Meta Auxiliary Learning for Top-K Recommendation

Ximing Li, Chen Ma*, Guozheng Li, Peng Xu, Chi Harold Liu, *Senior Member, IEEE,* Ye Yuan, and Guoren Wang

Abstract—Recommender systems are playing a significant role in modern society to alleviate the information/choice overload problem, since Internet users may feel hard to identify the most favorite items or products from millions of candidates. Thanks to the recent successes in computer vision, auxiliary learning has become a powerful means to improve the performance of a target (primary) task. Even though helpful, the auxiliary learning scheme is still less explored in recommendation models. To integrate the auxiliary learning scheme, we propose a novel meta auxiliary learning framework to facilitate the recommendation model training, i.e., user and item latent representations. Specifically, we construct two self-supervised learning tasks, regarding both users and items, as auxiliary tasks to enhance the representation effectiveness of users and items. Then the auxiliary and primary tasks are further modeled as a meta learning paradigm to adaptively control the contribution of auxiliary tasks for improving the primary recommendation task. This is achieved by an implicit gradient method guaranteeing less time complexity comparing with conventional meta learning methods. Via a comparison using four real-world datasets with a number of state-of-the-art methods, we show that the proposed model outperforms the best existing models on the Top-K recommendation by 3% to 23%.

Index Terms—Recommender Systems, Auxiliary Learning, Meta Learning, Implicit Gradient.

1 Introduction

Current Internet services have enabled the convenient access of a vast number of online products and services, providing benefits for the whole society. Even though helpful and beneficial, it also brings a heavy burden for users to pick up the items that will appeal to them from a plethora of candidates, which leads to the information/choice overload. To address the information/choice overload problem, recommender systems come into being and serve a large number of demands for uses in terms of identifying the most relevant information and providing personalized services. They not only help users easily discover products that are likely to interest them, but also create opportunities for product and service providers to better serve customers and to increase revenue.

To learn a good recommendation model, the learning of the interactions between users and items lies at the core, such as the historical click and purchase records of users in the real world. Except for only modeling the user-item interaction, a number of studies have also incorporated the useful information from other sources of users or items (see Figure 1), for example, user connections [1], [2] and item relations [3], [4], to improve the recommendation or benefit the recommendation from other aspects like recommendation explainability. These studies provide guidance for how to make good use of the auxiliary information other than only relying on the interaction data.

Recently, auxiliary (task) learning [5], [6] has become a potent solution for boosting the generalization ability of a pre-defined primary task in many applications like computer vision [7] and graph representation learning [8]. The sharing of features across tasks results in additional relevant features being explored and leveraged, which otherwise would not have been learned from training only on the primary task. Auxiliary learning is much similar to multi-task learning [9] with one major difference that auxiliary tasks are only constructed to help improve the performance of the primary task.

Although the auxiliary learning scheme has demonstrated its capacity in many domains, two major challenges still remain in the personalized recommendation. First, the design of auxiliary tasks requires domain knowledge and hand-crafted engineering, and not all auxiliary tasks can provide the inductive bias to make the model further capture meaningful representations. Thus, how to design useful auxiliary tasks with little extra effort that can benefit the primary recommendation task is non-trivial. Second, different auxiliary tasks make distinct contributions to the primary recommendation task. Simply setting a fixed contribution degree as a hyper-parameter incurs the tedious hyper-parameter search, specifically when the number of auxiliary tasks is large. Furthermore, different training phases may need different contribution magnitudes of each auxiliary task. Therefore, how to adaptively balance the contribution of different auxiliary tasks and make important ones contribute more is challenging.

To tackle the aforementioned challenges, we propose a novel recommendation paradigm, namely Meta Auxiliary Learning (MAL), to successfully integrate the auxiliary

- *X. Li, G. Li, P. Xu, C. H. Liu, Y. Yuan and G. Wang are with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China. Email: {ximing_li, guozheng.li, chiliu, yuan-ye, wanggrbit}@bit.edu.cn, xupeng_mii@163.com.*
- *C. Ma is an Assistant Professor in the Department of Computer Science, City University of Hong Kong, Hong Kong SAR. Email: chenma@cityu.edu.hk.*

** C. Ma is the corresponding author.*

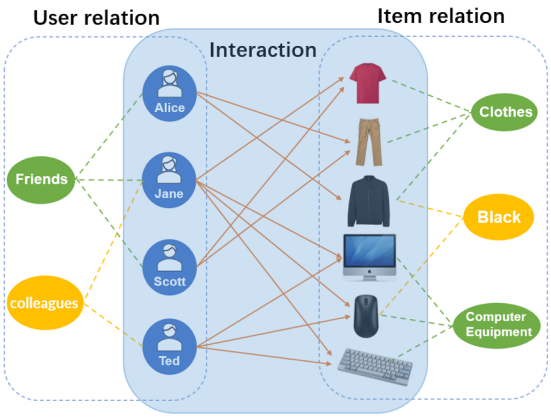


Fig. 1. The demonstration of the interactions between users and items, and relations between users/items in recommender systems.

learning scheme into the Top-K recommendation. First, we construct two auxiliary learning tasks for the explicit user-user and item-item modeling, which are borrowed from classic user-based and item-based collaborative filtering methods [10]. Specifically, we resort to the self-supervised learning methods, where we maximize the mutual information between the user or item with its similar neighbors to add extra inductive bias into the learned representations. Second, we assign additional learnable importance factors to auxiliary tasks to control their contributions to the primary recommendation task during training. In particular, these importance factors are automatically learned by the meta learning framework as the outer variable, where we adopt a one-step stochastic gradient descent trick to simplify the learning process. Third, for reducing the time complexity in importance factor updating, we adopt an implicit gradient method to explicitly construct the connection between recommendation model parameters and these importance factors, instead of using the second-order gradient updating in meta learning. We extensively evaluate our model on four real-world datasets, comparing it with many state-of-the-art methods using a variety of performance validation metrics. The experimental results not only demonstrate the improvements of our model over other baselines but also show the proposed meta auxiliary learning framework is a general training scheme that can be plugged into other recommendation models.

To summarize, the major contributions of the proposed model MAL are:

- We propose an effective recommendation framework that successfully integrates the auxiliary learning scheme.
- To control the contribution of each auxiliary task, we propose a meta learning framework to identify useful auxiliary tasks and balance their importance when training with the primary recommendation task.
- To reduce the complexity of computation, we employ an implicit gradient method to update the importance factors for auxiliary tasks, which avoids the high-other derivative computation.
- Experiments on four real-world datasets show that MAL significantly outperforms the state-of-the-art methods for the Top-K recommendation task. The implicit gradient method also demonstrates its time efficiency theoretically and empirically.

2 Related Work

In this section, we introduce the related work regarding Top-K recommendation, auxiliary learning and meta learning.

2.1 Top-K Recommendation

Early recommendation studies largely focused on explicit feedback [11], [12]. The recent research focus is shifting towards implicit data [13]. Implicit data includes clicks, visits, purchases, etc., while explicit feedback includes ratings. Collaborative filtering (CF) with implicit feedback is usually treated as a top-k item recommendation task, where the goal is to recommend a list of items to users that users may be interested in. It is more practical and challenging [14], and accords more closely with the real-world recommendation scenario. Early works mostly rely on matrix factorization techniques [15], [16] to learn latent features of users and items. Due to their ability to learn salient representations, (deep) neural network-based methods have been extensively researched. [17] proposed NeuMF, which is the concatenation of GMF layer and MLP layer, to extract user interest from implicit data. Autoencoder-based methods have also been proposed for top-k recommendation, such as [18] construct new feedback data by noise processing to make recommendation, [19] uses attention-based autoencoders to distinguish the user preference, [20] introduced neural gating mechanism for autoencoder-based recommender systems. In [21], [22], deep learning techniques are used to boost the traditional matrix factorization and factorization machine methods. Recently, graph neural networks [23] also demonstrate the effectiveness in recommender systems. Some methods use graph neural networks to model the interactions between users and items, such as [24] construct a light graph convolutional network which only includes neighborhood aggregation, [25] proposed a dual-channel hypergraph architecture to learn the embedding of users and items respectively. There are also some methods modeling the additional knowledge to assist in recommendation, such as [2] constructing user-user and item-item graphs to integrate the proximal information.

2.2 Auxiliary Learning

Auxiliary learning is similar to multi-task learning in certain aspects, but unlike multi-task learning, auxiliary learning only focuses on the performance of the primary task. Recently, auxiliary learning has been an effective solution to improve the performance of a primary task in many fields. For example, [26] apply auxiliary supervision by learning intermediate low-level representations to benefit the recognition of conversational speeches. [27] choose auxiliary tasks to enhance the single scene depth estimation and semantic segmentation. [28] propose a sentiment analysis model, which is constructed by two auxiliary tasks to learn the sentence embedding. By carefully designing the learning tasks, auxiliary learning can also be used in a unsupervised fashion without ground-truth labels. [5] improve the agent learning in Atari games by predicting the immediate

rewards from a short historical context, which are unsupervised auxiliary tasks. [29] perform unsupervised monocular depth estimation to predict the relative pose of multiple cameras. [30] propose to use the cosine similarity as an adaptive task weighting to determine which auxiliary task is more useful. Also, auxiliary learning has achieved a huge progress in the field of self-supervised learning. For example, [31] build a self-supervised auxiliary reconstruction task which shares a part of the network with the primary task, using a joint training scheme to enhance the performance of image deblurring. Recently, the auxiliary learning is also incorporated in graph representation learning by selecting meaningful meta-paths [8]. Several recent methods [32], [33] also apply the meta-learning or bilevel optimization to identify important auxiliary tasks. Auxiliary learning has also been explored in the recommender systems recently. [34] introduce a punitive auxiliary task to helps the model promote samples with high Conversion Rate (CVR) but low Click-Through Rate (CTR). [35] proposes to adjust the gradient of the auxiliary learning dynamically, and through this method, models can avoid a serious optimization imbalance problems.

2.3 Meta Learning

Meta learning, also known as learning to learn, is a kind of approaches focusing on the learning process itself, and tries to generalize the learning strategy on new tasks. In recent years, meta learning has developed rapidly and has been applied suitably in many fields. Early works explore the use of meta learning to automatically update the rules of neural networks [36]. Recent methods try to customize the optimizers by meta learning, which are based on LSTMs [37] or synthetic gradients [38]. There are also some models that use meta learning to solve special data sampling problems, such as [39] using meta learning to construct a weight function for sample-reweighting. Meta learning has also been studied for finding optimal hyper-parameters [40] and a good initialisation for few-shot learning [41]. [42] investigate few-shot learning via an external memory module. [43] realise the few-shot learning in the instance space via a differentiable nearest-neighbour approach. Recently, graph neural networks also introduce meta learning to perform specific tasks. For example, [44] structures the pre-training step to simulate the fine-tuning process on downstream tasks, so as to directly optimize the pre-trained model's quick adaptability to downstream tasks.

Different from the above three categories of methods, we distinguish our model by the first to integrate the auxiliary learning scheme in the recommendation model. Furthermore, two self-supervised learning tasks are carefully designed as the auxiliary task to enhance the primary recommendation task. To control the contribution of the auxiliary task, we propose a fine-grained weighting scheme optimized by a meta-learning framework using the implicit gradient method.

3 Problem Formulation

For the recommendation model proposed in this paper, the input data is the implicit feedback between users and items, such as clicks, purchases, or visits. For each user u, the items that this user has accessed are denoted by an item set $\mathcal{D}_u = \{I_1, ..., I_j, ..., I_{|\mathcal{D}_u|}\}$, where I_j is an item index in the dataset. Top-K recommendation aims to provide each user with a set of K most relevant items from whole items in the dataset, which can be formulated as: for each user i, given the training set $\mathcal{S}_u$ and a non-empty test set $\mathcal{T}_u$ (satisfy $\mathcal{S}_u \cup \mathcal{T}_u = \mathcal{D}_u$ and $\mathcal{S}_u \cap \mathcal{T}_u = \emptyset$), the recommender system should return an ordered set of items $\mathcal{X}_u$ such that $|\mathcal{X}_u| \leq K$ and $\mathcal{S}_u \cap \mathcal{X}_u = \emptyset$. Then the recommendation quality is measured by Recall@K and NDCG@K via comparing the items in $\mathcal{T}_u$ with which in $\mathcal{X}_u$. A summary of notations is shown in Table 1.

