Retrieving GNN Architecture for Collaborative Filtering

Fengqi Liang*
Beijing University of Posts and Telecommunications
Beijing, China
lfq@bupt.edu.cn

Huan Zhao*
4Paradigm Inc.
Beijing, China
zhaohuan@4paradigm.com

Zhenyi Wang
Wei Fang
Chuan Shi†
Beijing University of Posts and Telecommunications
Beijing, China
{zy_wang,fang_wei,shichuan}@bupt.edu.cn

ABSTRACT

Graph Neural Networks (GNNs) have been widely used in Collaborative Filtering (CF). However, when given a new recommendation scenario, the current options are either selecting from existing GNN architectures or employing Neural Architecture Search (NAS) to obtain a well-performing GNN model, both of which are expensive in terms of human expertise or computational resources. To address the problem, in this work, we propose a novel neural retrieval approach, dubbed RGCF, to search a well-performing architecture for GNN-based CF rapidly when handling new scenarios. Specifically, we design the neural retrieval approach based on meta-learning by developing two-level meta-features, ranking loss, and task-level data augmentation, and in a retrieval paradigm, RGCF can directly return a well-performing architecture given a new dataset (query), thus being efficient inherently. Experimental results on two mainstream tasks, i.e., rating prediction and item ranking, show that RGCF outperforms all models either by human-designed or NAS on two new datasets in terms of effectiveness and efficiency. Particularly, the efficiency improvement is significant, taking as an example that RGCF is 61.7-206.3x faster than a typical reinforcement learning based NAS approach on the two new datasets. Code and data are available at https://github.com/BUPT-GAMMA/RGCF.

CCS CONCEPTS

• **Information systems** → **Collaborative filtering**.

KEYWORDS

Collaborative Filtering, Graph Neural Networks, Neural Architecture Search, Meta-learning

ACM Reference Format:
Fengqi Liang, Huan Zhao, Zhenyi Wang, Wei Fang, and Chuan Shi. 2023. Retrieving GNN Architecture for Collaborative Filtering. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3583780.3615035

*Equal contribution.
†Corresponding author.

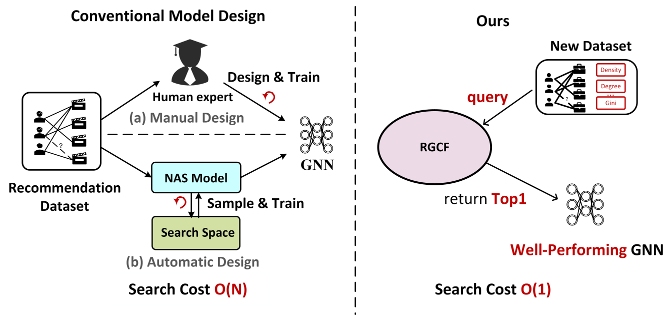


Figure 1: Comparison between conventional model design and our approach: Given a recommendation dataset, the conventional manual/automatic manners (Left) require human experts or a NAS model to design architectures or sample architectures from a search space and then train those architectures repeatedly. The search cost of two conventional manners are O(N), due to multiple model selecting and training. Our neural retrieval approach (Right) can dynamically retrieve a well-performing GNN for each query dataset with scarcely any search consumption. Thus, our approach cuts down the search cost from O(N) to O(1).

1 INTRODUCTION

Collaborative Filtering (CF) [10, 34] has been the most foundational technology in recommender systems to estimate the likelihood of interaction between users and items based on the history of interactions (clicking or rating). With the development of Graph Neural Networks (GNN) [8, 16, 38] for graph representation learning, numerous successful GNN-based models have been proposed and widely used in CF by modeling the user-item interactions as a bipartite graph to capture graph structural information and learn better user/item representations, such as PinSage [53], NGCF [41], LightGCN [9], DGCF [42], etc.

Despite the success of the manual design process, with the explosive growth of data and scenarios, given a new recommendation scenario, how to design a top-performing GNN architecture becomes a challenging problem [43]. As shown in Figure 1(a), the direct approach to solving this problem is that human experts spend massive time designing and picking a top-performing architecture for every new scenario, which leads to a huge expense of human expertise. Recently, with the success of Neural Architecture Search (NAS) [22, 30, 60] in computer vision, we can search a data-specific GNN-based CF model automatically on a pre-defined search space by adopting NAS methods, e.g., Reinforcement Learning (RL). However, as shown in Figure 1(b), given a new dataset, conventional NAS methods have to search from scratch, which is time- and resource-consuming. To address this problem, a straightforward method is adopting some meta-learning based methods [14, 18, 36], which learn a meta model from a series of tasks and leverage the learned prior knowledge to do fast architecture search on all new tasks in computer vision. However, these methods are not suitable for our problem, due to the gap between recommendation scenarios and visual tasks in various aspects. In this paper, we propose a novel neural **R**etrieval approach to design and search **G**NN-based **C**ollaborative **F**iltering (RGCF) architectures rapidly when handling a new recommendation scenario, based on meta-learning and retrieval paradigm which aims to retrieve the best-fitted item (architecture) for each given query (dataset). A sketchy paradigm of RGCF is shown in Figure 1 (right). However, it is non-trivial to design such a neural retrieval approach, presenting us with the following three key challenges.

- *How to represent different recommendation datasets?* Our neural retrieval approach measures the correlation between the GNN architecture and the recommendation dataset representation (query). One direct challenge is how to get the representation reflecting properties of recommendation datasets in various aspects, which is beneficial to retrieve the best-fitted architecture. The meta-learning based methods [14, 18, 36] in computer vision utilize a set encoder [55], which encodes full or part of images in a visual dataset, to represent it. However, since recommendation datasets may not have raw features on users/items bipartite graph, this manner cannot be applied to recommendation datasets directly. Meanwhile, this manner does not consider unique properties (graph structure, etc.) of recommendation datasets.
  Thus, it is necessary to explore a new approach to represent different recommendation datasets.
- *How to address the performance calibration issue in our architecture retrieval problem?* One direct approach to handle our architecture retrieval problem is training a model to predict the absolute value of the architecture performance. However, this direct approach may retrieve weak-performing architecture on new datasets, due to the performance calibration problem [19], i.e., GNN architecture for one recommendation dataset may not be consistent with the same architecture for another dataset. Thus, it is important to address the performance calibration issue in our approach.
- *How to improve the effectiveness of meta-learning based neural retrieval model with limited meta datasets?*
  In practice, the real-world recommendation datasets are limited, i.e., we can only obtain several recommendation datasets to train a meta-learning based retrieval model. Then insufficient meta-training tasks (meta datasets) in the meta-training phase may lead to the trained meta model ineffective [51]. Thus, it remains to explore how to improve the effectiveness of the neural retrieval model with limited meta datasets.

To handle the first challenge, we first adopt the graph-level meta-features which capture the graph structural characteristics, and distribution-level meta-features which capture the statistical characteristics, to represent the recommendation datasets.

For the second challenge, we convert our optimization objective from predicting the architecture performance into ranking the architectures in the search space. Then we propose to jointly optimize the pairwise loss and the listwise loss to retrieve a well-performing architecture from the search space. As for the third challenge, motivated by the success of task-level data augmentation on classification and regression task [51], we apply it to the our neural retrieval approach in meta-training phase, i.e., generating more meta-training tasks, and improve the effectiveness of our approach on new query datasets. To demonstrate the search performance and efficiency of our approach, we conduct experiments on two new datasets for both rating item ranking and rating prediction tasks. Experimental results show that the architecture retrieved by our approach outperforms all the manual and automatic design baselines. Particularly, our approach is 61.7-206.3x faster than the conventional RL-based NAS approach on the same search space for both item ranking and rating prediction tasks. A simple example in Figure 2 further demonstrates the superiority of our approach on search efficiency.

To summarize, this work makes the following contributions:

- To the best of our knowledge, we first consider a novel but the practical problem of how to get a well-performing GNN architecture for a new recommendation scenario rapidly.
- We propose a novel neural retrieval approach RGCF to address this problem by two-level meta-features, ranking loss and task-level data augmentation.
- Experiments show that RGCF outperforms all manual design and NAS baselines for both rating prediction and item ranking tasks. Particularly, RGCF also outperforms all NAS baselines on search efficiency over two orders of magnitude.

