TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 13/16 pp769–785
DOI: 10.26599/TST.2022.9010042
Volume 28, Number 4, August 2023

Neighbor Library-Aware Graph Neural Network for Third Party Library Recommendation

Ying Jin, Yi Zhang, and Yiwen Zhang*

Abstract: Modern software development has moved toward agile growth and rapid delivery, where developers must meet the changing needs of users instantaneously. In such a situation, plug-and-play Third-Party Libraries (TPLs) introduce a considerable amount of convenience to developers. However, selecting the exact candidate that meets the project requirements from the countless TPLs is challenging for developers. Previous works have considered setting up a personalized recommender system to suggest TPLs for developers. Unfortunately, these approaches rarely consider the complex relationships between applications and TPLs, and are unsatisfactory in accuracy, training speed, and convergence speed. In this paper, we propose a new end-to-end recommendation model called Neighbor Library-Aware Graph Neural Network (NLA-GNN). Unlike previous works, we only initialize one type of node embedding, and construct and update all types of node representations using Graph Neural Networks (GNN). We use a simplified graph convolution operation to alternate the information propagation process to increase the training efficiency and eliminate the heterogeneity of the app-library bipartite graph, thus efficiently modeling the complex high-order relationships between the app and the library. Extensive experiments on large-scale real-world datasets demonstrate that NLA-GNN achieves consistent and remarkable improvements over state-of-the-art baselines for TPL recommendation tasks.

Key words: Third-Party Library (TPL); TPL recommendation; Graph Neural Network (GNN); bipartite graph

1 Introduction

With the rapid development of the Internet and electronic devices, mobile application software has been attracting the attention of a growing number of developers, resulting in the creation of hundreds of excellent products. According to statistics, in the first quarter of 2021, Google Play and Apple App Store have approximately 3.48 and 2.22 million applications[†], respectively. Meanwhile, users downloaded 27.6 and 8.1 billion applications from Google Play and Apple App Store, respectively[‡]. This phenomenon is not only an opportunity but also a significant difficulty for mobile app developers. Mobile app developers must accelerate their development speed to fulfill the ever-changing demands of consumers[1]. Therefore, many developers construct apps using publicly available Third-Party Libraries (TPLs)[2], which prevent the repeated implementation of certain functions and allow for fast delivery[3].

TPLs have undeniably become an indispensable part of mobile application development. Recent studies[4, 5]

• Ying Jin is with the School of Artificial Intelligence and Big Data, Hefei University, Hefei 230601, China. E-mail: jiny@hfuu.edu.cn.
• Yi Zhang and Yiwen Zhang are with the School of Computer Science and Technology, Anhui University, Hefei 230601, China. E-mail: zhangyi.ahu@gmail.com; zhangyiwen@ahu.edu.cn.
* To whom correspondence should be addressed.
Manuscript received: 2022-06-13; revised: 2022-09-01; accepted: 2022-09-26

† https://www.statista.com/statistics/276623/
‡ https://www.statista.com/statistics/695094/

have shown that mobile applications in Google Play use an average of 11.81 TPLs, which also lead to an explosion in the number of TPLs[§]. However, this trend indicates that developers often need to spend additional time searching TPLs that meet their needs for their applications[6]. Moreover, different TPL versions, interfaces, and compatibility issues will interfere with the choices of developers, which will certainly slow down application development[7]. Many pioneering works[8] attempted to leverage the data-driven machine learning paradigm to recommend suitable TPLs for mobile application development to address the aforementioned problem. For example, LibRec[8] uses Collaborative Filtering (CF)[9] and association rule mining to find similar TPLs for developers. LibSeek[5] empirically finds the fairness bias[10] in TPL recommendation and combines Matrix Factorization (MF)[11, 12] and adaptive weight mechanism for recommendations. GRec[13] revisits the TPL recommendation with a graph structure, which models the relationship between mobile applications and TPLs as an application-library graph, and applies Graph Convolution Network (GCN)[14] to model high-order collaborative signals[15] between applications and TPLs explicitly to achieve state-of-the-art recommendation performance.

However, GRec uses GCN to iteratively propagate node information to obtain high-quality node embeddings, which still has several drawbacks. On the one hand, GCNs that perform well in node classification tasks are not ideal for CF tasks, according to some relevant studies[16, 17]. The core reason is that the raw data only contain the mapped ID information of the nodes (mobile app and library) and disregard information-rich features. Such sparse raw data will lead to feature transformation and nonlinear activation in GCNs, severely limiting the normal training of the model and the accuracy of downstream recommendation tasks. On the other hand, GRec attempts to use the app-library bipartite graph to learn high-quality embeddings. However, app and library are different types of nodes in the app-library graph because they have different semantic information and are in different hidden feature spaces[18]. In the app-library graph, the neighboring nodes (first-order neighbors) of an app node are the library nodes used by the app, and it's second-order neighbors are some similar apps that have used the same library. The simple scheme of aggregating information from neighbor nodes may affect the performance of the recommendation model, due to the inherent heterogeneity of bipartite graph[19].

