 | 0.2681 | 0.0741 | 0.0908 | 0.1129 |
| | | MultiVAE | 0.1243 | 0.1816 | 0.2786 | 0.0775 | 0.0946 | 0.1175 |
| | | EHCF | 0.1146 | 0.1674 | 0.2507 | 0.0724 | 0.0880 | 0.1078 |
| | | NGCF | 0.1210 | 0.1817 | 0.2843 | 0.0740 | 0.0921 | 0.1163 |
| | | LR-GCCF | 0.1215 | 0.1784 | 0.2734 | 0.0754 | 0.0923 | 0.1147 |
| | | LightGCN | 0.1213 | 0.1781 | 0.2723 | 0.0762 | 0.0932 | 0.1154 |
| **Self-Supervised (with NS)** | SimGCL | 0.1238 | 0.1758 | 0.2564 | 0.0784 | 0.0939 | 0.1130 |
| **Self-Supervised (without NS)** | BUIR | BPR | 0.0977 | 0.1445 | 0.2222 | 0.0601 | 0.0740 | 0.0924 |
| | | LightGCN | 0.1162 | 0.1745 | 0.2672 | 0.0719 | 0.0893 | 0.1113 |
| | LightGCN | BPR | 0.1126 | 0.1672 | 0.2508 | 0.0684 | 0.0847 | 0.1046 |
| Δ | -1.29% | 0.74% | 1.87% | -0.42% | 0.56% | 1.26% |
| SelfCF | LightGCN | BPR | 0.1287 | 0.1892 | 0.2877 | 0.0796 | 0.0977 | 0.1210 |
| Δ | 10.76% | 8.42% | 7.67% | 10.71% | 9.41% | 8.72% |
| SelfCF | LightGCN | BPR | 0.1174 | 0.1734 | 0.2712 | 0.0740 | 0.0906 | 0.1137 |
| Δ | 1.03% | -0.63% | 1.50% | 2.92% | 1.46% | 2.16% |
Table 5. Efficiency of SelfCF.

| Dataset | Model     | R@10 | R@20 | R@50 | N@10 | N@20 | N@50 | Time (s) |
|---------|-----------|------|------|------|------|------|------|----------|
| Games   | SelfCF$_{ed}$ 1-Layer | 0.0274 | 0.0456 | 0.0857 | 0.0175 | 0.0231 | 0.0332 | 3.19 |
| Games   | SelfCF$_{ed}$ 2-Layer | 0.0289 | 0.0485 | 0.0857 | 0.0181 | 0.0240 | 0.0332 | 3.75 |
| Games   | LightGCN 4-Layer | 0.0275 | 0.0461 | 0.0841 | 0.0175 | 0.0231 | 0.0326 | 8.22 |
| Games   | LightGCN 3-Layer | 0.0270 | 0.0458 | 0.0836 | 0.0176 | 0.0233 | 0.0326 | 7.60 |
| Games   | LightGCN 2-Layer | 0.0271 | 0.0454 | 0.0818 | 0.0174 | 0.0230 | 0.0320 | 6.78 |
| Games   | LightGCN 1-Layer | 0.0263 | 0.0448 | 0.0798 | 0.0172 | 0.0228 | 0.0315 | 5.05 |
| Food    | SelfCF$_{ed}$ 1-Layer | 0.0197 | 0.0316 | 0.0547 | 0.0135 | 0.0173 | 0.0233 | 14.09 |
| Food    | SelfCF$_{ed}$ 2-Layer | 0.0198 | 0.0316 | 0.0555 | 0.0135 | 0.0173 | 0.0235 | 17.20 |
| Food    | LightGCN 4-Layer | 0.0184 | 0.0286 | 0.0497 | 0.0125 | 0.0157 | 0.0211 | 59.81 |
| Food    | LightGCN 3-Layer | 0.0176 | 0.0280 | 0.0484 | 0.0122 | 0.0155 | 0.0207 | 48.54 |
| Food    | LightGCN 2-Layer | 0.0177 | 0.0280 | 0.0482 | 0.0121 | 0.0154 | 0.0206 | 41.46 |
| Food    | LightGCN 1-Layer | 0.0167 | 0.0267 | 0.0460 | 0.0118 | 0.0149 | 0.0198 | 26.02 |