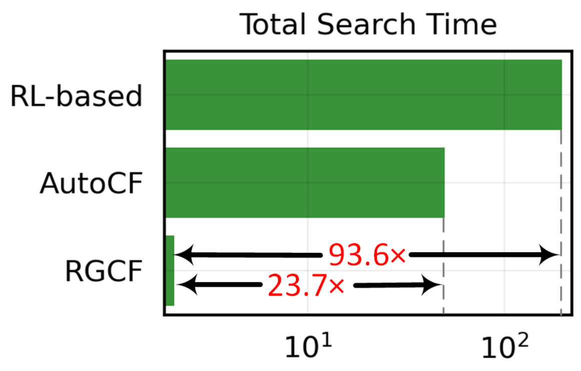


Figure 2: RGCF reduces the total search time by 93.6× compared with RL-based NAS on Epinions dataset for item ranking task. The search time of RGCF includes the time of meta feature extraction, the time for once forward propagation of our retrieval model, and the training time for the retrieved model. The time is represented on a log-scale of hours.

2 RELATED WORKS

2.1 Architecture Design for GNN-based CF

GNN-based collaborative filtering models the user-item interactions as a bipartite graph and utilizes graph neural networks to capture graph structural information and learn better user/item representations. Numerous manual design GNN-based CF models have been proposed in recent years, such as NGCF [41], LightGCN [9], DGCF [42], etc. More GNN-based CF works can be checked in the latest survey [47]. However, all the above works only design a single architecture for a specific scenario.

Neural Architecture Search (NAS) [22, 60, 61], which aims to automatically design neural architectures, have been widely adopted to design GNN [6, 13, 44–46, 57] or recommendation architectures [5, 20, 52, 59]. For example, GraphNAS [6] first adopts a RL-based algorithm to search GNN architectures; SIF [52] automatically select interaction function based on one-shot search algorithm; AutoCF [5] searches CF model by performance predictor guided random search. However, all of the NAS approaches in GNN and CF fields only handle a specific task and have to search from scratch for each new scenario repeatedly. A recent study [43] attempts to address the problem by shrinking the search space. However, our approach provides a novel paradigm, i.e., neural retrieval, based on meta-learning methods, which can obtain a well-performing GNN architecture for new recommendation scenarios rapidly.

2.2 Meta-Learning for NAS

The goal of meta-learning [40] is to train a model to extract general knowledge over the distribution of tasks, which can rapidly adapt to a new task. Most of the meta-learning based NAS papers [14, 18, 36] based on the reasonable setting that the costs for training a meta model can be amortized across all new tasks and be neglected in real world use. A few studies [3, 21, 37] combine gradient-based meta-learning MAML [4] with differentiable NAS to search for a well-fitted architecture for the given task. However, those approaches, which simultaneously optimize operator weights and shared model parameters with different tasks, are not suitable for the GNN-based CF problem, because of non-shared initialized embedding tables between different tasks. A recent study MetaD2A [18] proposes to use a meta generator to generate task-adaptive architecture for the given task. Following MetaD2A, TNAS [36] proposes to adopt meta bayesian optimization to search sota model for new tasks. Moreover, TANS [14] utilizes a meta model to retrieve both optimal architectures and parameter values. However, those methods are limited to computer vision tasks and don't consider how to capture graph structure and statistical information for recommendation datasets, and how to train a meta model with limited meta-train recommendation datasets, when handling the problem of GNN-based CF. Very recently, Autotransfer [1] proposes to transfer prior GNN design knowledge to the novel task on node and graph classification problems. However, this approach does not explore how to design GNN models in recommendation scenarios rapidly. MetaGL [28] adopts statistical informations of structural meta features to select graph learning models for the new graph for the link prediction task. However, it does not exactly work on recommendation problem (e.g., rating prediction) and tends to select the optimal model from existing graph learning models rather than designing GNN architectures for new recommendation tasks.

2.3 Neural Retrieval

Neural retrieval aims to learn a neural networks based model to rank search results (images [7] or texts [50]) in response to a query [25]. This problem can be addressed by the manner of learning to rank [23]. However, none of these works consider how to retrieve GNN-based CF architecture when given a new recommendation dataset. More relevant TANS [14] adopts a performance predictor-guided neural model to retrieve both optimal architectures and parameter values in the visual task, which doesn't consider how to get the query representation for the recommendation dataset. Moreover, it does not consider how to address the performance calibration problem in the scenario of GNN-based recommendation.

3 METHODOLOGY

In this section, we start by formalizing the task-adaptive architecture retrieval problem and explaining the key notations. Afterward, we elaborate on the proposed neural architecture retrieval approach. Finally, we present the optimization details of our retrieval approach in the mete-training phase. An overview of the proposed RGCF is shown in Fig. 3.

3.1 Problem Definition

Our goal is to search for an optimal GNN architecture for a recommendation dataset by learning a neural retrieval modal over a series of tasks. We first define that problem as task-adaptive architecture retrieval. Let $\mathscr{G} = \{\mathcal{G}_i\}_{i=1}^{I}$ denotes a collection of I recommendation datasets and Ω denotes the GNN architecture search space, where for each $\mathcal{G}_i = (\mathcal{U}, \mathcal{I}, \mathcal{E}, \mathcal{R})$ can be regarded as a bipartite graph, where $\mathcal{U}$ and $\mathcal{I}$ are the sets of users and items, $\mathcal{E}$ is the set of user-item interactions, and $\mathcal{R}$ denotes the rating score for explicit feedback which is invisible for implicit feedback.

Meta Database. Let task $\tau = \{\Omega, \mathcal{G}^\tau\}$ denotes the best performing architecture from the search space Ω with a task distribution $p(\tau)$. The meta database $\mathcal{D}$ is the task-level training set and contains some dataset-architecture-performance triples $(\mathcal{G}^\tau, a, r) \in \mathscr{G} \times \Omega \times \mathbf{R}$, where performance $r = \mathcal{M}\left(a, \mathcal{G}^\tau_{test}\right)$, $\mathcal{G}^\tau_{test}$ is the test set of $\mathcal{G}^\tau$, and $\mathcal{M}$ denotes a metric (Recall for item ranking and RMSE for rating prediction). $\mathcal{A}^\tau \subseteq \Omega$ contains all the architectures in $\mathcal{D}$ for the same meta-train dataset $\mathcal{G}^\tau$.

Task-adaptive architecture retrieval. Given the meta database, task-adaptive architecture retrieval aims to learn a function f_{ω}, which measures the correlation between architectures and datasets, on a series of meta-training tasks. It can be formally defined as follow:

$$\omega^* = \arg\min_{\omega} \mathbb{E}_{\tau \sim p(\tau)} \left[\mathcal{L}\left(f_\omega, \mathcal{A}^\tau, \mathcal{G}^\tau\right)\right], \tag{1}$$

where ω is the parameter of the meta model and $\mathcal{L}$ is the loss function of ranking loss on the meta-training tasks. After training, f_{ω^*} is applied to the meta-test phase for an new query dataset $\tilde{\mathcal{G}} \notin \mathscr{G}$ to retrieve a well-performing architecture $\hat{a}^*$ from Ω rapidly. This process can be formally defined as follow:

$$\hat{a}^* = \arg\max_{\hat{a} \in \Omega} f_{\omega^*}(\hat{a}, \tilde{\mathcal{G}}). \tag{2}$$