TABLE 1
A summary of notations.

u, i	user u, item i
$\mathbf{p}_u$	latent embedding of user u
$\mathbf{q}_i$	latent embedding of item i
d	dimension of the embedding
$\mathbf{\Theta}$	learnable parameters in primary task
$\mathcal{N}_u^U$	neighbor set of user u
$\mathcal{N}_i^I$	neighbor set of item i
u^+, u^-	positive and negative samples of user u
i^+, i^-	positive and negative samples of item i
λ_u^U	importance factor for user-related auxiliary tasks
λ_i^I	importance factor for item-related auxiliary tasks
$\mathbf{\Lambda}$	include λ_u^U and λ_i^I
k	a condition number to compute $\mathbf{\Theta}^*(\mathbf{\Lambda})$
D	the diameter of search space for getting a proper $\mathbf{\Theta}$

4 Methodology

In this section, we introduce the proposed framework shown in Figure 2, which integrates the auxiliary learning in the recommendation task. We first introduce the primary task (recommendation) in our model. Then we illustrate the auxiliary task which consists of two main components: 1) designing auxiliary tasks supporting the learning of the recommendation task, and 2) learning personalized parameters to softly select auxiliary tasks and balance them via meta learning. Lastly, we introduce the prediction and training procedure of the proposed model, as well as the theoretical analysis of the implicit gradient method.

4.1 Primary Task

The primary task in the proposed model is the user preference modeling, which lies at the core of all recommender systems. Based on the user-item interaction history and probably other available metadata, the recommendation model learns/constructs a scoring function to predict the user preference levels of user u on a certain item i:

$$\hat{r}_{ui} = f(u, i; \mathbf{\Theta}) \,, \tag{1}$$

where $\mathbf{\Theta}$ is the learnable parameters in the primary task. Popularized by the Netflix competition, the matrix factorization model [15], [45] is a classical and effective realization:

$$f(u, i; \mathbf{\Theta}) = \mathbf{q}_i^\top \cdot \mathbf{p}_u \,, \tag{2}$$

where $\mathbf{p}_u \in \mathbb{R}^d$ is the latent representation (embedding) of user u, $\mathbf{q}_i \in \mathbb{R}^d$ is the latent representation of item i, d is

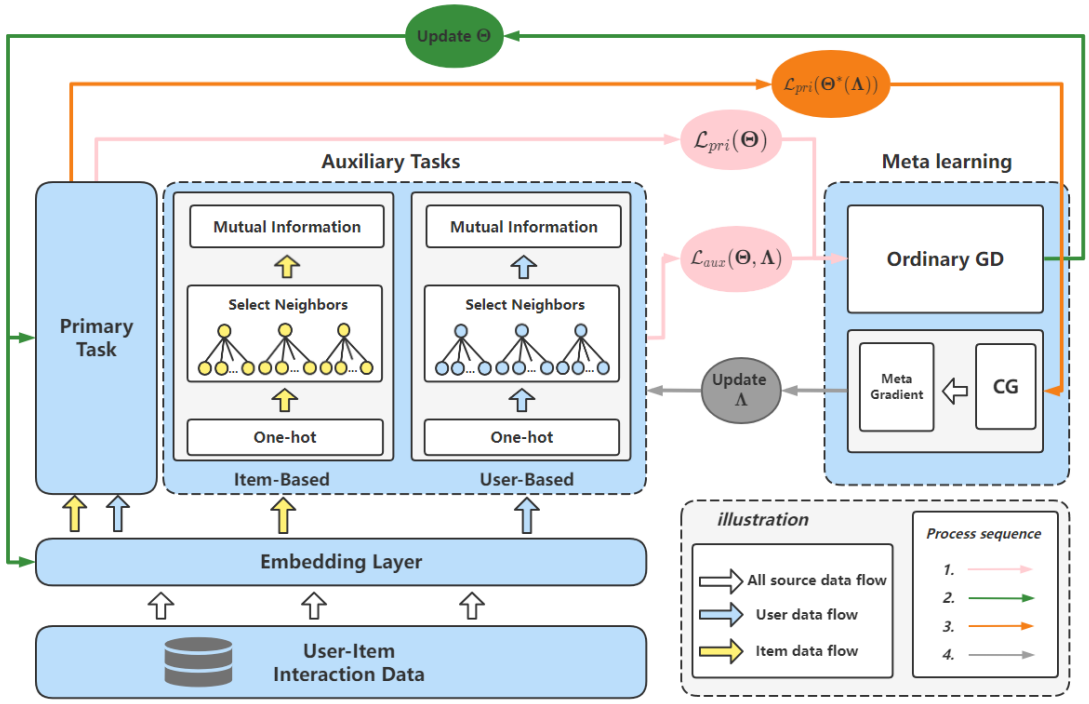


Fig. 2. Overview of our proposed framework of MAL. It consists of one primary recommendation task (any gradient-based recommendation model can be plugged in), two auxiliary tasks, and the meta learning framework. In each training update, **firstly**, the embeddings regarding specific user-item interactions are specified, and are used to calculate the loss $\mathcal{L}_{pri+aux}(\mathbf{\Theta}, \mathbf{\Lambda})$ through the primary task and two well-designed auxiliary tasks. **Secondly**, the loss $\mathcal{L}_{pri+aux}(\mathbf{\Theta}, \mathbf{\Lambda})$ passes through the inner stage of the meta learning framework to update $\mathbf{\Theta}$. **Thirdly**, building the connection between the outer loss $\mathcal{L}_{pri}(\mathbf{\Theta}^*(\mathbf{\Lambda}))$ and $\mathbf{\Lambda}$ by using a proxy $\mathbf{\Theta}^*(\mathbf{\Lambda})$. **Finally**, the loss $\mathcal{L}_{pri}(\mathbf{\Theta}^*(\mathbf{\Lambda}))$ passes through the outer stage of the meta learning framework and uses the conjugate gradient (CG) algorithm to obtain the meta-gradient, then update $\mathbf{\Lambda}$. *GD* denotes the gradient descent.

the dimension of the latent space, and $\cdot$ denotes the inner product. The user preference on a certain item is measured by the inner product between the latent representations of users and items. Here, for the ease of illustration, we only adopt the matrix factorization model as an example. Indeed, other recommendation models like deep learning-based methods, e.g., NeuMF [17] and LightGCN [24], can be easily plugged into our framework (shown in Section 5).

To learn the user preference, one promising and widely-used approach is the Bayesian Personalized Ranking (BPR) [16], which is used to model the pair-wise preference between an item i that user u has interacted with and a randomly sampled item i' that user u has not accessed. BPR assumes that interacted items, which are more reflective of a user's preferences, should be assigned with higher preference scores than unobserved ones. The objective function of BPR is shown as follows:

$$\mathcal{L}_{pri}(u, i, i'; \mathbf{\Theta}) = -\ln \sigma(\hat{r}_{ui} - \hat{r}_{ui'}) , \tag{3}$$

where σ is the sigmoid function, and $\mathbf{\Theta}$ represents the learnable parameters, e.g., $\mathbf{p}_u$ and $\mathbf{q}_i$ in the matrix factorization, of the primary task.

4.2 Auxiliary Task

Auxiliary learning [8], [32] is a promising approach to benefit the primary task, by training on additional auxiliary tasks. The sharing of features across tasks brings additional salient latent features, which can be hardly obtained from solely training on the primary task. Thus the incorporation of auxiliary tasks is a potential direction to improve the recommendation performance and bring other potential benefits such as recommendation explainability. Then here come two central questions: (i) what auxiliary tasks to construct and (ii) how to learn these auxiliary tasks to benefit the primary task.

Auxiliary Task Construction. To construct auxiliary tasks in recommendation models, we resort to the user-user and item-item relation learning, which complements the user-item interaction learning. In fact, other useful auxiliary tasks can also be added but may need further feature engineering and domain knowledge [8]. For simplicity, we borrow the idea from user-based and item-based collaborative filtering methods [10] to construct the similarity matrix of users and items, respectively. Formally, given an observed user-item interaction matrix $\mathbf{R} \in \mathbb{R}^{m \times n}$ (m users and n items), we calculate the cosine similarity between each row (user) and column (item), respectively, to get the pair-wise similarity among users and items. Then we set a threshold τ to select the high-value neighbors of users and items into neighbor sets $\mathcal{N}_u^U$ and $\mathcal{N}_i^I$ of user u and item i, respectively. These high-value neighbors of users and items are utilized to model the intuition that similar users or items would share similar interests or properties.

Auxiliary Task Learning. To learn from the high-value neighbors of users and items, we employ the self-supervised learning scheme by maximizing the mutual information (MI) [46] between a user/item and its neighbors to improve the representation effectiveness of users and items. Specifically, MI measures the amount of information obtained about a random variable X by observing some other

random variable Y. Formally, the MI between X and Y is defined as:

$$\begin{aligned} I(X;Y) &= D_{KL}\big(p(x,y) \parallel p(x)p(y)\big) \\ &= \mathbb{E}_{p(x,y)}\left[\log \frac{p(x,y)}{p(x)p(y)}\right], \end{aligned} \tag{4}$$

where $p(x,y)$ denotes the joint density, $p(x)$ and $p(y)$ denote marginal densities, and D_{KL} denotes the Kullback-Leibler (KL) divergence. However, directly maximizing the above mutual information formulation is generally intractable [47], [48] especially when involving complex deep learning models. Thus we turn to optimize the lower bound of MI. One particular lower bound that has been shown working well in practice is InfoNCE [49], which is based on the Noise Contrastive Estimation [50]. Specifically, the InfoNCE loss is formulated by:

$$\begin{aligned} \mathcal{L}^U_{aux}(u;\mathbf{\Theta}) &= -\log\left(\frac{\mathrm{e}^{s(u,u^+)}}{\sum_{u^-\notin\mathcal{N}^U_u}\mathrm{e}^{s(u,u^-)}}\right), \\ \mathcal{L}^I_{aux}(i;\mathbf{\Theta}) &= -\log\left(\frac{\mathrm{e}^{s(i,i^+)}}{\sum_{i^-\notin\mathcal{N}^I_i}\mathrm{e}^{s(i,i^-)}}\right), \end{aligned} \tag{5}$$

where u^+ is a similar neighbor of user u while u^- is randomly sampled and treated as negative samples. $s(u,u^+) = \mathbf{p}_{u^+}^\top \cdot \mathbf{p}_u$ is a function to measure the agreement score between u and u^+. The item side also follows the same manner. Intuitively, the optimization of InfoNCE loss aims to assign higher agreement scores between the user or item and its neighbors, whereas assigning lower scores on the negative samples. Especially, when using only one negative sample in the loss, the InfoNCE loss is equivalent to the Bayesian Personalized Ranking (BPR) loss [51]. Then our auxiliary tasks together with the primary recommendation task can be formulated by:

$$\mathcal{L}_{pri}(u,i,i';\mathbf{\Theta}) + \lambda^U \cdot \mathcal{L}^U_{aux}(u;\mathbf{\Theta}) + \lambda^I \cdot \mathcal{L}^I_{aux}(i;\mathbf{\Theta}), \tag{6}$$

where λ^U and λ^I are used to control the contribution of the auxiliary tasks.

4.3 Meta Auxiliary Learning

Although incorporating λ^U and λ^I along with auxiliary tasks has already boosted the primary recommendation task, it still yields two disadvantages. First, different users or items may need different levels of auxiliary learning. Thus adopting the same value for all users or items may not release the full potential of the auxiliary learning mechanism. Second, treating λ^U and λ^I as hyper-parameters will trigger tedious hyper-parameter tuning. When the number of auxiliary tasks further increases, manually tuning these hyper-parameters are almost unacceptable.

To tackle the aforementioned two problems, we first model the contribution of auxiliary learning into a more fine-grained level, i.e., modeling by the specific user and item:

$$\mathcal{L}_{pri}(u,i,i';\mathbf{\Theta}) + \sigma(\lambda^U_u)\cdot\mathcal{L}^U_{aux}(u;\mathbf{\Theta}) + \sigma(\lambda^I_i)\cdot\mathcal{L}^I_{aux}(i;\mathbf{\Theta}), \tag{7}$$

where λ^U_u and λ^I_i represent the contribution magnitude of the auxiliary learning for each user and item, respectively.