4.7 Understanding the Learning of SelfCF

In this section, we attempt to answer "why do SelfCF framework work well for recommendation?" Based on the line of work [1, 52], we hypothesize that the "stop-gradient" design in SelfCF has the de-correlation effect on learning informative representations. Following [1], we define the covariance matrix of $E = [e_1, \ldots, e_n]$ as:

$$C(E) = \frac{1}{n-1} \sum_{i=1}^{n} (e_i - \bar{e})(e_i - \bar{e})^T,$$

where $\bar{e} = \frac{1}{n} \sum_{i=1}^{n} e_i$. (11)

The covariance regularization term $c$ is defined as the sum of the squared off-diagonal coefficients of $C(E)$:

$$c(E) = \frac{1}{d} \sum_{i \neq j} [C(E)]_{i,j}^2.$$ (12)

where $1/d$ is a scale factor. A lower covariance value indicates a better de-correlation effect on representations.

Then, we compare the performance of SelfCF with and without "stop-gradient" component in training on Food dataset, as shown in Fig. 5. From the figure, we observe SelfCF with "stop-gradient" can decrease the covariance value of learnt representations to a significant extent compared with that of no "stop-gradient". Thanks to the de-correlation effect of SelfCF, the performance on recommendation with regard to Recall@20 is consistently improved with training. In contrast, its performance collapses at a fixed level with the removal of the "stop-gradient" component.

4.8 Hyper-parameter Sensitivity

To guide the selection of parameters of our framework, we preform a hyper-parameter study on the performance of SelfCF. In the implementation, we use the Food dataset as the evaluated dataset and LightGCN as the backbone of SelfCF. The results on Games and other datasets show similar patterns with Food, we put the results on Games in the Appendix for reference. We investigate the performance changes of our framework with regard to hyper-parameters on the momentum in historical embedding, the number of layers, the ratio of embedding dropout and the proportion of edges pruned in SelfCF.
The number of layers. We study how layers in LightGCN affect the performance of SelfCF by stepping its range from \([1, 2, 3, 4, 5, 6, 7, 8]\). We plot the results in Fig. 6.

SelfCF_{he} and SelfCF_{ed} show relatively slow performance degradation as the number of layers increasing. The performance of SelfCF_{ep} is not stable with regard to the number of layers. On the contrary, the performance of SelfCF_{ed} is not very affected with the number of layer in LightGCN. SelfCF_{ed} is capable of boosting up the performance of recommendation for the graph-based models within few layers.

The momentum/dropout and regularization coefficient. We set both the momentum of SelfCF_{he} and the dropout of SelfCF_{ed} value in the range of \([0.1, 0.6]\) with a step of 0.1. The \(L2\) regularization coefficient \(\lambda_1\) is searched in the range of \([0.0, 1e-05, 1e-04, ..., 1e-01]\). We plot the heatmap for SelfCF_{he}, SelfCF_{ed}, SelfCF_{ep} over Recall@20 and NDCG@20 in Fig. 7 and Fig. 8, respectively.

From Fig. 7 and Fig. 8, we observe the performance of SelfCF on Recall@20 is consistence with NDCG@20. Higher on Recall usually results in higher NDCG. The performance of our framework is less sensitive to the momentum and dropout than the regularization factor. In practice, it is preferable to put weak regularization to normalize the learned embeddings.

The hyper-parameter studies also show that three variants of SelfCF exhibit similar behaviors. Hence, we analyze recommendation result and perform ablation study on SelfCF_{ed} in the following sections.
Fig. 7. Performance of Recall@20 for three variations of SelfCF with respect to hyper-parameters of momentum, embedding dropout, edge dropout and regularization coefficient.