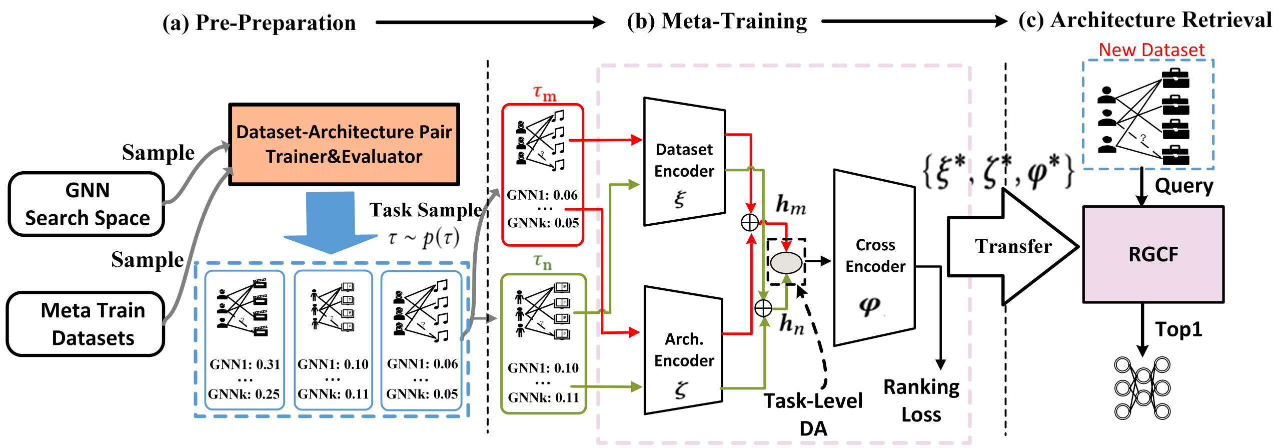


Figure 3: The overall framework of our proposed RGCF approach. (a) Pre-preparation stage. A meta database containing some dataset-architecture-performance triples is constructed at this stage. The dataset and architecture instance are sampled from a series of meta-train datasets and the GNN search space for CF. (b) Meta-training stage. We employ a PNN ζ to encode architectures, a MLP ξ to encode datasets, and a MLP φ to encode the combination of the outputs of two previous encoders. The red and green parts correspond to two sampled tasks (triples for the same dataset). Task-level data augmentation is applied to the cross encoder and the framework is optimized by ranking loss. (c) Architecture retrieval stage. The top1 scored architecture will be retrieved by our optimized framework for the new dataset in the meta-test phase.

3.2 Neural Retrieval Approach

Meta Database Construction: The meta database $\mathcal{D}$ contains some triples $(\mathcal{G}^\tau, a, r)$, which correspond to Figure 3(a) and definition in subsection 3.1. Note that the test set is visible for getting the performance r on meta-train datasets.

For a specific dataset $\mathcal{G}^\tau$, the architectures in the database for the same dataset constitute $\mathcal{A}^\tau$. The most straightforward way to construct $\mathcal{A}^\tau$, is training all architectures in the search space, i.e., $\mathcal{A}^\tau = \Omega$. However, it seems impossible to do that for all the datasets in a large search space. So $\mathcal{A}^\tau$ in our framework is formed by the architectures randomly sampled K times from the search space Ω where $K \ll |\Omega|$. After that, we train all the architectures in $\mathcal{A}^\tau$ and record all the performance r.

We repeat the construction process of $\mathcal{A}^\tau$ on all meta-train datasets $\mathscr{G}$. Then, all the records r with the corresponding dataset and architecture constitute the meta database. The meta database construction process corresponds to lines 1-9 in Algorithm 1.

Cross Encoder: The cross encoder encodes the dataset-architecture pair representation by concatenating the output of dataset and architecture encoder. The output of the cross encoder is the predicted correlation score of each dataset-architecture pair which is task-related. It can be formulated as follow:

$$\begin{aligned} f_\omega(a, \mathcal{G}^\tau) &= E_c\left(\boldsymbol{h_t}, \boldsymbol{\varphi}\right),\ \boldsymbol{h_t} = [\boldsymbol{h_d} \,||\, \boldsymbol{h_\alpha}], \\ \boldsymbol{h_d} &= E_d\left(\Phi^\tau; \xi\right) \ \text{ and } \ \boldsymbol{h_\alpha} = E_\alpha\left(a; \zeta\right), \end{aligned} \tag{3}$$

where Φ^τ is the meta feature extracted from dataset $\mathcal{G}^\tau$. $E_d : \mathbb{R}^{d_\Phi} \rightarrow \mathbb{R}^d$ is the dataset encoder; $E_\alpha : \Omega \rightarrow \mathbb{R}^d$ is the architecture encoder, with the parameter ξ and ζ, respectively. $\mathbf{E}_c : \mathbb{R}^{2d} \rightarrow \mathbb{R}$ is the cross encoder corresponds to MLP with the parameter $\boldsymbol{\varphi}$. Thus, three sets of parameters above constitute the parameter $\boldsymbol{\omega}$ of f in Eq. (1).

Dataset Encoder: We adopt meta-features to encode the recommendation dataset. The function of meta-features is as a query for our neural retrieval framework. A recent study [2] shows that different architectures are suitable for different recommendation datasets with different meta-features. Thus, the key to retrieving a well-fitted architecture from the search space is the more detailed meta-features to represent the meta-datasets. However, there is a trade-off between retrieved model performance and the meta-feature extraction time. On the one hand, we need more expressive meta-feature for architecture retrieval. On the other hand, the extraction time for the new query dataset should be acceptable. Guided by this consideration, we propose to use the two-level meta-features which are both efficient and effective.

Graph Level: Graph level meta-features treat the recommendation dataset as a graph and describe the global graph structure which is critical for the retrieval of GNN architecture. We adopt the graph level meta-features which contain:

- $Space_{\log} = \log_{10}\left(\frac{|\mathcal{U}| \text{x} |\mathcal{I}|}{1000}\right)$
- $Shape_{\log} = \log_{10}\left(\frac{|\mathcal{U}|}{|\mathcal{I}|}\right)$
- $Density_{\log} = \log_{10}\left(\frac{|\mathcal{E}|}{|\mathcal{U}| \text{x} |\mathcal{I}|}\right)$
- *Degree assortativity coefficient* [26]: Pearson correlation coefficient of degrees between linked nodes.
- *Number of Connected Components* [12].
- *User Average Degree.*
- *Item Average Degree.*

Let $\mathcal{P}_{graph}$ be the function to extract graph level meta-features Φ_{graph}, i.e., $\Phi^\tau_{graph} = \mathcal{P}_{graph}\left(\mathcal{G}^\tau\right)$.

Distribution Level: Distribution level meta-features containing $Gini_{user}, Gini_{item}$ from recent research on recommendation dataset [2], which represent the distribution of interactions over the set of users/items, can be a supplement of graph level meta-feature. The distribution level meta feature can be formulated as follow:

- $Gini_{\text{user}} = 1 - 2\sum_{u=1}^{|\mathcal{U}|}\left(\frac{|\mathcal{U}|+1-u}{|\mathcal{U}|+1}\right) \times \left(\frac{|\mathcal{E}_u|}{|\mathcal{E}|}\right)$

- $Gini_{\text{item}} = 1 - 2\sum_{i=1}^{|I|}\left(\frac{|\mathcal{I}|+1-i}{|\mathcal{I}|+1}\right) \times \left(\frac{|\mathcal{E}_i|}{|\mathcal{E}|}\right)$

Let $\mathcal{P}_{dis}$ be the function to extract distribution level meta-features Φ_{dis}, i.e., $\Phi^{\tau}_{dis} = \mathcal{P}_{dis}(\mathcal{G}^{\tau})$. To summarize, we concatenate the two-level meta-features to describe the recommendation dataset. We formally define the total meta-features as $\Phi = \left[\Phi_{dis}||\Phi_{graph}\right]$.

One notice is that the extraction of meta-feature on the query dataset doesn't contain the test set to avoid information leakage. After meta-features are extracted, we do standardization on the extracted meta-features and adopt a function E_d which is a two layers MLP with the relu activation function to make the meta-features level crossover.