Next, we formulate the learning of the primary recommendation task along with auxiliary tasks as a meta-learning problem, to adaptively adjust the importance of finer levels of auxiliary learning:

$$\begin{aligned} &\min_{\mathbf{\Lambda}} \sum_{(u,i,i')\in\mathcal{D}^{meta}} \mathcal{L}_{pri}\big(u,i,i';\mathbf{\Theta}^*(\mathbf{\Lambda})\big) \\ &\text{s.t. } \mathbf{\Theta}^*(\mathbf{\Lambda}) = \operatorname*{argmin}_{\mathbf{\Theta}} \sum_{(u,i,i')\in\mathcal{D}^{train}} \mathcal{L}_{pri}(u,i,i';\mathbf{\Theta}) + \\ &\sigma(\lambda^U_u)\cdot\mathcal{L}^U_{aux}(u;\mathbf{\Theta}) + \sigma(\lambda^I_i)\cdot\mathcal{L}^I_{aux}(i;\mathbf{\Theta}), \end{aligned} \tag{8}$$

where $\mathbf{\Lambda}$ includes λ^U_u and λ^I_i, $\mathcal{D}^{train}$ and $\mathcal{D}^{meta}$ are constructed training data and meta data, respectively, and $\sigma(\cdot)$ is the sigmoid function to make sure the value is within $(0,1)$. Following [52], we randomly sample user-item interactions to form $\mathcal{D}^{meta}$. For the ease of illustration, we simplify the formulation of Eq. 8 without loss of generality as:

$$\begin{aligned} &\min_{\mathbf{\Lambda}} \mathcal{L}_{pri}(\,\cdot\,;\mathbf{\Theta}^*(\mathbf{\Lambda})) \\ &\text{s.t. } \mathbf{\Theta}^*(\mathbf{\Lambda}) = \operatorname*{argmin}_{\mathbf{\Theta}} \mathcal{L}_{pri+aux}(\,\cdot\,;\mathbf{\Theta},\mathbf{\Lambda}). \end{aligned} \tag{9}$$

To optimize the above loss function, we adopt the one-step stochastic gradient descent (SGD) trick to build the proxy between $\mathcal{L}_{pri}$ (Eq. 9) and $\mathbf{\Lambda}$:

$$\mathbf{\Theta}^*(\mathbf{\Lambda}) \approx \tilde{\mathbf{\Theta}}(\mathbf{\Lambda}) = \mathbf{\Theta} - \alpha\nabla_{\mathbf{\Theta}}\mathcal{L}_{pri+aux}(\mathbf{\Theta},\mathbf{\Lambda}), \tag{10}$$

where α is the learning rate. Note that we do not numerically evaluate compute $\tilde{\mathbf{\Theta}}(\mathbf{\Lambda})$. Instead, we resort to the computation graph of deep learning libraries (e.g., PyTorch), where we treat $\tilde{\mathbf{\Theta}}(\mathbf{\Lambda})$ as an intermediary node to connect $\mathcal{L}_{pri}$ and $\mathbf{\Lambda}$. Then by minimizing $\mathcal{L}_{pri}$, the gradient can be passed to $\mathbf{\Lambda}$. Thus we can alternatively update $\mathbf{\Theta}$ or $\mathbf{\Lambda}$ while keep the other fixed.

4.4 Implicit Gradient Method

Although the bilevel optimization learning methods in Section 4.3 can adaptively control the contribution of each auxiliary task, it brings computation overheads. The reason is that, when using the one-step SGD trick to update $\mathbf{\Lambda}$, the meta learning process requires calculating the second-order derivative (a Hessian matrix), i.e., $\frac{\partial\mathcal{L}_{pri}(\tilde{\mathbf{\Theta}}^*(\mathbf{\Lambda}))}{\partial\mathbf{\Lambda}}$, which makes the update of $\mathbf{\Lambda}$ yield a longer gradient dependency path and involve more computation. To overcome this overhead, we borrow the idea from implicit gradient method [53] to avoid the second-order derivative.

According to [53], the dependency between $\mathbf{\Theta}$ and $\mathbf{\Lambda}$ shrinks and vanishes as the number of gradient steps grows when updating $\mathbf{\Lambda}$, making meta learning difficult to train. To prevent this from happening, we add a soft L2 norm between $\mathbf{\Theta}$ and $\mathbf{\Lambda}$ in $\mathcal{L}_{pri+aux}(\mathbf{\Theta},\mathbf{\Lambda})$, which can be expressed as:

$$\mathcal{L}_{pri+aux}(\mathbf{\Theta},\mathbf{\Lambda}) \leftarrow \mathcal{L}_{pri+aux}(\mathbf{\Theta},\mathbf{\Lambda}) + \frac{\gamma}{2}\left\|\mathbf{\Theta}-\mathbf{\Lambda}\right\|^2, \tag{11}$$

where γ is a hyper-parameter to control the regularization strength. To make $\mathbf{\Theta}$ and $\mathbf{\Lambda}$ have the same size, we multiply

$\mathbf{\Lambda}$ with a trainable matrix and denoted the result as $\mathbf{\Lambda}$. Thus, from Eq. 9, we can derive:

$$\begin{aligned} &\mathbf{\Theta}^*(\mathbf{\Lambda}) = \underset{\mathbf{\Theta}}{\operatorname{argmin}}\, \mathcal{L}_{pri+aux}(\mathbf{\Theta}, \mathbf{\Lambda}) \\ \Longrightarrow &\nabla \mathcal{L}_{pri+aux}(\mathbf{\Theta}^*(\mathbf{\Lambda})) + \gamma(\mathbf{\Theta}^*(\mathbf{\Lambda}) - \mathbf{\Lambda}) = 0 \\ \Longrightarrow &\mathbf{\Theta}^*(\mathbf{\Lambda}) = \mathbf{\Lambda} - \frac{1}{\gamma}\nabla \mathcal{L}_{pri+aux}(\mathbf{\Theta}^*(\mathbf{\Lambda})) \end{aligned} \tag{12}$$

When the derivative exists, we can differentiate Eq. 12 to obtain:

$$\begin{aligned} &\frac{\partial \mathbf{\Theta}^*(\mathbf{\Lambda})}{\partial \mathbf{\Lambda}} = \boldsymbol{I} - \frac{1}{\gamma}\nabla^2 \mathcal{L}_{pri+aux}(\mathbf{\Theta}^*(\mathbf{\Lambda}))\frac{\partial \mathbf{\Theta}^*(\mathbf{\Lambda})}{\partial \mathbf{\Lambda}} \\ \Longrightarrow &\frac{\partial \mathbf{\Theta}^*(\mathbf{\Lambda})}{\partial \mathbf{\Lambda}} = \left(\boldsymbol{I} + \frac{1}{\gamma}\nabla^2 \mathcal{L}_{pri+aux}(\mathbf{\Theta}^*(\mathbf{\Lambda}))\right)^{-1} \end{aligned} \tag{13}$$

Therefore, the Hessian matrix $\frac{\partial \mathcal{L}_{pri}(\mathbf{\Theta}^*(\mathbf{\Lambda}))}{\partial \mathbf{\Lambda}}$ can be reformulated as:

$$\begin{aligned} \frac{\partial \mathcal{L}_{pri}(\mathbf{\Theta}^*(\mathbf{\Lambda}))}{\partial \mathbf{\Lambda}} = &\left(\boldsymbol{I} + \frac{1}{\gamma}\nabla^2 \mathcal{L}_{pri+aux}(\mathbf{\Theta}^*(\mathbf{\Lambda})\right)^{-1} \cdot \\ &\nabla \mathcal{L}_{pri}(\mathbf{\Theta}^*(\mathbf{\Lambda}))\,, \end{aligned} \tag{14}$$

However, the explicit form and the inverse of the matrix in Eq. 14 to compute the Jacobian may be intractable in large deep neural networks. Instead, we solve a quadratic optimization problem of Eq. 15 and use an approximate as the high-order gradient:

$$\min_{\omega} \frac{1}{2}\omega^T \left(\boldsymbol{I} + \frac{1}{\gamma}\nabla^2 \mathcal{L}_{pri+aux}(\mathbf{\Theta}^*(\mathbf{\Lambda}))\right)\omega - \omega^T \nabla \mathcal{L}_{pri}(\mathbf{\Theta}^*(\mathbf{\Lambda})) \tag{15}$$

Then we can use the conjugate gradient (CG) algorithm to approximate this optimization problem. Compared to computing the high-order derivative directly, the CG algorithm has an excellent iterative convergence efficiency, which is conducive to computing the approximate values. Thus, when we fix $\mathbf{\Theta}$ to update $\mathbf{\Lambda}$, we use the CG algorithm to reduce the computational burden.

4.5 Model Training and Prediction

By adding the regularization term in Eq. 9, we have the overall loss function:

$$\begin{aligned} &\min_{\mathbf{\Lambda}} \mathcal{L}_{pri}(\,\cdot\,; \mathbf{\Theta}^*(\mathbf{\Lambda})) \\ &\text{s.t. } \mathbf{\Theta}^*(\mathbf{\Lambda}) = \underset{\mathbf{\Theta}}{\operatorname{argmin}}\, \mathcal{L}_{pri+aux}(\,\cdot\,; \mathbf{\Theta}, \mathbf{\Lambda}) + \frac{\gamma}{2}\left\|\mathbf{\Theta} - \mathbf{\Lambda}\right\|^2\,, \end{aligned} \tag{16}$$

where γ is a hyper-parameter to control the effect of regularization. We apply Adam [54] optimizer to update $\mathbf{\Theta}$, and use implicit gradient algorithm mentioned in Section 4.4 to speed up the optimization of $\mathbf{\Lambda}$. The detailed training algorithm is shown in Algorithm 1 and Algorithm 2.

Recommendation. For each user u, we compute the preference score $\hat{r}_{ui}$ on each item i. Then the Top-K items with larger prediction scores meanwhile not in the training set will be recommended to user u.

Algorithm 1: Training algorithm of MAL

Input : Training data $\mathcal{D}^{train}$ and meta data $\mathcal{D}^{meta}$, regularization parameter γ, accuracy thresholds δ and $\delta^{'}$.
Initialize: parameters $\mathbf{\Theta}$, $\mathbf{\Lambda}$
while *not converged* **do**
  $\mathcal{D}_m^{train}$ = BatchSampler($\mathcal{D}^{train}$);
  $\mathcal{D}_m^{meta}$ = BatchSampler($\mathcal{D}^{meta}$);
  Fix $\mathbf{\Lambda}$ update $\mathbf{\Theta}$ with $\mathcal{D}_m^{train}$:
    $\mathbf{\Theta} \leftarrow OPT_{\mathbf{\Theta}}\left(\mathbf{\Theta}, \nabla_{\mathbf{\Theta}} \mathcal{L}_{pri+aux}(\mathbf{\Theta}, \mathbf{\Lambda})\right)$;
  Obtain $\mathbf{\Theta}^*(\mathbf{\Lambda})$ with an iterative optimizer:
    $\|\mathbf{\Theta}^*(\mathbf{\Lambda}) - \underset{\mathbf{\Theta}}{\operatorname{argmin}} \mathcal{L}_{pri+aux}(\,\cdot\,; \mathbf{\Theta}, \mathbf{\Lambda})\| \leq \delta$;
  Compute the approximation of $\nabla_{\mathbf{\Lambda}} \mathcal{L}_{pri}$:
    $\nabla_{\mathbf{\Lambda}} \hat{\mathcal{L}}_{pri}$= Implicit-gradient($\mathbf{\Theta}^*(\mathbf{\Lambda})$, γ, $\delta^{'}$) ;
  Fix $\mathbf{\Theta}$ update $\mathbf{\Lambda}$ with $\mathcal{D}_m^{meta}$:
    $\mathbf{\Lambda} \leftarrow OPT_{\mathbf{\Lambda}}\left(\mathbf{\Lambda}, \nabla_{\mathbf{\Lambda}} \hat{\mathcal{L}}_{pri}\right)$;
end

Algorithm 2: Implicit gradient optimization

Input : Updated $\mathbf{\Theta}^*(\mathbf{\Lambda})$, regularization parameter γ, optimization threshold $\delta^{'}$.
Compute loss of primary task: $l = \mathcal{L}_{pri}\left(\mathbf{\Theta}^*(\mathbf{\Lambda})\right)$;
Compute the outer-level gradient: $v = \nabla_{\mathbf{\Theta}} l$;
Use CG algorithm to compute g such that:
  $\|g - \left(I + \frac{1}{\gamma}\nabla^2 \mathcal{L}_{pri+aux}(\mathbf{\Theta}^*(\mathbf{\Lambda}))\right)^{-1} v\| \leq \delta^{'}$;
Return: g

4.6 Theoretical Analysis

In Section 4.4, we adopt an implicit gradient optimization method to update $\mathbf{\Lambda}$. Here, we compare the implicit gradient method (Section 4.3) with the second-order gradient method (Section 4.4), and provide a theoretical analysis of the reduction in terms of time complexity brought by this implicit gradient method.