Fig. 8. Performance of NDCG@20 for three variations of SelfCF with respect to hyper-parameters of momentum, embedding dropout, edge dropout and regularization coefficient.

Fig. 9. The recommendation results on different degree of users. User’ degree indicates the number of interactions of a user.

4.9 Diving into the Recommendation Results

In our framework, we recommend top-K items to users relying solely on positive user-item interaction pairs. We further study how our framework differentiate with the supervised models in recommendation results. Specifically, we encapsulate LightGCN into our framework, and compare the recommendation results between SelfCF and LightGCN with regard to users’ degree on Food. We plot the results in Fig. 9.

On metrics Recall@50 and NDCG@50, we see SelfCF outperforms LightGCN in every category. Our proposed framework is able to alleviate the cold-start issue to certain extent. The most significant improvement (14.4%) is
observed on cold-start users, occupied about 63.92% users in the testing dataset. The second highest improvement is observed with loyal users, which gains 12.80%. From our data analysis, we find users with a high degree of interactions in the training are prefer to select items with a low degree in the testing. Thus, it is difficult to recommend the right items to these users. Our self-supervised framework can partial tackle the problem of commendation degradation on loyal user [24]. We speculate the underlying reason is that for these users the supervised models sample a large number of unobserved but potentially positive items for training, which makes the models unable to consider the negatively sampled items in recommendation list.

Regarding research question RQ2, SelfCF boosts up the recommendation performance of all users. Especially for cold-start users, it improves the recommendation accuracy of LightGCN by 14.4% on Recall@50.

4.10 Representation of nodes

SelfCF leverages positive samples merely to learn the latent user and item representations. We examine how the representations differ between supervised and self-supervised learnings. We draw 2D t-SNE plots of node representations learned from LightGCN and SelfCF<sub>ed</sub> with regard to Food dataset in Fig 10. For computational complexity consideration, we only plot the representations of users and items in the test set. In this figure, we can observe the representations of users and items are highly melded with each other in the supervised model, LightGCN. On the contrary, the representations of users are items are repulsed apart in SelfCF<sub>ed</sub>. The result is in line with Equation 6. Without the negative samples, the loss function is unable to enforce the embeddings of positive users and items similar to each other. Instead, it maximizes the similarity subject to different conditions. That is, our proposed self-supervised model encourages similar users congregate in a group, and similar items cluster together with each other.

5 ABLATION STUDY

We investigate each component in SelfCF to study its contribution to the recommendation performance. All ablation studies are performed on SelfCF<sub>ed</sub> trained on Food dataset. The encapsulated baseline is LightGCN with two convolutional layers. The dropout of embedding for SelfCF<sub>ed</sub> is set as 0.5.

5.1 Predictor

We study the recommendation performance considering predictor $h$ under several variants. Table 6 summarizes the variants and their recommendation performance.

Different from the predictor in SimSiam [13], our framework still works by removing the predictor $h$, but resulting in performance degeneration to the level of Popularity algorithm. A fixed random initialization with
Table 6. Impact of predictor $h$.

| Predictor               | R@10  | R@20  | R@50  | N@10  | N@20  | N@50  |
|------------------------|-------|-------|-------|-------|-------|-------|
| 1-layer MLP            | 0.0198| 0.0316| 0.0555| 0.0135| 0.0173| 0.0235|
| No predictor           | 0.0124| 0.0191| 0.0359| 0.0109| 0.0132| 0.0176|
| Fixed random init.     | 0.0127| 0.0191| 0.0309| 0.0109| 0.0129| 0.0160|
| 2-layer MLP            | 0.0199| 0.0313| 0.0545| 0.0137| 0.0173| 0.0233|

Table 7. Effectiveness of loss function.

| Similarity loss | R@10  | R@20  | R@50  | N@10  | N@20  | N@50  |
|-----------------|-------|-------|-------|-------|-------|-------|
| Cosine          | 0.0198| 0.0316| 0.0555| 0.0135| 0.0173| 0.0235|
| Cross-entropy   | 0.0124| 0.0191| 0.0350| 0.0110| 0.0133| 0.0174|