Architecture Encoder: To encode the architecture $a \in \Omega$, we first create a randomly initialized embedding table of all the candidate operations in the search space. Let $L_a = \{o_1, o_2, \ldots, o_N\}$ denote the index list of all the operations of a architecture a on each space dimension, where N is the dimensions of the search space. Then, the inputs of architecture encoder are as follows: $\boldsymbol{H_a} = Lookup(L_a)$, where $\boldsymbol{H_a} \in \mathbb{R}^{N\times d}$. To encode the connection between the dimensions of search space, we adopt PNN [31], which adds a product layer to capture the interactive pattern between dimensions. In the product layer, our employed PNN conducts dot product between any two operations embeddings of the input architecture. Then the outputs of the product layer are concatenated to the original embeddings and fed into a fully connected layer. The backbone of the architecture encoder can be formulated as follow: $\boldsymbol{h_\alpha} = PNN(\boldsymbol{H_a})$.

3.3 Optimization

Ranking Loss. Our goal of architecture retrieval is to learn a pattern of ranking all the architectures in the search space for every new dataset. Thus, it is not necessary to predict the absolute value of a specific metric and it is more meaningful to distinguish the better model with several candidates. Moreover, similar to calibration problem [19] in collaborative ranking, the architecture performance for one dataset may not be consistent with the same architecture for another dataset. Thus, directly approximating absolute performance for meta database instances may retrieve lower-performance architecture on new datasets. To avoid these drawbacks, we adopt pairwise loss based on the fact that for all datasets an ordered pair of architectures means the same thing: one architecture is preferred to another. The approach's $\mathcal{L}_{pair}$ can be formulated as follow:

$$\begin{aligned}&\mathcal{L}_{pair}\left(f_\omega, \mathcal{A}^\tau, \mathcal{G}^\tau\right)\\ &= -\frac{1}{|\mathcal{E}|}\sum_{(i,j)\in\mathcal{E}}(y_{i,j}\log\sigma(f_{\boldsymbol{\omega}}(a_i,\mathcal{G}^\tau) - f_{\boldsymbol{\omega}}(a_j,\mathcal{G}^\tau))\\ &\quad + (1-y_{i,j})\log(1-\sigma(f_{\boldsymbol{\omega}}(a_i,\mathcal{G}^\tau) - f_{\boldsymbol{\omega}}(a_j,\mathcal{G}^\tau)))),\end{aligned} \tag{4}$$

where $\mathcal{E} = \left\{(i,j)|a_i \in \mathcal{A}^\tau, a_j \in \mathcal{A}^\tau \text{ and } a_i \neq a_j\right\}, y_{i,j} = \begin{cases}1, & r_i > r_j\\ 0.5, & r_i = r_j\\ 0, & r_i < r_j\end{cases}$

and $\sigma(\cdot)$ is the sigmoid function.

It is difficult to rank all the architecture in a large search space in the right order. In our architecture retrieval problem, we focus on retrieving the top-performing architecture. Based on that fact, following the topk learning to rank [27], we adopt the listwise loss listMLE [48] which gives the top-performance architecture a larger optimization gradient to jointly optimize the neural retrieval model. This approach's $\mathcal{L}_{list}$ can be formulated as follow:

$$\mathcal{L}_{list}\left(f_\omega, \mathcal{A}^\tau, \mathcal{G}^\tau\right) = -\log\prod_{i=1}^{|T|}\frac{\exp\left(f_\omega\left(a_i, \mathcal{G}^\tau\right)\right)}{\sum_{k=i}^{|T|}\exp\left(f_\omega\left(a_k, \mathcal{G}^\tau\right)\right)}, \tag{5}$$

where $T = \{a_1, a_2, \ldots, a_n | r_1 \geq r_2 \geq \ldots \geq r_n\}$ is the sorted set corresponding to $\mathcal{A}^\tau$ by the real performance of the architectures in $\mathcal{G}^\tau$.

Then the joint loss $\mathcal{L}$ in Eq. (2) can be formulated as follow:

$$\mathcal{L} = \rho \times \mathcal{L}_{pair} + (1-\rho) \times \mathcal{L}_{list}, \tag{6}$$

where ρ is a trade-off coefficient to balance the two types of loss.

Two-Stage Optimization: In the meta-training stage, we adopt two stages of gradient-based meta-learning model-agnostic meta-learning (MAML) [4] to optimize the above formula in Eq. (2). For every episodic, we random sample several tasks $\mathscr{T} = \{\tau \mid \tau \sim p(\tau)\}$ from the $p(\tau)$ and split each task τ into a support set $\tau_s = \{\mathcal{A}^\tau_s, \mathcal{G}^\tau\}$ and a query set $\tau_q = \{\mathcal{A}^\tau_q, \mathcal{G}^\tau\}$.

$$\omega^* \leftarrow \arg\min_{\omega}\frac{1}{|\mathscr{T}|}\sum_{i=1}^{|\mathscr{T}|}\left[\mathcal{L}\left(f_{\xi,\phi,\vartheta}, \mathcal{A}^{\tau_i}_q, \mathcal{G}^{\tau_i}\right)\right], \tag{7}$$

$$\begin{aligned}\text{where } \phi &= \boldsymbol{\zeta} - \eta\nabla_{\boldsymbol{\zeta}}\mathcal{L}\left(f_{\xi,\zeta,\boldsymbol{\varphi}}, \mathcal{A}^{\tau_i}_s, \mathcal{G}^{\tau_i}\right),\\ \vartheta &= \boldsymbol{\varphi} - \eta\nabla_{\boldsymbol{\varphi}}\mathcal{L}\left(f_{\xi,\zeta,\boldsymbol{\varphi}}, \mathcal{A}^{\tau_i}_s, \mathcal{G}^{\tau_i}\right),\end{aligned} \tag{8}$$

where η denotes the inner-loop learning rate and we freeze the parameter ξ of dataset encoder in the inner loop because the inputs of meta-feature are the same in the inner loop. With the meta-feature to identify meta datasets, we no longer need to fine-tune the single task in the meta-test stage. In the experiment, we find that the retrieved model is also well-performing without fine-tuning on the meta-test stage, which is also called zero-shot [49].

Task-level Data Augmentation (DA): Inspired by the recently meta-learning approach MLTI [51] to do a task-level DA on a task-insufficient scenario for classification and regression tasks, we apply it to our neural retrieval approach to ease the problem of the limited meta-train datasets. The key idea behind task-level DA is to generate new tasks by adopting manifold mixup [39] between meta-train tasks. MLTI theory prove that task-level DA leads to a better generalization bound and will improve the effectiveness of meta model [51]. We adopt it on our cross encoder which is task-related in our neural retrieval approach. The details of the cross encoder are a L layers MLP with activation function relu on each hidden layer, the hidden representation of task embedding $\boldsymbol{h_l}$ at the l-th layer is denoted as $\boldsymbol{h^l} = E_c\left(\boldsymbol{h_t}, \varphi_l\right)$ $(0 \leq l < L)$, where $\boldsymbol{h^0} = \boldsymbol{h_t}$. We randomly select one layer l and apply task interpolation on hidden representations like the MITL for label-sharing tasks [51] which is formulated as follows:

$$\begin{aligned}&\boldsymbol{h}^{x,l}_{mix,i} = \lambda\boldsymbol{h}^{x,l}_{m,i} + (1-\lambda)\boldsymbol{h}^{x,l}_{n,i},\ \boldsymbol{h}^{x,l}_{mix,j} = \lambda\boldsymbol{h}^{x,l}_{m,j} + (1-\lambda)\boldsymbol{h}^{x,l}_{n,j},\\ &y^x_{mix,i,j} = \lambda y^x_{m,i,j} + (1-\lambda)y^x_{n,i,j},\end{aligned} \tag{9}$$

where m, n is the index of randomly sampled two tasks from task set $\mathscr{T}$, $\lambda \in [0,1]$ is sampled from a Beta distribution Beta(α, ρ), $x \in \{s, q\}$ denotes the query set or the support set, and the subscript "*mix*" represents the mixup. We only apply it on pairwise loss based on the intuition one mixed architecture embedding is better for anthor mixed architecture embedding with mixed probability. However, the listwise loss does not have this intuition.