To discuss the time complexity for computing the meta-gradient ($\frac{\partial \mathcal{L}_{pri}(\mathbf{\Theta}^*(\mathbf{\Lambda}))}{\partial \mathbf{\Lambda}}$) when applying a $\delta-$accurate inner optimization, we first assume that $\mathcal{L}_{pri+aux}(\,\cdot\,; \mathbf{\Theta}, \mathbf{\Lambda})$ in the inner optimization is a differentiable convex function. We then assume that the condition number to compute $\mathbf{\Theta}^*(\mathbf{\Lambda})$ is a constant number k, which can be viewed as a measure of the hardness to update $\mathbf{\Theta}^*(\mathbf{\Lambda})$. But when we use Eq. 15 to get the approximate solution of the high-order gradient ω^T, the condition number will only become $\sqrt{k}$. This is because when using the implicit gradient method to perform meta learning, we can turn the high-order gradient into a quadratic optimization problem, which makes it unnecessary to back-propagate gradients to the inner optimization. Furthermore, we assume the diameter of the search space for getting a proper $\mathbf{\Theta}$ is D. Then the computational complexity for finding a proper $\mathbf{\Theta}$ of each iteration is $\log\left(\frac{D}{\delta}\right)$, where δ is the accuracy threshold used to control the tolerance of error.

Although this implicit gradient optimization method can greatly reduce the computational complexity, it will incur the bias for the meta gradient because it obtains an approximation of $\mathbf{\Theta}^*(\mathbf{\Lambda})$. By introducing an approximate

threshold δ in Algorithm 1, it computes an approximate $\mathbf{\Theta}^*(\mathbf{\Lambda})$ with the following guarantee:

$$\|\mathbf{\Theta}^*(\mathbf{\Lambda}) - \underset{\mathbf{\Theta}}{\operatorname{argmin}} \mathcal{L}_{pri+aux}(\,\cdot\,; \mathbf{\Theta}, \mathbf{\Lambda})\| \leq \delta\,. \tag{17}$$

Therefore, when the implicit gradient method (Section 4.4) is used to update $\mathbf{\Lambda}$, the computational complexity will reduce from $k \log(\frac{D}{\delta})$ to $\sqrt{k} \log(\frac{D}{\delta})$ compared to the second-order gradient method (Section 4.3). Correspondingly, it will bring δ error when calculating $\mathbf{\Theta}^*(\mathbf{\Lambda})$.

5 Evaluation

In this section, we first describe the experimental setup. We then report the results of the conducted experiments and demonstrate the effectiveness of the proposed modules.

5.1 Datasets

The proposed model is evaluated on four real-world datasets from various domains with different sparsities: *Amazon-Books*, *Amazon-CDs*, *Gowalla* and *MovieLens-20M*. The *Amazon-Books* and *Amazon-CDs* datasets are adopted from the Amazon review dataset with different categories, i.e., CDs and Books, which cover a large amount of user-item interaction data, e.g., user ratings and reviews. The *Gowalla* is a point-of-interest check-in dataset obtained from *Gowalla* website. *MovieLens-20M* is a user-movie dataset collected from the *MovieLens* website, which has 20 million user-movie interactions. All the above datasets follow the 10-core setting to ensure that each user and item have at least ten interactions. The data statistics after preprocessing are shown in Table 2.

For each dataset, 80% of interaction data of each user is randomly selected to constitute the training set, and we treat the remaining 20% as the test set. From the training set, 10% of interactions are randomly selected as the validation set to tune hyper-parameters. The experiments of all the models are executed five times and the average result is reported.

TABLE 2
The statistics of the datasets.

Dataset	#Users	#Items	#Interactions	Density
CDs	24,934	24,634	478,048	0.078%
Gowalla	29,858	40,981	1,027,370	0.084%
Books	77,754	66,963	2,517,343	0.048%
ML20M	129,780	13,663	9,926,480	0.560%

5.2 Evaluation Metrics

We evaluate all the methods in terms of *Recall@K*, *NDCG@K* and *Precision@K*. For each user, Recall@K (R@K) indicates what percentage of her rated items emerge in the top *K* recommended items. NDCG@K (N@K) is the normalized discounted cumulative gain at *K*, which takes the position of correctly recommended items into account. Precision@K (P@K) indicates what percentage of ground-truth items in the top *K* recommended items.

They are:

$$\begin{aligned}
\text{Recall@K} &= \frac{1}{M}\sum_{i=1}^{M}\frac{\mathcal{X}_i(K)\cap\mathcal{T}_i}{|\mathcal{T}_i|}\,,\\
\text{NDCG@K} &= \frac{1}{M}\sum_{i=1}^{M}\frac{\sum_{j\in\mathcal{T}_i}\frac{1}{\log_2(\text{rank}(j)+1)}}{\text{IDCG}\left(\min(K,|\mathcal{T}_i|)\right)}\,,\\
\text{Precision@K} &= \frac{1}{M}\sum_{i=1}^{M}\frac{\mathcal{X}_i(K)\cap\mathcal{T}_i}{|\mathcal{X}_i(K)|}\,,
\end{aligned} \tag{18}$$

where $\text{IDCG}(K) = \sum_{j=1}^{K}\frac{1}{\log_2(j+1)}$, M is the number of users, $\mathcal{X}_i(K)$ is a ordered set of top-k recommended items for user i excluding the items in the training set, and $\mathcal{T}_i$ is a set of ground-truth items to be evaluated. $rank(j)$ is the position of item j in the recommendation set $\mathcal{X}_i(K)$.

5.3 Methods Studied

To demonstrate the effectiveness of our framework, we compare to the following recommendation methods:

- **MF** [16]: a matrix factorization (MF) model optimized by the Bayesian personalized ranking (BPR) loss, which is a classical method for learning pair-wise item rankings.
- **NeuMF** [17]: a typical recommendation algorithm based on deep learning, which combines the traditional matrix factorization and multi-layer perceptrons.
- **CMN** [55]: a classic memory-based model, which utilizes the advantages of the global structure of the latent factor model and the structure based on local neighborhoods.
- **GC-MC** [56]: a model designed to employ a graph convolutional network on graph-structured data.
- **Mult-VAE** [57]: an item-based CF method based on the variational autoencoder (VAE).
- **NGCF** [58]: capturing the user-item interactions with a bipartite graph, and exploiting the user-item graph structure by propagating embeddings.
- **RecVAE** [59]: a VAE-based model which introduces a mixture of Gaussian priors and the latent code distribution in the VAE, and an alternating update method in the encoder and decoder.
- **LightGCN** [24]: learning user and item embeddings by linearly propagation on the user-item interaction graph, and using the weighted sum of the embeddings learned at all layers as the final embedding.
- **SinkhornCF** [60]: a method which is based on the Sinkhorn divergence, achieving recommendation by considering the item-similarity.
- **DCF** [61]: a model which uses factor-level attention to capture the fine-grained representations of the implicit embedding and employs a relation aggregator to learn the explicit embedding.
- **SGL** [62]: a graph-based model which generates subgraphs through node dropout, edge dropout and random walk, and constructs self-supervised contrastive learning tasks on different subgraphs.
- **IMP-GCN** [63]: a model aims to perform the graph convolution network (GCN) on subgraphs which consist of users with similar interests and their interacted items.

TABLE 3
The performance comparison of all methods in terms of *Recall@K*. We present the relative improvements of our framework over the base model in bold. *Imp.* denotes the improvement.

Methods	CDs				Books				Gowalla				ML20m			
	R@5	R@10	R@15	R@20	R@5	R@10	R@15	R@20	R@5	R@10	R@15	R@20	R@5	R@10	R@15	R@20
NeuMF	0.0433	0.0702	0.0906	0.1087	0.0275	0.0468	0.0628	0.0778	0.0689	0.0961	0.1199	0.1348	0.1075	0.1703	0.2196	0.2596
CMN	0.0445	0.0711	0.0919	0.1094	0.0279	0.0472	0.0638	0.0785	0.0701	0.0978	0.1213	0.1359	0.1098	0.1742	0.2238	0.2647
GC-MC	0.0439	0.0708	0.0913	0.1090	0.0281	0.0475	0.0641	0.0788	0.0684	0.0954	0.1191	0.1342	0.1084	0.1728	0.2216	0.2621
SinkhornCF	0.0503	0.0786	0.1029	0.1208	0.0376	0.0622	0.0814	0.0968	0.0754	0.1091	0.1342	0.1558	0.1187	0.1886	0.2412	0.2868
MF	0.0453	0.0724	0.0935	0.1117	0.0286	0.0483	0.0649	0.0792	0.0728	0.1043	0.1274	0.1472	0.1106	0.1762	0.2263	0.2681
MF-MAL	0.0561	0.0885	0.1133	0.1320	0.0388	0.0651	0.0857	0.1038	0.0773	0.1117	0.1374	0.1598	0.1217	0.1938	0.2481	0.2954
Imp.	**23.84%**	**22.24%**	**21.18%**	**18.17%**	**35.66%**	**34.78%**	**32.05%**	**31.06%**	**6.18%**	**7.09%**	**7.85%**	**8.56%**	**10.04%**	**9.99%**	**9.63%**	**10.18%**
MultVAE	0.0507	0.0791	0.1035	0.1218	0.0375	0.0623	0.0812	0.0966	0.0759	0.1096	0.1347	0.1564	0.1184	0.1882	0.2409	0.2861
MultVAE-MAL	0.0583	0.0921	0.1178	0.1354	0.0454	0.0787	0.1018	0.1184	0.0818	0.1150	0.1402	0.1660	0.1261	0.2008	0.2558	0.3027
Imp.	**14.99%**	**16.43%**	**13.82%**	**11.17%**	**21.07%**	**26.32%**	**25.37%**	**22.57%**	**7.77%**	**4.93%**	**4.08%**	**6.14%**	**6.50%**	**6.70%**	**6.19%**	**5.80%**
NGCF	0.0464	0.0737	0.0948	0.1132	0.0364	0.0591	0.0768	0.0919	0.0747	0.1068	0.1312	0.1517	0.1150	0.1860	0.2396	0.2830
NGCF-MAL	0.0569	0.0903	0.1151	0.1339	0.0446	0.0753	0.0961	0.1127	0.0802	0.1141	0.1389	0.1613	0.1248	0.1989	0.2534	0.2997
Imp.	**22.63%**	**22.52%**	**21.41%**	**18.29%**	**22.53%**	**27.41%**	**25.13%**	**22.63%**	**7.36%**	**6.84%**	**5.87%**	**6.33%**	**8.52%**	**6.94%**	**5.76%**	**5.90%**
RecVAE	0.0529	0.0827	0.1081	0.1267	0.0390	0.0648	0.0842	0.1005	0.0784	0.1137	0.1400	0.1617	0.1232	0.1956	0.2503	0.2982
RecVAE-MAL	0.0565	0.0880	0.1138	0.1326	0.0403	0.0672	0.0878	0.1049	0.0812	0.1181	0.1461	0.1696	0.1303	0.2061	0.2625	0.3116
Imp.	**6.78%**	**6.43%**	**5.29%**	**4.69%**	**3.48%**	**3.79%**	**4.19%**	**4.36%**	**3.54%**	**3.88%**	**4.39%**	**4.87%**	**5.83%**	**5.35%**	**4.88%**	**4.48%**
LightGCN	0.0557	0.0879	0.1130	0.1323	0.0478	0.0798	0.1056	0.1220	0.0891	0.1268	0.1552	0.1785	0.1286	0.2027	0.2578	0.3040
LightGCN-MAL	0.0602	0.0947	0.1197	0.1395	0.0496	0.0832	0.1113	0.1287	0.0921	0.1315	0.1613	0.1856	0.1368	0.2154	0.2733	0.3214
Imp.	**8.08%**	**7.74%**	**5.93%**	**5.44%**	**3.77%**	**4.26%**	**5.40%**	**5.49%**	**3.37%**	**3.71%**	**3.93%**	**3.98%**	**6.38%**	**6.27%**	**6.01%**	**5.72%**
DCF	0.0509	0.0805	0.1033	0.1227	0.0394	0.0645	0.0841	0.0991	0.0793	0.1153	0.1405	0.1629	0.1238	0.1967	0.2543	0.3005
DCF-MAL	0.0547	0.0860	0.1089	0.1287	0.0410	0.0675	0.0889	0.1049	0.0822	0.1199	0.1466	0.1702	0.1321	0.2094	0.2697	0.3181
Imp.	**7.52%**	**6.82%**	**5.43%**	**4.86%**	**4.27%**	**4.69%**	**5.72%**	**5.82%**	**3.76%**	**3.98%**	**4.38%**	**4.48%**	**6.72%**	**6.43%**	**6.08%**	**5.86%**
IMP-GCN	0.0582	0.0920	0.1185	0.1386	0.0500	0.0837	0.1106	0.1277	0.0928	0.1324	0.1616	0.1864	0.1346	0.2130	0.2710	0.3186
IMP-GCN-MAL	0.0620	0.0978	0.1243	0.1448	0.0514	0.0864	0.1149	0.1337	0.0950	0.1359	0.1664	0.1923	0.1410	0.2226	0.2818	0.3302
Imp.	**6.49%**	**6.20%**	**4.87%**	**4.48%**	**2.76%**	**3.14%**	**3.88%**	**4.67%**	**2.28%**	**2.59%**	**2.99%**	**3.19%**	**4.80%**	**4.49%**	**3.98%**	**3.65%**
SGL	0.0606	0.0911	0.1138	0.1328	0.0505	0.0866	0.1139	0.1313	0.0878	0.1239	0.1536	0.1753	0.1378	0.2184	0.2789	0.3272
SGL-MAL	0.0630	0.0948	0.1182	0.1368	0.0543	0.0902	0.1154	0.1331	0.0892	0.1252	0.1553	0.1772	0.1427	0.2214	0.2843	0.3320
Imp.	**3.96%**	**4.06%**	**3.87%**	**3.01%**	**7.52%**	**4.16%**	**1.32%**	**1.37%**	**1.59%**	**1.07%**	**1.09%**	**1.08%**	**3.56%**	**1.37%**	**1.94%**	**1.46%**