Table 8. Effectiveness of stop-gradient (sg) operator.

| Case     | sg | Predictor | R@10  | R@20  | R@50  | N@10  | N@20  | N@50  |
|----------|----|-----------|-------|-------|-------|-------|-------|-------|
| Baseline | ✓  | ✓         | 0.0198| 0.0316| 0.0555| 0.0135| 0.0173| 0.0235|
| (a)      | -  | ✓         | 0.0124| 0.0191| 0.0359| 0.0109| 0.0133| 0.0176|
| (b)      | -  | -         | 0.0124| 0.0191| 0.0359| 0.0109| 0.0132| 0.0176|

the predictor makes the self-supervised framework difficult to learn good representations of users/items. On the contrary, a 2-layer MLP also achieves a competitive performance as the 1-layer version.

5.2 Loss function

In contrastive learning, it is a common practice for losses measuring a cosine similarity [12, 17, 44]. We substitute the loss function with cross-entropy similarity by modifying $C$ with:

$$C(e_u, e_i) = -\text{softmax}(e_i) \cdot \log(\text{softmax}(e_u))$$  \hspace{1cm} (13)

Table 7 shows the results compared with cosine similarity. The cross-entropy similarity can prevent the solution collapsing to some extent. Cosine similarity captures the interaction preference between user and item directly, hence shows better performance.

5.3 Stop-gradient

Existing researches [4, 13, 17, 64] on SSL highlight the crucial role of stop-gradient in preventing solution collapsing. We evaluate with adding or removing the stop-gradient operator with/without a linear predictor. The results in Table 8 show that our self-supervised framework works even under a completely symmetry setting. The loss function of Equation 6 is able to capture the invariant and salient features in the embeddings of users/items by dropping the noise signal. However, without the “stop gradient” operator, the performance of SelfCF decreases greatly. We speculate the loss backpropagated to both directions (online and target networks) leads to the framework unable to learn the optimal parameters of the baseline.
Based on our ablation studies with regard to research question RQ3, we observe that SelfCF does not rely on a single component for preventing solution collapsing. It shows a different behavior from other self-supervised models, in which the "stop gradient" operator is identified as a crucial component to prevent solution collapsing [13]. The underlying reason is that our loss function is designed as the similarity between latent embeddings of user and item, hence it can capture the preference of user to some extent.

6 CONCLUSION AND FUTURE DIRECTIONS
In this paper, we propose a framework on top of Siamese networks to learn representation of users and items without negative samples or labels. We argue the self-supervised learning techniques that widely used in vision cannot be directly applied in recommendation domain. Hence we design a Siamese network architecture that perturbs the output of backbone instead of augmenting the input data. By encapsulating two popular recommendation models into the framework, our experiments on four datasets show the proposed framework is on par or better than other self-supervised framework, BUIR. The performance is also competitive to the supervised counterpart, obtaining a gain of 6.55% over LightGCN. We hope our study will shed light on further researches on self-supervised collaborative filtering.

We also discuss the potential directions based on our framework, SelfCF. a. More powerful predictor. In the above ablation study, we observed both 1-layer MLP and 2-layer MLP show promising performance. Other than the supervised baselines, e.g., LightGCN, the predictor is a crucial component in our framework. Our framework learns not only the representation of users and items but also the parameters of the predictor. As a result, future researches can be paid to the design of predictor in our framework. b. Combining with supervised signals. In recent years, a line of work integrates self-supervised learning into the classic pairwise BPR supervised loss and show promising improvement on recommendation performance [33, 46, 49, 50]. However, in our framework, we only use the positive user-item interactions pairs for recommendation. It is worth researching the integration of supervised signals into our framework. c. Embedding augmentation methods. This paper proposes three embedding perturbation techniques that can be divided into two categories. i). Graph-based augmentation. Like BUIR, the edge pruning methods use another graph network to generate the contrastive embeddings. ii). Non-graph-based augmentation. The other two techniques directly distort the original view and can significantly ease the computation burden. However, other methods [33] for embedding augmentation can be explored in SelfCF. d. Fusing multimodal features for effective recommendation. To alleviate the data sparsity problem and the cold start issue in CF, various methods have been developed to fuse the multimodal information (e.g., text descriptions and images) of items into the current CF paradigm [59]. In this direction of future work, we are interested in exploring effective ways of fusing multimodal features for recommendation.