Algorithm 1: The Process of RGCF

```
Input: GNN search space $\Omega$, datasets collection $\mathscr{G}$ and
       ranking model $f_{\omega}$ with randomly initialized
       parameter $\omega = \{\xi, \zeta, \varphi\}$.
Output: The learned ranking model $f_{\omega^*}$.
Initialize Meta database $\mathcal{D} = \emptyset$.
foreach $\mathcal{G}^\tau \in \mathscr{G}$ do
    Subset $\mathcal{A}^\tau = \emptyset$.
    for $i \in [1, K]$ do
        Random sample $a_i$ from $\Omega$ and add $a_i$ to $\mathcal{A}^\tau$;
        Train $a_i$ on $\mathcal{G}^\tau$ to get the performance $r$;
        Add record $(\mathcal{G}^\tau, a_i, r)$ to $\mathcal{D}$;
    end
end
while not converge do
    Sample several tasks $\mathscr{T} = \{\{\mathcal{A}^\tau, \mathcal{G}^\tau\} | \tau \sim p(\tau)\}$;
    foreach $\tau_i \in \mathscr{T}$ do
        Split $\tau_i$ into support set and query set.
    end
    foreach $\tau_i \in \mathscr{T}$ do
        Randomly select layer $l$ of the cross encoder;
        Randomly select another $\tau_j$ from $\mathscr{T}$;
        Apply task interpolation via Eq. (9);
        Calculate loss on support set $\mathcal{L}_s$ via Eq. (6);
        Inner loop gradient update for $\phi$, $\vartheta$ via Eq. (8);
        Calculate loss on query set $\mathcal{L}_q^\tau$ via Eq. (6);
    end
    Update parameters $\omega$ by optimizing Eq. (7).
end
```

4 EXPERIMENT

In this section, we perform extensive experiments to evaluate the proposed method and answer four research questions:

- **RQ1:** How does RGCF perform in searching performance and searching efficiency on both rating prediction and item ranking tasks for new datasets?
- **RQ2:** How do different types of meta-features affect the performance of RGCF?
- **RQ3:** How does ranking loss and the task-level data augmentation affect the performance of RGCF?
- **RQ4:** What insights can we obtain from the retrieved models?

4.1 Experimental setup

Meta Datasets. For meta-train datasets, we adopt 9 real-world datasets including *Yelp*, *Amazon-CDs*, *Amazon-Movies*, *Amazon-Beauty*, *YahooMusic*, *Flixster*, *Douban*, *MovieLens-1M* and *MovieLens-100K* which are collected from different domains (e.g., business recommendation, e-commerce, movie rating and music rating). For meta-test datasets, we consider 2 **new** datasets: *Epinions* and *Amazon-Sports*. We follow the preprocessing and dataset splitting in [43] for rating prediction task. As for item ranking task, we treat all the observed interactions as positive instances. The details of dataset statistics are presented in Table 2.

Meta Database Construction. For the search space, we follow [43] which contains 9 dimensions and total 103,680 candidate GNN architectures. The names of dimensions and the corresponding operations are listed in Table 1. The more details can be found in [43]. To construct the meta database, following [43], we randomly search 3400 architectures for the search space on nine meta-train datasets and construct nine subsets $\mathcal{A}^\tau$ of Ω. For each meta-train dataset, we split each $\mathcal{A}^\tau$ into 0.8/0.2 train/valid sets. We sample 8 pairs as the support set and 200 pairs as the query set for each episode in the meta-training phase. As for the meta-valid phase, we calculate the average spearman's rank correlation coefficient of all valid pairs on each meta-train dataset, to save the best retrieval model.

Tasks and Metrics. We conduct experiments on two mainstream recommendation tasks: rating prediction and item ranking. *1) Rating Prediction.* Rating prediction task aims to train models to estimate the ratings users would give to items. As for the evaluation metric, we follow the [43] and adopt Rooted Mean Square Error (RMSE) to evaluate the model performance. *2) Item Ranking.* Item ranking task aims to recommend ordered lists of items for users. For the evaluation metric, we adopt common metric Recall [9, 41] on the whole item candidates [33], particularly, we adopt Recall@20. We repeat training all the searched models and human design models on 10 random seeds for the rating prediction task and 3 random seeds for the item ranking task, and report the mean and standard deviation to evaluate model performance and stability.

Baselines Settings. We report the result of two variants of our approach: RGCF-T1 and RGCF-T5 (T denotes Top). RGCF-T1 directly adopts the predicted top1 GNN architecture and RGCF-T5 select the top1 performing architecture in the valid set from the predicted top5 architectures. Our proposed methods are compared with the following two types of baselines. **1) Manually design GNN-based CF models**, including MF [17], NCF [10], NGCF [41], LightGCN [9], DGCF [42]. **2) NAS methods for CF**. We search 200 child models for item ranking task and 300 child models for rating prediction task; Random Search: randomly select 10 architectures on the whole search space; RS-10 [43]: the random search of 10 architectures in the shrunk search space pruned from the whole search space by controlled random search [54]; AutoCF [5]: A performance predictor guiding random search without considering GNN in design space dimension. We adopt the default search setting in their raw paper [5] and only change the number of search models to 300.

Hyperparameter Settings. For our neural retrieval model, we adopt Adam [15] optimizer with 0.0001 learning rate for the outer loop of meta-training, early stopping strategy with patience 300, and saved the best epoch on the meta-valid sets. The maximum number of epochs is set to 3,000 in the item ranking task and 1,500 in the rating prediction task. For the architecture encoder, we adopt standarded PNN with 16 hidden dimensions and select the inner product as the operation of the interactive layer. For the dataset encoder, we use a two-layer mlp with 16 hidden dimensions, and for the across encoder, we use a three-layers mlp with 32 hidden dimensions. For the inner loop, we set the learning rates $\eta = 0.001$ and the step of gradient update 2. For ranking loss, we set $\rho = 0.6$

Table 1: Following [43], we employ the SOTA GNN based CF search space in the experiments.

Initial Embedding Dimension d	Message Function $m(\cdot)$	Aggregation $f(\cdot)$
64, 128, 256	Identity, Hadamard	None, GCN, GAT, GIN, GraphSAGE
Activation $\sigma(\cdot)$	**Layer Number L**	**Layer Combination $g(\cdot)$**
Identity, Sigmoid, Tanh, ReLU, PReLU, LeakyReLU	1, 2, 3, 4	Stack, Concat, Sum, Mean
Component Number K	**Component Combination $c(\cdot)$**	**Interaction Function $p(\cdot)$**
1, 2, 3, 4	Concat, Mean, Att	Dot Product, Concat+MLP, Sum+MLP

Table 2: Statistics of datasets from different domains.

	Dataset	#Users	#Items	#Interactions
Meta-Train	Yelp	58,069	31,721	1,160,605
	Amazon-CDs	31,296	24,379	622,163
	Amazon-Movies	44,439	25,047	1,070,860
	YahooMusic	1,357	1,363	5,335
	Amazon-Beauty	7,068	3,570	79,506
	Flixster	2,341	2,956	26,173
	Douban	2,999	3,000	136,891
	MovieLens-1M	6,040	3,706	1,000,209
	MovieLens-100K	943	1,682	100,000
Meta-Test	Epinions	40,163	139,738	664,824
	Amazon-Sports	11,435	5,405	108,004

for the item ranking task and $\rho = 0.8$ for the rating prediction task. We set Beta$(0.5, 0.5)$ for Beta distribution in section 3.3.

For the searched GNN based CF models in **RGCF**, **RL-based**, **Random Search**, **AutoCF**, and **RS-10**, we have the following unified hyperparameter settings: For the rating prediction task, we follow the settings in previous work [43]. For the item ranking task, we adopt the BPR loss [32] and random negative sampler [10, 32] that randomly chooses one unobserved item as a negative sample for each observed user-item interaction every epoch and the maximum train epoch is set to 50. Meanwhile, for each training epoch, we adopt mini-batch [11] with batch size 4096. We adopt Adam [15] optimizer with 0.001 learning rate and 0.00005 weight decay.