TABLE 4
The performance comparison of all methods in terms of *NDCG@K*. We present the relative improvements of our framework over the base model in bold. *Imp.* denotes the improvement.

Methods	CDs				Books				Gowalla				ML20m			
	N@5	N@10	N@15	N@20	N@5	N@10	N@15	N@20	N@5	N@10	N@15	N@20	N@5	N@10	N@15	N@20
NeuMF	0.0377	0.0363	0.0360	0.0358	0.0340	0.0326	0.0320	0.0317	0.1052	0.0930	0.0899	0.0884	0.2437	0.2196	0.2083	0.2018
CMN	0.0385	0.0371	0.0368	0.0366	0.0352	0.0338	0.0331	0.0328	0.1067	0.0948	0.0916	0.0902	0.2475	0.2232	0.2115	0.2046
GC-MC	0.0380	0.0367	0.0363	0.0361	0.0347	0.0332	0.0326	0.0323	0.1073	0.0952	0.0922	0.0908	0.2486	0.2229	0.2125	0.2053
SinkhornCF	0.0430	0.0416	0.0413	0.0409	0.0440	0.0426	0.0419	0.0416	0.1123	0.1004	0.0971	0.0958	0.2641	0.2406	0.2294	0.2217
MF	0.0392	0.0378	0.0374	0.0371	0.0359	0.0345	0.0339	0.0335	0.1090	0.0978	0.0947	0.0932	0.2537	0.2299	0.2188	0.2122
MF-MAL	0.0479	0.0471	0.0464	0.0459	0.0468	0.0451	0.0442	0.0438	0.1162	0.1042	0.1008	0.0991	0.2792	0.2517	0.2387	0.2313
Imp.	**22.19%**	**24.60%**	**24.06%**	**23.72%**	**30.36%**	**30.72%**	**30.38%**	**30.75%**	**6.61%**	**6.54%**	**6.44%**	**6.33%**	**10.05%**	**9.48%**	**9.10%**	**9.00%**
MultVAE	0.0424	0.0411	0.0407	0.0404	0.0436	0.0423	0.0415	0.0412	0.1118	0.0996	0.0967	0.0952	0.2608	0.2387	0.2276	0.2203
MultVAE-MAL	0.0504	0.0498	0.0492	0.0485	0.0511	0.0489	0.0476	0.0464	0.1225	0.1113	0.1057	0.1031	0.2843	0.2598	0.2453	0.2380
Imp.	**18.87%**	**21.17%**	**20.88%**	**20.05%**	**17.20%**	**15.60%**	**14.70%**	**12.62%**	**9.57%**	**11.75%**	**9.31%**	**8.30%**	**9.01%**	**8.84%**	**7.78%**	**8.03%**
NGCF	0.0402	0.0388	0.0385	0.0383	0.0429	0.0412	0.0406	0.0403	0.1108	0.0993	0.0960	0.0945	0.2560	0.2329	0.2222	0.2147
NGCF-MAL	0.0491	0.0484	0.0476	0.0469	0.0498	0.0476	0.0463	0.0457	0.1214	0.1097	0.1045	0.1023	0.2811	0.2556	0.2414	0.2347
Imp.	**22.14%**	**24.74%**	**23.64%**	**22.45%**	**16.08%**	**15.53%**	**14.04%**	**13.40%**	**9.57%**	**10.47%**	**8.85%**	**8.25%**	**9.80%**	**9.75%**	**8.64%**	**9.32%**
RecVAE	0.0442	0.0430	0.0425	0.0419	0.0457	0.0439	0.0430	0.0427	0.1163	0.1043	0.1006	0.0994	0.2703	0.2484	0.2369	0.2297
RecVAE-MAL	0.0470	0.0459	0.0454	0.0446	0.0491	0.0471	0.0460	0.0455	0.1220	0.1092	0.1050	0.1036	0.2816	0.2593	0.2481	0.2416
Imp.	**6.25%**	**6.89%**	**6.74%**	**6.49%**	**7.45%**	**7.27%**	**6.92%**	**6.55%**	**4.87%**	**4.77%**	**4.42%**	**4.18%**	**4.17%**	**4.39%**	**4.73%**	**5.19%**
LightGCN	0.0496	0.0483	0.0478	0.0474	0.0586	0.0568	0.0559	0.0556	0.1261	0.1143	0.1125	0.1112	0.2807	0.2586	0.2498	0.2422
LightGCN-MAL	0.0531	0.0518	0.0512	0.0509	0.0638	0.0620	0.0612	0.0608	0.1310	0.1198	0.1163	0.1148	0.2934	0.2711	0.2632	0.2556
Imp.	**7.06%**	**7.25%**	**7.11%**	**7.38%**	**8.87%**	**9.15%**	**9.48%**	**9.35%**	**3.89%**	**4.81%**	**3.38%**	**3.24%**	**4.52%**	**4.83%**	**5.36%**	**5.53%**
DCF	0.0442	0.0427	0.0421	0.0418	0.0470	0.0449	0.0442	0.0438	0.1205	0.1080	0.1056	0.1027	0.2786	0.2539	0.2442	0.2361
DCF-MAL	0.0471	0.0456	0.0449	0.0445	0.0507	0.0483	0.0474	0.0468	0.1248	0.1131	0.1094	0.1061	0.2908	0.2658	0.2563	0.2486
Imp.	**6.59%**	**6.82%**	**6.62%**	**6.52%**	**7.88%**	**7.46%**	**7.16%**	**6.82%**	**3.57%**	**4.76%**	**3.58%**	**3.32%**	**4.39%**	**4.69%**	**4.93%**	**5.30%**
IMP-GCN	0.0520	0.0505	0.0501	0.0499	0.0613	0.0599	0.0585	0.0586	0.1307	0.1187	0.1168	0.1153	0.2972	0.2727	0.2641	0.2550
IMP-GCN-MAL	0.0555	0.0541	0.0536	0.0534	0.0664	0.0651	0.0638	0.0639	0.1355	0.1238	0.1208	0.1189	0.3099	0.2852	0.2772	0.2685
Imp.	**6.83%**	**7.02%**	**6.92%**	**7.07%**	**8.26%**	**8.62%**	**9.16%**	**9.08%**	**3.65%**	**4.28%**	**3.43%**	**3.19%**	**4.28%**	**4.60%**	**4.97%**	**5.28%**
SGL	0.0542	0.0528	0.0524	0.0522	0.0652	0.0643	0.0630	0.0628	0.1235	0.1117	0.1098	0.1094	0.3021	0.2803	0.2714	0.2626
SGL-MAL	0.0568	0.0547	0.0542	0.0538	0.0673	0.0661	0.0647	0.0641	0.1256	0.1148	0.1125	0.1119	0.3098	0.2864	0.2765	0.2684
Imp.	**4.80%**	**3.52%**	**3.44%**	**3.07%**	**3.22%**	**2.80%**	**2.70%**	**2.07%**	**1.70%**	**2.78%**	**2.46%**	**2.29%**	**2.55%**	**2.18%**	**1.88%**	**2.21%**

TABLE 5
he performance comparison of all methods in terms of *Precision@K*. We present the relative improvements of our framework over the base model in bold. *Imp.* denotes the improvement.