ACKNOWLEDGMENTS
This work is partially supported by the MOE AcRF Tier 1 funding (RG90/20), Alibaba Group through Alibaba Innovative Research (AIR) Program and Alibaba-NTU Singapore Joint Research Institute (JRI), Nanyang Technological University, Singapore. We thank Professor Toru Ishida from the Department of Computer Science of Hong Kong Baptist University for his valuable advices on this work.

REFERENCES
[1] Adrien Bardes, Jean Ponce, and Yann Lecun. 2022. VICReg: Variance-Invariance-Covariance Regularization for Self-Supervised Learning. In ICLR 2022-10th International Conference on Learning Representations.
[2] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2018. Graph convolutional matrix completion. In Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery and data mining.
[3] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1993. Signature verification using a “siamese” time delay neural network. Advances in neural information processing systems 6 (1993).
[4] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. 2020. Unsupervised learning of visual features by contrasting cluster assignments. In Proceedings of the 34th Annual Conference on Neural Information Processing Systems.

[5] Chao Chen, Dongsheng Li, Junchi Yan, and Xiaokang Yang. 2021. Modeling Dynamic User Preference via Dictionary Learning for Sequential Recommendation. IEEE Transactions on Knowledge and Data Engineering (2021).

[6] Chong Chen, Weizhi Ma, Min Zhang, Chenyang Wang, Yiqun Liu, and Shaoqing Ma. 2022. Revisiting Negative Sampling VS. Non-Sampling in Implicit Recommendation. ACM Transactions on Information Systems (TOIS) (2022).

[7] Chong Chen, Min Zhang, Weizhi Ma, Yiqun Liu, and Shaoqing Ma. 2020. Efficient non-sampling factorization machines for optimal context-aware recommendation. In Proceedings of The Web Conference 2020. 2400–2410.

[8] Chong Chen, Min Zhang, Weizhi Ma, Yiqun Liu, and Shaoqing Ma. 2020. Jointly non-sampling learning for knowledge graph enhanced recommendation. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 189–198.

[9] Chong Chen, Min Zhang, Yongfeng Zhang, Weizhi Ma, Yiqun Liu, and Shaoqing Ma. 2020. Efficient heterogeneous collaborative filtering without negative sampling for recommendation. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34. 19–26.

[10] Jianfei Chen, Jun Zhu, and Le Song. 2017. Stochastic training of graph convolutional networks with variance reduction. In International Conference on Machine Learning.

[11] Lei Chen, Le Wu, Richang Hong, Kun Zhang, and Meng Wang. 2020. Revisiting graph based collaborative filtering: A linear residual graph convolutional network approach. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34. 27–34.

[12] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In International Conference on Machine Learning. 1597–1607.

[13] Xinlei Chen and Kaiming He. 2021. Exploring Simple Siamese Representation Learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. IEEE.

[14] Danilo Dessì, Gianni Fenu, Mirko Marras, and Diego Reforgiato Recupero. 2018. Coco: Semantic-enriched collection of online courses at scale with experimental use cases. In World Conference on Information Systems and Technologies. Springer. 1386–1396.

[15] Matthias Fey, Jan E Lenssen, Frank Weichert, and Jure Leskovec. 2021. GNNAutoScale: Scalable and Expressive Graph Neural Networks via Historical Embeddings. In International Conference on Machine Learning.