For all manual design models, we follow the implementation on [43] for rating prediction task and we keep the same hyperparameters as GNN based CF model on on item ranking task for a fair comparison.

Implementation Settings. We implement all NAS models with PyTorch [29] and manual design models using the RecBole framework [58]. We run all experiments utilizing Intel(R) Xeon(R) Gold 6230R CPU @ 2.10GHz and GeForce RTX 3090 as the experimental environment.

4.2 Rating Prediction (RQ1)

The RMSE scores and search time of different methods on the item ranking task are reported in Table 3. From the results, we have the following observations:

- Our approach outperforms all the manual design models and other NAS baselines for both RGCF-T1 and RGCF-T5 architectures on rating prediction task. Let RGCF-T1 be called RGCF for brevity in the later part.
- Compared with RL-based baseline, RGCF get 0.46% reduction (from 0.8717 to 0.8677) of RMSE on Epinions and 0.54% reduction (from 0.9354 to 0.9303) on Amazon-Sports. For the search time, RGCF achieves 183x and 206x speed up for Epinions and Amazon-Sports dataset respectively, which proves the superiority of RGCF on searching performance and efficiency. In fact, RGCF only needs 88.9 (9.1+79.8) seconds and 18.7 (1.3+17.4) seconds, which contain the inference time of the neural retrieval model and the training time of the retrieved architecture, to get a well-performing architecture outperforming all baselines on Epinions and Amazon-Sports datasets, respectively.
- Random Search baseline consistently gets poor performance on both datasets demonstrating that RGCF excludes a lot of bad architectures from the large search space, which contains 103,680 achitectures, by transferring meta knowledge learned in the meta-training phase. Note that Random Search spend less time to search 10 architectures compared with RS-10, because the candidate architectures in the shrunk seach space are more complex and need more time to optimize.
- The AutoCF baseline outperforms other baselines by searching 300 architectures on a search space without considering GNNs. However, our approach can further improve the performance on both datasets with 1,025.5x and 2,998.8x speed up on search efficiency.
- Compared with RS-10, RGCF also achieves performance and efficiency improvements with equivalent dataset-architecture-performance triples on meta-train datasets. In fact, RGCF and RS-10 are two orthogonal directions: RS-10 focuses on shrinking the search space while RGCF focuses on retrieving architecture from the search space.

4.3 Item Ranking (RQ1)

The Recall@20 results and search time on the item ranking task are reported in Table 3. From the table, the following observations can be obtained:

- Similar to the results of rating prediction, our approach outperforms all baselines for both RGCF-T1 and RGCF-T5 on the item ranking task, which further demonstrates the generality of our method.
- Compared with RL-based baseline, RGCF improves 10.6% and 7.4% of Recall@20 on Epinions and Amazon-sports. As for the search time, RGCF achieves 93.6x and 61.7x speed up for Epinions and Amazon-Sports dataset. As shown in Figure 2 and Table 3, we can also observe that the RL-based method needs 196.6 hours to search a model for a larger dataset Epinions on item ranking task, which is unacceptable. Meanwhile, RGCF only needs 2.1 hours to get the final trained model which is acceptable in real applications.

Table 3: Performance and the search time of our model and all baselines on 2 new datasets. The search time denotes the time to search a trained model on the new dataset. Note the search time of our approach contains two parts: The inference time of neural retrieval model (including extraction time on the new dataset) and the training time of the retrieved architectures.

Model	Epinions				Amazon-Sports			
	Search Time	**RMSE**	**Search Time**	**Recall@20**	**Search Time**	**RMSE**	**Search Time**	**Recall@20**
MF [17]	-	0.9945 ± 0.0000	-	0.0344 ± 0.0009	-	0.9882 ± 0.0007	-	0.0911 ± 0.0081
NCF [10]	-	1.0070 ± 0.0055	-	0.0344 ± 0.0007	-	0.9342 ± 0.0008	-	0.0636 ± 0.0011
NGCF [41]	-	1.1437 ± 0.0240	-	0.0290 ± 0.0002	-	1.0668 ± 0.0038	-	0.0636 ± 0.0011
LightGCN [9]	-	0.9926 ± 0.0001	-	0.0321 ± 0.0003	-	0.9705 ± 0.0003	-	0.0776 ± 0.0006
DGCF [42]	-	1.6800 ± 0.2272	-	0.0321 ± 0.0003	-	0.9894 ± 0.0000	-	0.0922 ± 0.0010
RL-based [6]	16,263s	0.8717 ± 0.0014	196.6h	0.0498 ± 0.0010	3,858s	0.9354 ± 0.0027	31,165s	0.0897 ± 0.0005
RS-10 [43]	1,353s	0.8729 ± 0.0014	28.2h	0.0221 ± 0.0005	364s	0.9327 ± 0.0006	21,148s	0.0226 ± 0.0030
Random Search	559s	0.8942 ± 0.0032	9.73h	0.0298 ± 0.0008	182s	0.9442± 0.0017	3,298s	0.0578 ± 0.0016
AutoCF [5]	91,171s	0.8699 ± 0.0022	49.8h	0.0347± 0.0005	56,078s	0.9321± 0.0014	58,505s	0.0697 ± 0.0012
RGCF-T1 (ours)	9.1s+79.8s	0.8677 ± 0.0011	9.1s+2.1h	0.0551 ± 0.0011	1.3s+17.4s	**0.9303 ± 0.0011**	1.3s+503s	0.0963 ± 0.0022
RGCF-T5 (ours)	9.1s+426s	**0.8611 ± 0.0005**	9.1s+8.1h	**0.0640 ± 0.0014**	1.3s+123s	**0.9303 ± 0.0011**	1.3s+2326s	**0.1004 ± 0.0004**

- Compared with AutoCF, RGCF also gets 23.7x and 116x reduction in search time while achieving 58.8% and 38.2% performance improvements for both datasets respectively, which further demonstrates the superiority of RGCF on search efficiency.
- RS-10 performs poorly on item ranking task. A possible explanation is that the shrunk search space is pruned on rating prediction task. Thus, the candidate architectures in the shrunk space are not suitable for item ranking task, which further demonstrate the generalization of RGCF.

As mentioned in introduction and similar to [14, 18, 36], the time of meta database construction and the meta-training phase is offline and only once, so it can be neglected considering that we can amortize the cost to all query datasets when the method is deployed for real-world use. On the other hand, the extraction time of meta-features for new datasets is online, and as shown in Table 4, it could be extracted quickly for our neural retrieval approach.

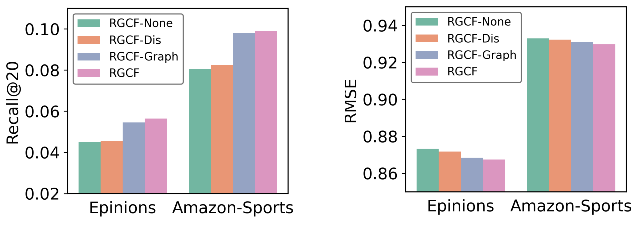
Table 4: Time of extracting different type meta-feature.

Model	**Epinions**	**Amazon-Sports**
Distribution	0.302s	0.002s
Graph	8.651s	1.125s

4.4 Ablation Study (RQ2 & RQ3)

4.4.1 Analysis of Meta-features. To demonstrate the importance of meta-features, we design three ablated models. 1) **RGCF-None** removes the dataset encoder and only considers learning a shared architecture encoder between different datasets. 2) **RGCF-Dis** only consider distribution-level meta features. 3) **RGCF-Graph** only consider graph-level meta-features. Figure 4 presents the results of our model and three ablated models. Through the experimental results, we can conclude that the performance gain of RGCF can be mainly attributed to the graph-level meta-features and the distribution level meta-features as a supplement can further benefit our model.