Methods	CDs				Books				Gowalla				ML20m			
	P@5	P@10	P@15	P@20	P@5	P@10	P@15	P@20	P@5	P@10	P@15	P@20	P@5	P@10	P@15	P@20
NeuMF	0.0327	0.0268	0.0231	0.0213	0.0312	0.0267	0.0243	0.0216	0.0830	0.0590	0.0498	0.0418	0.2281	0.1903	0.1662	0.1504
CMN	0.0334	0.0271	0.0235	0.0211	0.0315	0.0271	0.0246	0.0218	0.0845	0.0609	0.0500	0.0423	0.2353	0.1920	0.1696	0.1529
GC-MC	0.0330	0.0258	0.0235	0.0207	0.0317	0.0273	0.0248	0.0221	0.0801	0.0564	0.0489	0.0405	0.2254	0.1937	0.1675	0.1512
SinkhornCF	0.0370	0.0294	0.0259	0.0239	0.0409	0.0334	0.0301	0.0277	0.0904	0.0660	0.0559	0.0483	0.2503	0.2126	0.1810	0.1675
MF	0.0337	0.0271	0.0238	0.0215	0.0329	0.0282	0.0254	0.0234	0.0876	0.0633	0.0521	0.0454	0.2338	0.1933	0.1695	0.1533
MF-MAL	0.0410	0.0331	0.0284	0.0253	0.0428	0.0363	0.0323	0.0295	0.0937	0.0684	0.0563	0.0491	0.2586	0.2164	0.1874	0.1686
Imp.	**21.66%**	**22.14%**	**19.33%**	**17.67%**	**30.09%**	**28.72%**	**27.17%**	**26.07%**	**6.96%**	**8.06%**	**8.06%**	**8.15%**	**10.61%**	**11.95%**	**10.56%**	**9.98%**
MultVAE	0.0375	0.0294	0.0264	0.0238	0.0406	0.0331	0.0298	0.0272	0.0918	0.0672	0.0559	0.0489	0.2512	0.2063	0.1826	0.1645
MultVAE-MAL	0.0425	0.0337	0.0300	0.0266	0.0485	0.0405	0.0361	0.0328	0.0976	0.0701	0.0584	0.0515	0.2656	0.2194	0.1931	0.1726
Imp.	**13.24%**	**14.45%**	**13.39%**	**11.74%**	**19.56%**	**22.43%**	**21.29%**	**20.75%**	**6.34%**	**4.23%**	**4.64%**	**5.35%**	**5.74%**	**6.35%**	**5.75%**	**4.96%**
NGCF	0.0348	0.0274	0.0239	0.0218	0.0397	0.0320	0.0292	0.0264	0.0898	0.0643	0.0538	0.0466	0.2409	0.2040	0.1786	0.1622
NGCF-MAL	0.0422	0.0333	0.0287	0.0261	0.0478	0.0395	0.0358	0.0319	0.0955	0.0682	0.0569	0.0491	0.2594	0.2186	0.1897	0.1712
Imp.	**21.43%**	**21.53%**	**20.35%**	**19.55%**	**20.37%**	**23.34%**	**22.77%**	**20.86%**	**6.35%**	**6.12%**	**5.87%**	**5.35%**	**7.68%**	**7.12%**	**6.24%**	**5.58%**
RecVAE	0.0387	0.0317	0.0270	0.0244	0.0426	0.0361	0.0320	0.0292	0.0944	0.0681	0.0579	0.0494	0.2660	0.2120	0.1872	0.1677
RecVAE-MAL	0.0409	0.0333	0.0283	0.0255	0.0442	0.0373	0.0333	0.0305	0.0977	0.0706	0.0602	0.0514	0.2796	0.2227	0.1963	0.1755
Imp.	**5.48%**	**5.24%**	**4.76%**	**4.23%**	**3.75%**	**3.35%**	**4.13%**	**4.33%**	**3.53%**	**3.76%**	**3.96%**	**4.01%**	**5.13%**	**5.03%**	**4.85%**	**4.63%**
LightGCN	0.0409	0.0331	0.0296	0.0253	0.0442	0.0378	0.0351	0.0294	0.1050	0.0793	0.0653	0.0564	0.2700	0.2185	0.1899	0.1750
LightGCN-MAL	0.0439	0.0354	0.0312	0.0269	0.0458	0.0392	0.0366	0.0315	0.1082	0.0818	0.0676	0.0584	0.2854	0.2311	0.2006	0.1843
Imp.	**7.36%**	**6.95%**	**5.25%**	**6.32%**	**3.63%**	**3.70%**	**4.13%**	**7.14%**	**3.05%**	**3.26%**	**3.51%**	**3.70%**	**5.70%**	**5.75%**	**5.64%**	**5.34%**
DCF	0.0382	0.0303	0.0263	0.0236	0.0421	0.0357	0.0315	0.0288	0.0964	0.0706	0.0576	0.0509	0.2616	0.2184	0.1905	0.1734
DCF-MAL	0.0406	0.0322	0.0279	0.0249	0.0438	0.0372	0.0330	0.0302	0.0995	0.0731	0.0598	0.0528	0.2777	0.2312	0.2013	0.1829
Imp.	**6.35%**	**6.42%**	**5.96%**	**5.74%**	**4.05%**	**4.13%**	**4.73%**	**4.85%**	**3.24%**	**3.58%**	**3.79%**	**3.91%**	**6.13%**	**5.91%**	**5.70%**	**5.48%**
IMP-GCN	0.0431	0.0338	0.0300	0.0268	0.0463	0.0395	0.0377	0.0323	0.1108	0.0789	0.0660	0.0570	0.2842	0.2345	0.2007	0.1810
IMP-GCN-MAL	0.0456	0.0356	0.0316	0.0281	0.0475	0.0407	0.0389	0.0334	0.1130	0.0808	0.0678	0.0587	0.2963	0.2442	0.2089	0.1878
Imp.	**5.78%**	**5.53%**	**5.13%**	**4.85%**	**2.68%**	**2.96%**	**3.14%**	**3.52%**	**2.03%**	**2.44%**	**2.68%**	**2.97%**	**4.25%**	**4.14%**	**4.07%**	**3.79%**
SGL	0.0427	0.0345	0.0292	0.0259	0.0466	0.0402	0.0381	0.0327	0.1014	0.0750	0.0613	0.0539	0.2903	0.2387	0.2057	0.1858
SGL-MAL	0.0448	0.0354	0.0302	0.0265	0.0481	0.0427	0.0418	0.0344	0.1027	0.0761	0.0623	0.0548	0.2945	0.2415	0.2084	0.1879
Imp.	**4.92%**	**2.61%**	**3.42%**	**2.32%**	**3.22%**	**6.22%**	**9.71%**	**5.11%**	**1.28%**	**1.47%**	**1.63%**	**1.67%**	**1.45%**	**1.17%**	**1.31%**	**1.13%**

- **Base-MAL**: our model, integrating other base models, such as MF, LightGCN, and IMP-GCN, into our meta auxiliary learning framework, which takes two self-supervised learning tasks regarding users and items, respectively, as auxiliary tasks optimized by the meta learning framework.

5.4 Experiment Settings

In the experiments, the latent dimension of all models is set to 50 for a fair comparison. In order to show the best performance of the baseline methods, we initialize the parameters as proposed in the corresponding papers, and we have fine-tuned the parameters to improve the performance. We search for the learning rate over the range [0.0001, 0.0005, 0.001, 0.005, 0.01], and tune the coefficient of L2 normalization amongst [0.0001, 0.001, ..., 0.1]. The dropout ratio is selected in the range of [0.0, 0.1, ..., 0.9] to prevent over-fitting. Besides, we employ the node dropout ratio is tuned in [0.0, 0.1, ..., 0.9] for GM-MC and NGCF. For NeuMF, we employ three hidden layers for MLP, and keep the dimension of each hidden layer the same. Regarding CMN, the number of hops is tuned over the values [1, 2, 3]. For Mult-VAE, the model architecture we use is the suggested one in the paper: $600 \rightarrow 200 \rightarrow 600$. For NGCF, we test the layer number over the values [1, 2, 3, 4]. For RecVAE, we keeps the sample times parameter of encoder and decoder satisfy $M_{enc} = 3M_{dec}$. Also, the Bernoulli noise parameter μ_{noise} is set to 0.5. Regarding DCF, the disagreement regularization weight λ_1 are searched in [0.000001, 0.00001, ..., 0.001]. For SinkhornCF, we set the hyper-parameter $\varepsilon = 1.0$ which controls the smoothness of the optimization objective and the interaction number $K = 5$. For SGL, we selcet Edge Dropout method, and tune λ_1 within the range of [0.005, 0.01, 0.05, 0.1, 0.5, 1.0]. For IMP-GCN, the L2 regularization coefficient is searched in [0.00001, 0.0001, ..., 0.01]. For the model with the MAL framework, the threshold τ which is used when selecting the neighbors of users/items is tested over the values [0.1, 0.2, ..., 0.5]. The λ^U and λ^I are initialized from the range [0.0, 0.1, ..., 1.0]. The batch size is set to 5000. The conjugate gradient step is set to 2. Hyper-parameters are tuned by the grid search on the validation set. Our experiments are conducted with PyTorch running on GPU machines (Nvidia Titan RTX).

5.5 Performance Comparison

Tables 3, 4, 5 and Figure 3 present the performance of different models in terms of Recall@K, NDCG@K, and Precision@K on four datasets, respectively. Due to the space limit, Figure 3 does not demonstrate the result of Precison@K.

Observations about our model. First, IMP-GCN-MAL, which adopts the IMP-GCN as the base model integrated into our MAL framework, achieves the best performance for all evaluation metrics on all four datasets. This illustrates the superiority and flexibility of our framework. Second, the performance of seven base models with our MAL framework is better than the corresponding original base model. One major reason is that all the base models only

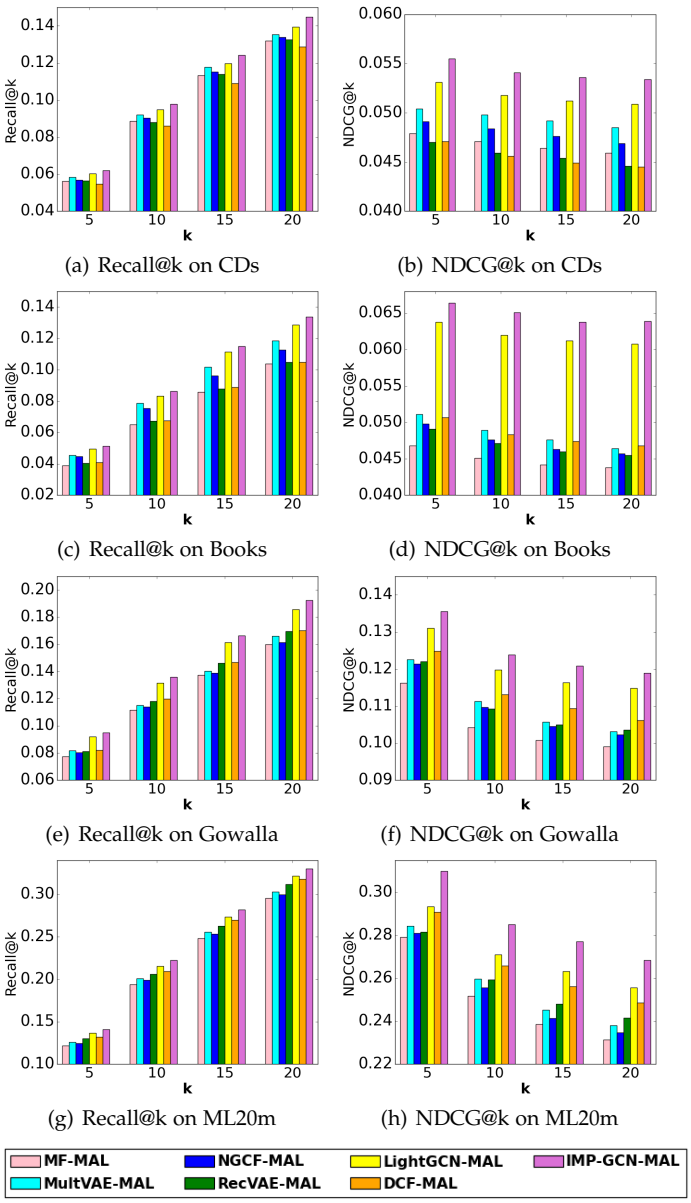


Fig. 3. The performance comparison of *Base-MAL* on all datasets.

focus on the primary recommendation task without considering incorporating additional useful features from potential auxiliary tasks. Third, IMP-GCN-MAL outperforms MF-MAL, MultVAE-MAL, and LightGCN-MAL. Even though these seven base models are developed within the same MAL framework, IMP-GCN has demonstrated its superior performance comparing with a number of state-of-the-art methods [63] on the primary recommendation task. Fourth, the base models with our MAL framework obtain better results than SinkhornCF. Although SinkhornCF is based on the Sinkhorn divergence [64] and utilizes the information of item similarities in its loss, it neglects the helpful information between users, which is well-modeled as an auxiliary task in our MAL framework. Fifth, MF-MAL, MultVAE-MAL, LightGCN-MAL achieve better performance than NeuMF, CMN, GC-MC, and NGCF. Although NeuMF, CMN, GC-MC, and NGCF model first-order and high-order connectivities between users and items, they fail to model the potential information between users and between items. On the other hand, MAL introduces auxiliary tasks to flexibly induce extra useful information for learning better embedding representations. Furthermore, MAL also applies meta learning to distinguish the importance of each auxiliary task. Sixth, LightGCN-MAL, and MF-MAL obtain better results than Mult-VAE. One possible reason is that Mult-VAE adopts a binary cross-entropy loss which is not tailored to the implicit feedback in the recommendation. By contrast, the primary task of LightGCN-MAL and MF-MAL is the BPR loss which can leverage the implicit feedback effectively.