[16] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the 34th Annual Conference on Neural Information Processing Systems.

[17] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In International Conference on Machine Learning. 1597–1607.

[18] Matthias Fey, Jan E Lenssen, Frank Weichert, and Jure Leskovec. 2021. GNNAutoScale: Scalable and Expressive Graph Neural Networks via Historical Embeddings. In International Conference on Machine Learning.

[19] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the 34th Annual Conference on Neural Information Processing Systems. 1597–1607.

[20] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In Proceedings of the 34th Annual Conference on Research and Development in Information Retrieval. 639–648.

[21] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 639–648.

[22] Yiottong Ji, Aixin Sun, Jie Zhang, and Chenliang Li. 2022. Recommender May Not Favor Loyal Users. arXiv preprint arXiv:2204.05927 (2022).

[23] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In International Conference on Learning Representations.

[24] 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.

[25] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. 1989. Backpropagation applied to handwritten zip code recognition. Neural computation 1, 4 (1989), 541–551.

[26] Dongha Lee, SeongKu Kang, Hyunjun Ju, Chanyoung Park, and Hwanjo Yu. 2021. Bootstrapping User and Item Representations for One-Class Collaborative Filtering. In Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval.
22 • Xin Zhou, Aixin Sun, Yong Liu, Jie Zhang, and Chunyan Miao

[29] Xiaohan Li, Mengqi Zhang, Shu Wu, Zheng Liu, Liang Wang, and Philip S Yu. 2020. Dynamic Graph Collaborative Filtering. In International Conference on Data Mining, 322–331.

[30] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. 2018. Variational autoencoders for collaborative filtering. In Proceedings of the 2018 world wide web conference. 689–698.

[31] Qidong Liu, Enguang Yao, Chaoxue Liu, Xin Zhou, Yafei Li, and Mingliang Xu. 2023. M2GCN: multi-modal graph convolutional network for modeling polypharmacy side effects. Applied Intelligence 53, 6 (2023), 6814–6825.

[32] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. 2018. Self-supervised learning: Generative or contrastive. IEEE Transactions on Knowledge and Data Engineering (2021).

[33] Xian Liu, Ming Jin, Shurui Pan, Chuan Zhou, Yu Zheng, Feng Xia, and Philip Yu. 2022. Graph self-supervised learning: A survey. IEEE Transactions on Knowledge and Data Engineering (2022).

[34] Dongsheng Luo, Wei Cheng, Wenchao Yu, Bo Zong, Jingchao Ni, Haifeng Chen, and Xiang Zhang. 2021. Learning to drop: Robust graph neural network via topological denoising. In Proceedings of the 14th ACM International Conference on Web Search and Data Mining. 779–787.

[35] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). 188–197.

[36] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence. 452–461.

[37] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2020. Droppedge: Towards deep graph convolutional networks on node classification. In International Conference on Learning Representations.

[38] Yue Shi, Martha Larson, and Alan Hanjalic. 2014. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. ACM Computing Surveys (CSUR) 47, 1 (2014), 1–45.

[39] Connor Shorten and Taghi M Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. Journal of Big Data 6, 1 (2019), 1–48.

[40] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research 15, 1 (2014), 1929–1958.

[41] Aixin Sun. 2022. From counter-intuitive observations to a fresh look at recommender system. arXiv preprint arXiv:2210.04149 (2022).

[42] Antti Tarvainen and Harri Valpola. 2017. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. Advances in neural information processing systems 30.

[43] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative deep learning for recommender systems. In Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining. 1235–1244.

[44] 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.

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

[46] Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. 2021. Self-supervised graph learning for recommendation. In Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval. 726–735.

[47] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. 2018. Unsupervised feature learning via non-parametric instance discrimination. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 3733–3742.

[48] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2020. Graph convolutional neural networks for web-scale recommender systems. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 974–983.