To further demonstrate the effectiveness of meta-features, we sample 10 anchored architectures from the whole search space



(a) Item Ranking **(b) Rating Prediction**

Figure 4: The comparisons of RGCF and its variants on meta-features.

and adopt the metric spearman's ranking correlation coefficient (SRCC) [35, 56], which measures the consistency with the real performance ranking and the predicted ranking of these anchored architectures. As shown in Figure 5, RGCF gets the highest SRCC compared to the other three variants, i.e., our proposed meta-feature tends to rank the whole search space in the right order. We can also observe that RGCF gets the most SRCC improvement by graph-level meta-features, which demonstrates that our proposed meta-features (especially the graph-level) capture the correlation between recommendation datasets and GNN architectures, thus better ability of performance prediction, which guarantees the performance of the proposed architecture retrieval method.

In summary, it is necessary to take advantage of different level of meta-features to retrieve best-fitted architecture for query datasets.

4.4.2 Analysis of Ranking Loss. To validate the effectiveness of Ranking loss, we design an ablated model: **RGCF w/o Rank** optimize the retrieval model by mse loss. As shown in Figure 6, RGCF outperforms RGCF w/o Rank for both tasks, which demonstrate ranking loss can improve the performance of retreived architecture by avoid performance calibration problem.

4.4.3 Analysis of Task-Level Data Augmentation. To validate the effectiveness of task-level data augmentation, we design an ablated model: **RGCF w/o TDA** remove the task-level mixup. As shown in Figure 6, RGCF outperforms RGCF w/o TDA on all the new datasets for both tasks showing that the task level data augmentation improve generalization of retrieval model on the phase of meta-test.

Table 5: Best retrieved architectures on the two new datasets.

Task	Dataset	Model	d	$m(\cdot)$	$f(\cdot)$	$\sigma(\cdot)$	L	$g(\cdot)$	K	$c(\cdot)$	$p(\cdot)$	Performance
Rating Prediction	Epinions	RGCF-T5	256	Identity	None	Sigmoid	3	Mean	4	Mean	Concat+MLP	0.08611 ± 0.0005
		RGCF-T1	64	Identity	None	Sigmoid	3	Concat	4	Mean	Sum+MLP	0.08677 ± 0.0011
	Amazon-Sports	RGCF-T5	256	Identity	GCN	Sigmoid	2	Concat	4	Mean	Concat+MLP	0.09303 ± 0.0011
		RGCF-T1	256	Identity	GCN	Sigmoid	2	Concat	4	Mean	Concat+MLP	0.09303 ± 0.0011
Item Ranking	Epinions	RGCF-T5	64	Identity	GAT	Sigmoid	3	Stack	4	Mean	Dot Product	0.0640 ± 0.00014
		RGCF-T1	256	Identity	GAT	Sigmoid	3	Concat	4	Mean	Dot Product	0.0551 ± 0.0011
	Amazon-Sports	RGCF-T5	256	Identity	GAT	Sigmoid	3	Stack	3	Att	Dot Product	0.1004 ± 0.0004
		RGCF-T1	128	Identity	GAT	Sigmoid	3	Stack	3	Att	Dot Product	0.0963 ± 0.0022

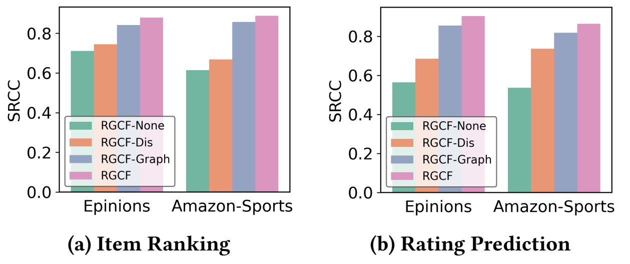


Figure 5: The comparisons of correlation between GNN architectures and datasets on different meta-features.

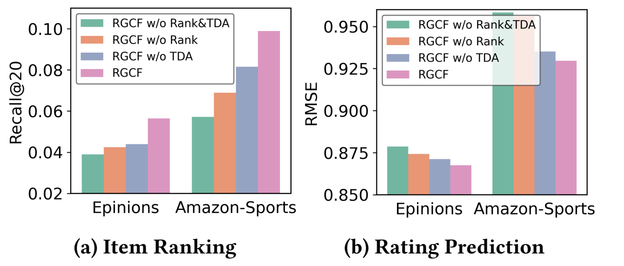


Figure 6: The comparisons of RGCF and its two variants: no ranking loss and no task-level data augmentation.

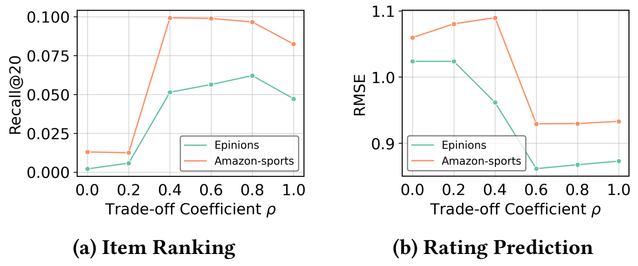


Figure 7: Results of the parameter sensitivity analysis.

Thus, it is important to utilize task-level data augmentation with limited meta-train datasets. To further validate the effectiveness of both ranking loss and task-level data augmentation, we further consider an ablated model: **RGCF w/o Rank&TDA** remove both modules. As shown in Figure 6, RGCF w/o Rank&TDA gets the worst performance for both tasks, which indicates that both ranking loss and task-level data augmentation can improve the performance of retrieved architecture and the combination of both modules will achieve the best performance.

4.5 Case Study: Task-adaptive retrieved architectures (RQ4)

We present the retrieved architecture by RGCF-T1 and RGCF-T5 for each task on all query datasets in Table 5. From the presented architectures, we have the following observations:

- When we increase the retrieval number of architectures from 1 to 5, the performance of retrieved architectures will further be improved. It indicates that there are a series of well-performing models in the search space, which is aligned with the observation in [43]. However, we get this conclusion with the neural retrieval approach on the other line.
- The best models are not the same for different datasets and different tasks which further validates the necessity of task-adaptive architecture retrieval. We also find that GAT-based architectures achieve good performance on item ranking tasks for all datasets which is ignored by researchers all along. A possible explanation is that the item ranking task fits a more complex model.

4.6 Parameter Sensitivity Analysis

Finally, we perform a sensitivity analysis of the trade-off coefficient parameter ρ between pairwise loss and listwise loss on all tasks for both two new datasets. As shown in Fig. 7, a suitable range of ρ is 0.4 to 0.8. When we fully use listwise loss, i.e., $\rho = 0$, the retrieved model shows poor performance. Meanwhile, fully using listwise loss, i.e., $\rho = 1$ will also lead to a performance drop. Thus, it is important to balance the weights of pairwise loss and listwise loss.

5 CONCLUSION

In this work, we try to address the problem of how to get a well-preforming GNN architecture rapidly for new recommendation scenarios. We propose a novel neural retrieval approach to return a well-performing architecture directly based on meta-learning. We propose to optimize the neural retrieval framework by ranking loss to address performance calibration problem and adopt a novel task-level data augmentation to further improve the effectiveness of our approach in case of limited recommendation datasets. The experimental results on two new datasets for both rating prediction and item ranking demonstrate the evidence of the proposed RGCF in terms of effectiveness and efficiency.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (No. 62192784, U1936104, U20B2045, 62172052, 62002029). We thank all anonymous reviewers for their constructive comments.

REFERENCES

[1] Kaidi Cao, Jiaxuan You, Jiaju Liu, and Jure Leskovec. 2023. AutoTransfer: AutoML with Knowledge Transfer - An Application to Graph Neural Networks. In *ICLR*.

[2] Jin Yao Chin, Yile Chen, and Gao Cong. 2022. The Datasets Dilemma: How Much Do We Really Know About Recommendation Datasets?. In *WSDM*. 141–149.

[3] Thomas Elsken, Benedikt Staffler, Jan Hendrik Metzen, and Frank Hutter. 2020. Meta-learning of neural architectures for few-shot learning. In *CVPR*. 12365–12375.

[4] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*. 1126–1135.

[5] Chen Gao, Quanming Yao, Depeng Jin, and Yong Li. 2021. Efficient Data-specific Model Search for Collaborative Filtering. In *KDD*. 415–425.