Other observations. First, IMP-GCN yields the best results among all baseline methods, which confirms the results reported in [63]. Second, NGCF outperforms NeuMF and GC-MC. By stacking multiple embedding propagation layers, NGCF is capable of exploring the high-order connectivity in an explicit way, while NeuMF and GC-MC only utilize the first-order neighbors to guide the representation learning. Third, CMN generally performs better than GC-MC on most datasets. One major reason is that CMN has a neural attention mechanism, which can specify the attentive weight of each neighboring user, rather than the same weight used in GC-MC. Fourth, Mult-VAE achieves better performance than NGCF in most cases. One possible reason is that the polynomial likelihood is particularly suitable for modeling the implicit feedback data. Fifth, RecVAE performs better than Mult-VAE. The reason is presumably that the alternating update of the encoder and decoder training methods facilitate the model training. Sixth, DCF outperforms NGCF. One possible reason is that DCF decomposes users and items into multiple factor-level representations to characterize fine-grained preferences. Seventh, SinkhornCF outperforms MF. One major reason is that SinkhornCF induces fine topology by considering the item similarity, while MF only considers the feedback information between users and items. Eighth, IMP-GCN achieves better performance than LightGCN. IMP-GCN constructs subgraphs of users with similar interests and their interacted items. On top of these, IMP-GCN performs the high-order graph convolution on these subgraphs, while LightGCN only performs the high-order graph convolution for the graph with all users.

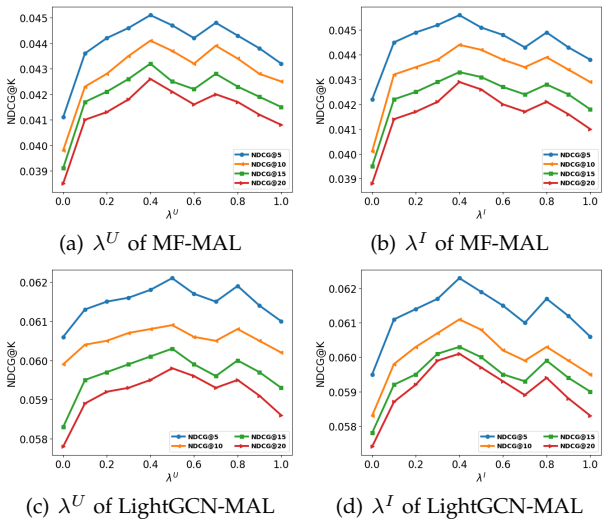
TABLE 6
The ablation analysis. *U* denotes the user auxiliary task, *I* denotes the item auxiliary task, and *M* denotes meta learning.

Architecture	CDs R@10	CDs N@10	CDs P@10	Books R@10	Books N@10	Books P@10
(1) MF	0.0724	0.0378	0.0271	0.0483	0.0345	0.0282
(2) MF+U	0.0748	0.0394	0.0279	0.0571	0.0387	0.0317
(3) MF+I	0.0793	0.0401	0.0290	0.0598	0.0392	0.0323
(4) MF+U+M	0.0765	0.0409	0.0283	0.0595	0.0401	0.0319
(5) MF+I+M	0.0818	0.0413	0.0294	0.0602	0.0398	0.0325
(6) MF+U+I	0.0858	0.0448	0.0312	0.0638	0.0436	0.0346
(7) MF-MAL	0.0885	0.0471	0.0331	0.0651	0.0451	0.0363
(1) LightGCN	0.0879	0.0483	0.0331	0.0798	0.0568	0.0378
(2) LightGCN+U	0.0891	0.0488	0.0335	0.0803	0.0577	0.0380
(3) LightGCN+I	0.0908	0.0496	0.0339	0.0811	0.0591	0.0382
(4) LightGCN+U+M	0.0897	0.0491	0.0334	0.0807	0.0583	0.0379
(5) LightGCN+I+M	0.0918	0.0502	0.0341	0.0816	0.0599	0.0385
(6) LightGCN+U+I	0.0932	0.0511	0.0347	0.0824	0.0611	0.0388
(7) LightGCN-MAL	0.0947	0.0518	0.0354	0.0832	0.0620	0.0392

5.6 Ablation Analysis

To verify the effectiveness of the proposed MAL framework, we conduct an ablation study in Table 6. These demonstrate the contribution of each module to the MAL framework. Due to the space limit, we only report the result of MF-MAL and LightGCN-MAL on the Amazon-CDs and Amazon-Books datasets. Since MF-MAL and LightGCN-MAL have the same ablation setting, we omit the description of LightGCN-MAL as follows. In (1), we use the classical MF optimized by the BPR loss. In (2), we integrate the auxiliary task regarding users into MF, where the value of λ^U is selected by a grid search. In (3), we integrate the auxiliary task regarding items into MF, where the value of λ^I is also determined by the grid search. In (4), we add the meta learning scheme to (2). In (5), we add the meta learning scheme to (3). In (6), we integrate user-based auxiliary tasks and item-based auxiliary tasks in MF, where λ^U and λ^I use the same value of which in (2) and (3). In (7), we present the overall MF-MAL model to show the effectiveness of our framework.

From the results shown in Table 6, we make the following observations. First, comparing (1), (2), and (3), we can observe that the auxiliary tasks play an important role in improving the performance of the primary task. Also, the auxiliary tasks of users and items may have different magnitudes of contributions. Second, from (2), (3), (4), and (5), we observe that by utilizing the meta learning scheme, the performance of (2) and (3) can be further improved. One major reason is that meta learning can adaptively adjust the importance of the auxiliary task to facilitate the primary recommendation task. Third, from (2), (3), and (6), we observe that the use of both auxiliary tasks together can achieve better performance than solely using one. Fourth, from (6) and (7), we can observe that our framework is good at coordinating these auxiliary tasks.



(a) λ^U of MF-MAL (b) λ^I of MF-MAL

(c) λ^U of LightGCN-MAL (d) λ^I of LightGCN-MAL

Fig. 4. The variation of hyper-parameters λ^U and λ^I.

5.7 Influence of Adaptive λ^U and λ^I

To measure the influence of meta learning in our framework, we fix either λ^U or λ^I as hyper-parameters to demonstrate the dynamic contributions of the auxiliary tasks. We conduct experiments of MF-MAL and LightGCN-MAL on the *Amazon-Books* dataset. The experimental results are shown in Figure 4. The result of other methods has a similar trend.

From the results in the figure 4, we observe that the value of the parameters λ^U and λ^I significantly affect the final recommendation performance. If these parameters are not selected properly, the performance of the recommendation will drop by about 6%. In addition, we can also obverse that when one of the parameters (λ^U and λ^I) is fixed, the optimal performance can not match the results obtained by our meta learning framework. This result adequately demonstrates that fine-grained and adaptive regularization can improve the performance of recommendations.

5.8 Influence of Implicit Gradient Method

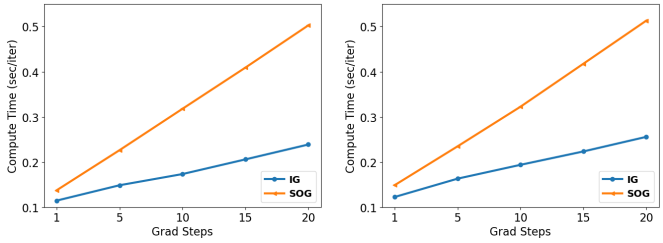
In Section 4.4, we conduct a theoretical analysis of the proposed implicit gradient method, showing that this method is theoretically efficient in terms of time complexity and accuracy when updating $\mathbf{\Lambda}$. In this section, we design a series of experiments with the goal of answering the following two questions: (1) What is the performance change in the recommendation result when using the implicit gradient method (Section 4.4) to update $\mathbf{\Lambda}$ instead of using the second-order gradient method (Section 4.3)? (2) What would the training time be when using the implicit gradient method to update $\mathbf{\Lambda}$ compared with the second-order gradient method? To answer these two questions, we conduct experiments regarding MF-MAL and LightGCN-MAL on *Amazon-Books* and *Amazon-CDs*. The performance of other methods has similar trends.

TABLE 7
The recommendation performance with different updating methods regarding $\mathbf{\Lambda}$. **SOG** denotes the second-order gradient method (Section 4.3), and **IG** denotes the implicit gradient method (Section 4.4).

Method	CDs			Books		
	R@10	N@10	P@10	R@10	N@10	P@10
MF-MAL (SOG)	0.0891	0.0474	0.0335	0.0655	0.0456	0.0367
MF-MAL (IG)	0.0885	0.0471	0.0331	0.0651	0.0451	0.0363
LightGCN-MAL (SOG)	0.0952	0.0522	0.0360	0.0834	0.0625	0.0393
LightGCN-MAL (IG)	0.0947	0.0518	0.0354	0.0832	0.0620	0.0392

For question (1), the second-order gradient method applies the one-step SGD which mentioned in Eq. 10 as the bridge to connect $\mathcal{L}_{pri}(\cdot;\mathbf{\Theta}^*(\mathbf{\Lambda}))$ and $\mathbf{\Lambda}$. Table 7 shows the recommendation results when using the implicit gradient method and second-order gradient descent to update $\mathbf{\Lambda}$, respectively. We find that when the implicit gradient method is used, the overall recommendation result is slightly lower than using the second-order gradient method. This is consistent with the theoretical analysis we have done, that using the implicit gradient method to update $\mathbf{\Theta}$ will produce an error of δ from the correct direction, resulting in a minor negative but acceptable impact on the final result.

To answer question (2), we change the number of grad steps in the the second-order gradient method and conduct experiments to demonstrate the computation time compared with the implicit gradient method. Figure 5 presents the results, and it shows that the empirical computation time



(a) MF-MAL on CDs (b) LightGCN-MAL on CDs

(c) MF-MAL on Books (d) LightGCN-MAL on Books

Fig. 5. The empirical computation time comparison. **SOG** denotes the second-order gradient method (Section 4.3), and **IG** denotes the implicit gradient method (Section 4.4).

of the original gradient descent method grows faster than the implicit gradient method. The reason is that the second-order gradient method requires the back-propagation process to compute the Hessian matrix, which is much more expensive. On the other hand, the implicit gradient method applies an L2 norm trick and adopts a conjugate gradient algorithm to approximate the optimization, which yields more efficiency gain. Notably, even when the grad step is 1, the implicit gradient method also spends less time than the one-step SGD.

6 Conclusion

In this paper, we propose a meta auxiliary learning framework, MAL, for the Top-K recommendation. Specifically, we construct two self-supervised learning tasks as auxiliary tasks to enhance the representation qualities of users and items, respectively, for facilitating the training of the primary recommendation task. The auxiliary and primary tasks are further modeled as the meta-learning paradigm to adaptively balance the contribution of auxiliary tasks for improving the primary recommendation task. The implicit gradient method is further adopted to improve the time efficiency of the meta learning framework. Experimental results on four real-world datasets clearly validate the performance advantages of our model over multiple state-of-the-art methods and demonstrate the effectiveness of each of the proposed constituent modules.

References

[1] W. Fan, Y. Ma, Q. Li, Y. He, Y. E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *WWW*. ACM, 2019, pp. 417–426.

[2] J. Sun, Y. Zhang, C. Ma, M. Coates, H. Guo, R. Tang, and X. He, "Multi-graph convolution collaborative filtering," in *ICDM*. IEEE, 2019.

[3] X. Wang, X. He, Y. Cao, M. Liu, and T. Chua, "KGAT: knowledge graph attention network for recommendation," in *KDD*. ACM, 2019, pp. 950–958.

[4] C. Ma, L. Ma, Y. Zhang, H. Wu, X. Liu, and M. Coates, "Knowledge-enhanced top-k recommendation in poincaré ball," in *AAAI*. AAAI Press, 2021, pp. 4285–4293.

[5] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," in *ICLR*. OpenReview.net, 2017.

[6] A. Valada, N. Radwan, and W. Burgard, "Deep auxiliary learning for visual localization and odometry," in *ICRA*. IEEE, 2018, pp. 6939–6946.

[7] L. Zhang, M. Yu, T. Chen, Z. Shi, C. Bao, and K. Ma, "Auxiliary training: Towards accurate and robust models," in *CVPR*. Computer Vision Foundation / IEEE, 2020, pp. 369–378.

[8] D. Hwang, J. Park, S. Kwon, K. Kim, J. Ha, and H. J. Kim, "Self-supervised auxiliary learning with meta-paths for heterogeneous graphs," in *NeurIPS*, 2020.