[49] Junliang Yu, Hongzhi Yin, Xin Xia, Tong Chen, Lihen Cui, and Nguyen Quoc Viet Hung. 2022. Are Graph Augmentations Necessary? Simple Graph Contrastive Learning for Recommendation. arXiv preprint arXiv:2112.08679 (2022).

[50] Junliang Yu, Hongzhi Yin, Xin Xia, Tong Chen, Jundong Li, and Zi Huang. 2022. Self-Supervised Learning for Recommender Systems: A Survey. arXiv preprint arXiv:2203.15876 (2022).

[51] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. 2021. Barlow twins: Self-supervised learning via redundancy reduction. arXiv preprint arXiv:2103.03230 (2021).

[52] Chaoming Zhang, Kang Zhang, Chang-Dong Yoo, and In-So Kweon. 2022. How Does SimSiam Avoid Collapse Without Negative Samples? Towards a Unified Understanding of Progress in SSL. In The International Conference on Learning Representations, ICLR 2022. The International Conference on Learning Representations (ICLR).

[53] Lingzi Zhang, Yong Liu, Xin Zhou, Chunyan Miao, Guoxin Wang, and Haibong Tang. 2022. Diffusion-based graph contrastive learning for recommendation with implicit feedback. In Database Systems for Advanced Applications: 27th International Conference, DASFAA 2022, Virtual Event, April 11–14, 2022, Proceedings, Part II. Springer, 232–247.

ACM Trans. Recomm. Syst.
We plot the performance of SelfCF varies with the number of layers in Fig. 11. The performance of Recall@20 and NDCG@20 changes with momentum, embedding dropout, edge dropout and regularization coefficient on Games.

Fig. 11. Performance of SelfCF varies with regard to the number of layers on Games.

Fig. 12. Performance of Recall@20 for various values of SelfCF with respect to the number of layers on Games.

A HYPER-PARAMETER STUDY ON GAMES

We plot the performance of SelfCF varies with the number of layers in Fig. 11. The performance of Recall@20 and NDCG@20 changes with momentum, embedding dropout, edge dropout and regularization coefficient in Fig. 12 and Fig. 13, respectively. The patterns in Games are in line with that of Food.
Fig. 13. Performance of NDCG@20 for three variations of SelfCF with respect to hyper-parameters of momentum, embedding dropout, edge dropout and regularization coefficient on Games.

Table 9. Influence of sampling methods on LightGCN evaluated with MOOC dataset.

| Model          | R@10 | R@20 | R@50 | N@10 | N@20 | N@50 |
|----------------|------|------|------|------|------|------|
| LightGCN-UniS  | 0.2507 | 0.3321 | 0.4844 | 0.1588 | 0.1835 | 0.2208 |
| LightGCN-DNS   | 0.2560 | 0.3297 | 0.4644 | 0.1644 | 0.1864 | 0.2191 |
| SelfCF_he      | 0.2545 | 0.3500 | 0.4914 | 0.1696 | 0.1986 | 0.2328 |
| SelfCF_ed      | 0.2460 | 0.3337 | 0.5088 | 0.1752 | 0.2009 | 0.2443 |
| SelfCF_ep      | 0.2514 | 0.3485 | 0.4964 | 0.1671 | 0.1963 | 0.2323 |

B PERFORMANCE OF LIGHTGCN UNDER DIFFERENT SAMPLING METHODS

In response to the research issue mentioned in Introduction, we evaluate the performance of LightGCN under both uniform sampling (UniS) and dynamic negative sampling (DNS) methods on MOOC. The results are summarized in Table 9. DNS samples hard negative user-item pairs for LightGCN, hence obtains higher ranking score (i.e., NDCG@10) when \( K \) is low in top-\( K \). However, when \( k \) increases, it is difficult to retrieve related but low ranked items for a target user. Because it always rank items based on the current representations of users and items learned by LightGCN. In many cases, the sampled negative pairs are from the test set. As a result, the recall of dynamic negative sampling on LightGCN with regard to \( K = 20 \) and \( K = 50 \) is worse than the uniform sampling method.