[6] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. 2021. Graph neural architecture search. In *IJCAI*. 1403–1409.

[7] Albert Gordo, Jon Almazan, Jerome Revaud, and Diane Larlus. 2017. End-to-end learning of deep visual representations for image retrieval. *International Journal of Computer Vision* 124, 2 (2017), 237–254.

[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*. 1024–1034.

[9] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *SIGIR*. 639–648.

[10] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *TheWebConf*. 173–182.

[11] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. 2012. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on* 14 (2012), 2.

[12] Daniel S. Hirschberg, Ashok K. Chandra, and Dilip V. Sarwate. 1979. Computing connected components on parallel computers. *Commun. ACM* 22 (1979), 461–464.

[13] ZHAO Huan, YAO Quanming, and TU Weiwei. 2021. Search to aggregate neighborhood for graph neural network. In *ICDE*. 552–563.

[14] Wonyong Jeong, Hayeon Lee, Geon Park, Eunyoung Hyung, Jinheon Baek, and Sung Ju Hwang. 2021. Task-Adaptive Neural Network Search with Meta-Contrastive Learning. In *NeurIPS*. 21310–21324.

[15] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[16] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

[17] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*. 426–434.

[18] Hayeon Lee, Eunyoung Hyung, and Sung Ju Hwang. 2021. Rapid neural architecture search by learning to generate graphs from datasets. In *ICLR*.

[19] Joonseok Lee, Samy Bengio, Seungyeon Kim, Guy Lebanon, and Yoram Singer. 2014. Local collaborative ranking. In *TheWebConf*. 85–96.

[20] Zelong Li, Jianchao Ji, Yingqiang Ge, and Yongfeng Zhang. 2022. AutoLossGen: Automatic Loss Function Generation for Recommender Systems. In *SIGIR*.

[21] Dongze Lian, Yin Zheng, Yintao Xu, Yanxiong Lu, Leyu Lin, Peilin Zhao, Junzhou Huang, and Shenghua Gao. 2019. Towards fast adaptation of neural architectures with meta learning. In *ICLR*.

[22] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. Darts: Differentiable architecture search. In *ICLR*.

[23] Tie-Yan Liu et al. 2009. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval* 3 (2009), 225–331.

[24]]DBLP:conf/sigir/LiuYZHZWZHWS23 Yaoqi Liu, Cheng Yang, Tianyu Zhao, Hui Han, Siyuan Zhang, Jing Wu, Guangyu Zhou, Hai Huang, Hui Wang, and Chuan Shi. [n. d.]. GammaGL: A Multi-Backend Library for Graph Neural Networks. In *SIGIR*.

[25] Bhaskar Mitra and Nick Craswell. 2017. Neural models for information retrieval. *arXiv preprint arXiv:1705.01509* (2017).

[26] Mark EJ Newman. 2003. Mixing patterns in networks. *Physical review E* 67 (2003), 026126.

[27] Shuzi Niu, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. 2012. Top-k learning to rank: labeling, ranking and evaluation. In *SIGIR*. 751–760.

[28] Namyong Park, Ryan A. Rossi, Nesreen Ahmed, and Christos Faloutsos. 2023. MetaGL: Evaluation-Free Selection of Graph Learning Models via Meta-Learning. In *ICLR*.

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[30] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient neural architecture search via parameters sharing. In *ICML*. 4095–4104.

[31] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-based neural networks for user response prediction. In *ICDM*. 1149–1154.

[32] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618* (2012).

[33] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. 2020. Neural collaborative filtering vs. matrix factorization revisited. In *RecSys*. 240–248.

[34] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *TheWeConf*. 285–295.

[35] Patrick Schober, Christa Boer, and Lothar A Schwarte. 2018. Correlation coefficients: appropriate use and interpretation. *Anesthesia & Analgesia* 126 (2018), 1763–1768.

[36] Gresa Shala, Thomas Elsken, Frank Hutter, and Josif Grabocka. 2023. Transfer NAS with Meta-learned Bayesian Surrogates. In *ICLR*.

[37] Albert Shaw, Wei Wei, Weiyang Liu, Le Song, and Bo Dai. 2019. Meta architecture search. In *NeurIPS*. 11225–11235.

[38] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *ICLR*.

[39] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. 2019. Manifold mixup: Better representations by interpolating hidden states. In *ICML*. 6438–6447.

[40] Ricardo Vilalta and Youssef Drissi. 2002. A perspective view and survey of meta-learning. *Artificial intelligence review* 18, 2 (2002), 77–95.

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

[42] Xiang Wang, Hongye Jin, An Zhang, Xiangnan He, Tong Xu, and Tat-Seng Chua. 2020. Disentangled Graph Collaborative Filtering. In *SIGIR*. 1001–1010.

[43] Zhenyi Wang, Huan Zhao, and Chuan Shi. 2022. Profiling the Design Space for Graph Neural Networks Based Collaborative Filtering. In *WSDM*. 1109–1119.

[44] Lanning Wei, Zhiqiang He, Huan Zhao, and Quanming Yao. 2023. Search to Capture Long-range Dependency with Stacking GNNs for Graph Classification. In *TheWebConf*. 588–598.

[45] Lanning Wei, Huan Zhao, and Zhiqiang He. 2022. Designing the Topology of Graph Neural Networks: A Novel Feature Fusion Perspective. In *TheWebConf*. 1381–1391.

[46] Lanning Wei, Huan Zhao, Quanming Yao, and Zhiqiang He. 2021. Pooling architecture search for graph classification. In *CIKM*. 2091–2100.

[47] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. 2020. Graph neural networks in recommender systems: a survey. *ACM Computing Surveys (CSUR)* (2020).

[48] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. 2008. Listwise approach to learning to rank: theory and algorithm. In *ICML*. 1192–1199.

[49] Yongqin Xian, Bernt Schiele, and Zeynep Akata. 2017. Zero-shot learning-the good, the bad and the ugly. In *CVPR*. 4582–4591.

[50] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *ICLR*.

[51] Huaxiu Yao, Linjun Zhang, and Chelsea Finn. 2021. Meta-Learning with Fewer Tasks through Task Interpolation. In *ICLR*.

[52] Quanming Yao, Xiangning Chen, James T Kwok, Yong Li, and Cho-Jui Hsieh. 2020. Efficient neural interaction function search for collaborative filtering. In *TheWebConf*. 1660–1670.

[53] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD*. 974–983.

[54] Jiaxuan You, Zhitao Ying, and Jure Leskovec. 2020. Design space for graph neural networks. In *NeurIPS*. 17009–17021.

[55] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. 2017. Deep sets. In *NeurIPS*. 3391–3401.

[56] Yongqi Zhang, Zhanke Zhou, Quanming Yao, and Yong Li. 2022. KGTuner: Efficient Hyper-parameter Search for Knowledge Graph Learning. *arXiv preprint arXiv:2205.02460* (2022).

[57] Tianyu Zhao, Cheng Yang, Yibo Li, Quan Gan, Zhenyi Wang, Fengqi Liang, Huan Zhao, Yingxia Shao, Xiao Wang, and Chuan Shi. 2022. Space4hgnn: a novel, modularized and reproducible platform to evaluate heterogeneous graph neural network. In *SIGIR*. 2776–2789.

[58] Wayne Xin Zhao, Yupeng Hou, Xingyu Pan, Chen Yang, Zeyu Zhang, Zihan Lin, Jingsen Zhang, Shuqing Bian, Jiakai Tang, Wenqi Sun, et al. 2022. RecBole 2.0: Towards a More Up-to-Date Recommendation Library. *arXiv preprint arXiv:2206.07351* (2022).

[59] Xiangyu Zhao, Haochen Liu, Wenqi Fan, Hui Liu, Jiliang Tang, and Chong Wang. 2021. Autoloss: Automated loss function search in recommendations. In *KDD*. 3959–3967.

[60] Barret Zoph and Quoc V Le. 2017. Neural architecture search with reinforcement learning. In *ICLR*.

[61] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *CVPR*. 8697–8710.