[9] S. Ruder, "An overview of multi-task learning in deep neural networks," *CoRR*, vol. abs/1706.05098, 2017.

[10] F. Ricci, L. Rokach, and B. Shapira, Eds., *Recommender Systems Handbook*. Springer, 2015.

[11] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *WWW*. ACM, 2001.

[12] Y. Koren, "Factorization meets the neighborhood: a multifaceted collaborative filtering model," in *KDD*. ACM, 2008.

[13] T. Tran, K. Lee, Y. Liao, and D. Lee, "Regularizing matrix factorization with user and item embeddings for recommendation," in *CIKM*. ACM, 2018.

[14] R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. M. Lukose, M. Scholz, and Q. Yang, "One-class collaborative filtering," in *ICDM*. IEEE Computer Society, 2008.

[15] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *ICDM*. IEEE Computer Society, 2008, pp. 263–272.

[16] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "BPR: bayesian personalized ranking from implicit feedback," in *UAI*. AUAI Press, 2009, pp. 452–461.

[17] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. Chua, "Neural collaborative filtering," in *WWW*. ACM, 2017, pp. 173–182.

[18] Y. Wu, C. DuBois, A. X. Zheng, and M. Ester, "Collaborative denoising auto-encoders for top-n recommender systems," in *WSDM*. ACM, 2016.

[19] C. Ma, Y. Zhang, Q. Wang, and X. Liu, "Point-of-interest recommendation: Exploiting self-attentive autoencoders with neighbor-aware influence," in *CIKM*. ACM, 2018.

[20] C. Ma, P. Kang, B. Wu, Q. Wang, and X. Liu, "Gated attentive-autoencoder for content-aware recommendation," in *WSDM*. ACM, 2019.

[21] J. Lian, X. Zhou, F. Zhang, Z. Chen, X. Xie, and G. Sun, "xdeepfm: Combining explicit and implicit feature interactions for recommender systems," in *KDD*. ACM, 2018.

[22] H. Xue, X. Dai, J. Zhang, S. Huang, and J. Chen, "Deep matrix factorization models for recommender systems," in *IJCAI*. ijcai.org, 2017.

[23] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017, pp. 1024–1034.

[24] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, "Lightgcn: Simplifying and powering graph convolution network for recommendation," in *SIGIR*. ACM, 2020, pp. 639–648.

[25] S. Ji, Y. Feng, R. Ji, X. Zhao, W. Tang, and Y. Gao, "Dual channel hypergraph collaborative filtering," in *KDD*, 2020, pp. 2020–2029.

[26] S. Toshniwal, H. Tang, L. Lu, and K. Livescu, "Multitask learning with low-level auxiliary tasks for encoder-decoder based speech recognition," in *INTERSPEECH*. ISCA, 2017, pp. 3532–3536.

[27] L. Liebel and M. Körner, "Auxiliary tasks in multi-task learning," *CoRR*, vol. abs/1805.06334, 2018.

[28] J. Yu and J. Jiang, "Learning sentence embeddings with auxiliary tasks for cross-domain sentiment classification," in *EMNLP*, 2016, pp. 236–246.

[29] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," in *CVPR*. IEEE Computer Society, 2017, pp. 6612–6619.

[30] Y. Du, W. M. Czarnecki, S. M. Jayakumar, R. Pascanu, and B. Lakshminarayanan, "Adapting auxiliary losses using gradient similarity," *CoRR*, vol. abs/1812.02224, 2018.

[31] Z. Chi, Y. Wang, Y. Yu, and J. Tang, "Test-time fast adaptation for dynamic scene deblurring via meta-auxiliary learning," in *ICCV*, 2021, pp. 9137–9146.

[32] S. Liu, A. J. Davison, and E. Johns, "Self-supervised generalisation with meta auxiliary learning," in *NeurIPS*, 2019, pp. 1677–1687.
[33] A. Navon, I. Achituve, H. Maron, G. Chechik, and E. Fetaya, "Auxiliary learning by implicit differentiation," in *ICLR*. OpenReview.net, 2021.
[34] D. Zhang, H. Wu, G. Zeng, Y. Yang, W. Qiu, Y. Chen, and H. Hu, "Ctnocvr: A novelty auxiliary task making the lower-ctr-higher-cvr upper," in *SIGIR*, 2022, pp. 2272–2276.
[35] Y. He, X. Feng, C. Cheng, G. Ji, Y. Guo, and J. Caverlee, "Metabalance: Improving multi-task recommendations via adapting gradient magnitudes of auxiliary tasks," in *WWW*, 2022, pp. 2205–2215.
[36] J. Schmidhuber, "Learning complex, extended sequences using the principle of history compression," *Neural Comput.*, vol. 4, no. 2, pp. 234–242, 1992.
[37] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *ICLR*. OpenReview.net, 2017.
[38] M. Jaderberg, W. M. Czarnecki, S. Osindero, O. Vinyals, A. Graves, D. Silver, and K. Kavukcuoglu, "Decoupled neural interfaces using synthetic gradients," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 1627–1635.
[39] J. Shu, Q. Xie, L. Yi, Q. Zhao, S. Zhou, Z. Xu, and D. Meng, "Meta-weight-net: Learning an explicit mapping for sample weighting," *NIPS*, vol. 32, 2019.
[40] Z. Li, F. Zhou, F. Chen, and H. Li, "Meta-sgd: Learning to learn quickly for few shot learning," *CoRR*, vol. abs/1707.09835, 2017.
[41] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 1126–1135.
[42] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. P. Lillicrap, "Meta-learning with memory-augmented neural networks," in *ICML*, ser. JMLR Workshop and Conference Proceedings, vol. 48. JMLR.org, 2016, pp. 1842–1850.
[43] J. Snell, K. Swersky, and R. S. Zemel, "Prototypical networks for few-shot learning," in *NIPS*, 2017, pp. 4077–4087.
[44] Y. Lu, X. Jiang, Y. Fang, and C. Shi, "Learning to pre-train graph neural networks," in *AAAI*, 2021.
[45] Y. Koren, R. M. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.
[46] M. Tschannen, J. Djolonga, P. K. Rubenstein, S. Gelly, and M. Lucic, "On mutual information maximization for representation learning," in *ICLR*. OpenReview.net, 2020.
[47] M. I. Belghazi, A. Baratin, S. Rajeswar, S. Ozair, Y. Bengio, R. D. Hjelm, and A. C. Courville, "Mutual information neural estimation," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 80. PMLR, 2018, pp. 530–539.
[48] B. Poole, S. Ozair, A. van den Oord, A. Alemi, and G. Tucker, "On variational bounds of mutual information," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 5171–5180.
[49] A. van den Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *CoRR*, vol. abs/1807.03748, 2018.
[50] M. Gutmann and A. Hyvärinen, "Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics," *J. Mach. Learn. Res.*, vol. 13, pp. 307–361, 2012.
[51] Y. Xie, Z. Xu, Z. Wang, and S. Ji, "Self-supervised learning of graph neural networks: A unified review," *CoRR*, vol. abs/2102.10757, 2021.
[52] Y. Chen, B. Chen, X. He, C. Gao, Y. Li, J. Lou, and Y. Wang, "λopt: Learn to regularize recommender models in finer levels," in *KDD*. ACM, 2019, pp. 978–986.
[53] A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine, "Meta-learning with implicit gradients," in *NeurIPS*, 2019, pp. 113–124.
[54] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR (Poster)*, 2015.
[55] T. Ebesu, B. Shen, and Y. Fang, "Collaborative memory network for recommendation systems," in *SIGIR*. ACM, 2018, pp. 515–524.
[56] R. van den Berg, T. N. Kipf, and M. Welling, "Graph convolutional matrix completion," *CoRR*, vol. abs/1706.02263, 2017.
[57] D. Liang, R. G. Krishnan, M. D. Hoffman, and T. Jebara, "Variational autoencoders for collaborative filtering," in *WWW*. ACM, 2018, pp. 689–698.
[58] X. Wang, X. He, M. Wang, F. Feng, and T. Chua, "Neural graph collaborative filtering," in *SIGIR*. ACM, 2019, pp. 165–174.
[59] I. Shenbin, A. Alekseev, E. Tutubalina, V. Malykh, and S. I. Nikolenko, "Recvae: A new variational autoencoder for top-n recommendations with implicit feedback," in *WSDM*, 2020.
[60] X. Li, J. Y. Chin, Y. Chen, and G. Cong, "Sinkhorn collaborative filtering," in *WWW*. ACM / IW3C2, 2021, pp. 582–592.
[61] H. Chen, X. Xin, D. Wang, and Y. Ding, "Decomposed collaborative filtering: Modeling explicit and implicit factors for recommender systems," in *WSDM*, 2021, pp. 958–966.
[62] J. Wu, X. Wang, F. Feng, X. He, L. Chen, J. Lian, and X. Xie, "Self-supervised graph learning for recommendation," in *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*, 2021, pp. 726–735.
[63] F. Liu, Z. Cheng, L. Zhu, Z. Gao, and L. Nie, "Interest-aware message-passing gcn for recommendation," in *WWW*, 2021, pp. 1296–1305.
[64] M. Cuturi, "Sinkhorn distances: Lightspeed computation of optimal transport," in *NIPS*, 2013, pp. 2292–2300.

Ximing Li received his B.Eng. degree in Software Engineering from Beijing Institute of Technology, China in 2020. He is currently an M.Eng. student in the School of Computer Science and Technology, under the supervision of Prof. Chi Harold Liu. From 2021, he also closely works with Prof. Chen Ma at CityU, Hong Kong SAR. His main research interests include recommender system and machine learning.

Chen Ma received his BSc and MSc degrees in Software Engineering from Beijing Institute of Technology, China in 2013 and 2015, respectively. He received his PhD degree in Computer Science from McGill University, Canada. He is currently an Assistant Professor in the Department of Computer Science, City University of Hong Kong, Hong Kong SAR. His research interests lie in the theory of data mining and machine learning and their applications in recommender system, knowledge graph, and social computing/social good.

Guozheng Li received his Ph.D. degree in Computer Science at the School of EECS, Peking University in 2021. He is currently an assistant professor with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing. His major research interests include information visualization and human-computer interaction, especially hierarchical data visualization and visualization authoring.

Peng Xu is currently a part-time PhD student at School of Computer Science and Engineering, Beijing Institute of Technology, China, under the supervision of Prof. Chi Harold Liu. He received his M.Eng. degree at Beijing University of Posts and Telecommunications, and his current research interests are edge computing and deep learning.

Chi Harold Liu (SM'15) receives a Ph.D. degree in Electronic Engineering from Imperial College, UK in 2010, and a B.Eng. degree in Electronic and Information Engineering from Tsinghua University, China in 2006.

He is currently a Full Professor and Vice Dean at the School of Computer Science and Technology, Beijing Institute of Technology, China. Before moving to academia, he worked for IBM Research - China as a staff researcher and project manager from 2010 to 2013, worked as a postdoctoral researcher at Deutsche Telekom Laboratories, Germany in 2010, and as a Research Staff Member at IBM T. J. Watson Research Center, USA in 2009. His current research interests include the big data analytics, mobile computing, and machine learning. He received the IBM First Plateau Invention Achievement Award in 2012, ACM SigKDD'21 Best Paper Runner-up Award, ACM MobiCom'21 Best Commnity Paper Runner-up Award, and IEEE DataCom'16 Best Paper Award. He serves as the Associate Editor for IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING (with Excellent Editor Award, 2021). He is a senior member of IEEE and a Fellow of IET, British Computer Society, and Royal Society of Arts.

Ye Yuan received the B.S., M.S., and Ph.D. degrees in computer science from Northeastern University, in 2004, 2007, and 2011, respectively. He is currently a Professor with the Department of Computer Science, Beijing Institute of Technology, China. His research interests include graph databases, probabilistic databases, and social network analysis.

Guoren Wang received the BSc, MSc, and PhD degrees from the Department of Computer Science, Northeastern University, China, in 1988, 1991 and 1996, respectively. Currently, he is a Professor in the Department of Computer Science, Beijing Institute of Technology, Beijing, China. His research interests include XML data management, query processing and optimization, bioinformatics, high dimensional indexing, parallel database systems, and cloud data management. He has published more than 100 research papers.