In order to solve the problems listed above, we propose a novel Neighbor Library Aware Graph Neural Network for third-party library recommendation, called NLA-GNN. Specifically, we use the simplified GCN structure[17, 20] to encode the high-order connectivity between mobile apps and libraries explicitly in a rapid way. We empirically remove the redundant nonlinear activations and feature transformation operations in GRec to accelerate the model training without compromising model performance. Different from GRec, we split the complete app-library bipartite graph into two independent graphs: the **app library graph** and the **library app graph**, respectively. We only initialize one type of node feature (i.e., embeddings) of the mobile app/library, and propagate information in an alternating and iterative manner on two separate graphs to eliminate the negative impact caused by the heterogeneity of the bipartite graph. Extensive experiments on large-scale real-world datasets demonstrate that NLA-GNN achieves consistent and remarkable improvements over state-of-the-art baselines for TPL recommendation tasks. Moreover, NLA-GNN has significant advantages in training efficiency, convergence speed, and model size by removing unnecessary operations and learning only one type of feature representation.

To summarize, the contributions of our work are as follows.

- We propose a novel end-to-end model named NLA-GNN for TPL recommendation, which uses a simplified GCN architecture to learn high-quality mobile app and library embeddings.
- To eliminate the negative impact of heterogeneity of the app-library bipartite graph, we split the graph into two independent graphs and model the high-order connectivity between app and library on both graphs using one type of node.
- We conduct extensive experiments on large-scale and real-world dataset MALib. The experimental results show that compared with GRec and other state-of-the-art TPL recommendation models, NLA-GNN can achieve consistent and remarkable improvements and has advantages in training speed, convergence speed, and model size.

[§] In this paper, "application", "mobile application", "app", and "mobile app" all mean the same thing, and "library" is the same as "TPL".

2 Preliminary

In this part, we briefly formalize the TPL recommendation problem. To make our descriptions easier to read and understand, we present the definitions of some of the symbols used (Table 1). Then, we recapitulate the GRec[13] model, which is a novel and state-of-the-art solution for the TPL recommendation task.

2.1 TPL recommendation

Definition 1 (TPL recommendation) The TPL recommendation scenario has a set of M mobile apps $\mathcal{A} = \{A_1, A_2, \ldots, A_M\}$ and a set of N TPLs $\mathcal{L} = \{L_1, L_2, \ldots, L_N\}$. Following previous works[5, 13], we regard the relationship between mobile apps and libraries as binary implicit feedback. The app-library graph $\mathcal{G}$ can be described by an interaction matrix $\boldsymbol{R} \in \mathbf{R}^{M \times N}$ according to the historical app-library interactions: for app A_i and library L_j, if a mobile app-library usage record exists, then an edge exists between app A_i and library L_j on graph $\mathcal{G}$ and $R_{ij} = 1$. Moreover, the adjacency matrix of the app-library bipartite graph can be defined as $\boldsymbol{A} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{R} \\ \boldsymbol{R}^{\mathrm{T}} & \boldsymbol{0} \end{bmatrix}$. We can then formulate the TPL recommendation task as follows:

- **Input:** App-library interaction bipartite graph $\mathcal{G}$ and interaction matrix $\boldsymbol{R}$.
- **Output:** Top-N TPLs that the app is most likely to use in the future.

Table 1 Notations and explanations.

Notation	Explanation
$\mathcal{A}/\mathcal{L}$	Set of mobile apps/TPLs
M/N	Number of mobile apps/TPLs
$\mathcal{G}$	Interaction graph of mobile apps and TPLs
V/E	Set of nodes/edges with respect to graph $\mathcal{G}$
$\boldsymbol{R}$	Interaction matrix of mobile apps and TPLs
$\boldsymbol{A}$	Adjacency matrix of the interaction matrix
$\boldsymbol{D}$	Diagonal degree matrix of the adjacency matrix
$\widetilde{\boldsymbol{A}}$	Graph Laplacian adjacency matrix
$\boldsymbol{e}_A^{(k)}/\boldsymbol{e}_L^{(k)}$	Embedding of app A/library L in the k-th layer
$\boldsymbol{E}_{\mathcal{A}}^{(k)}/\boldsymbol{E}_{\mathcal{L}}^{(k)}$	Embedding lookup table of apps/TPLs
$\mathcal{N}_A/\mathcal{N}_L$	First-order neighbor set of app A/TPL L
$\boldsymbol{W}^{(k)}$	Trainable layer-specific transformation matrix
K	Total number of alternating information distillation layers
σ	Nonlinear activation function (e.g., ReLU)
λ	Coefficient controlling the L_2 norm
$\hat{R}_{AL}$	Predicted score between app A and TPL L

2.2 Recap GRec

GRec[13] attempts to use the powerful high-order modeling capabilities of graph neural network to establish connections between mobile apps and TPLs. In brief, GRec is an end-to-end model based on the graph structure, and its training process can be divided into four parts: representation initialization, information distillation, information aggregation, and prediction.

2.2.1 Representation initialization

Similar to the traditional recommendation models based on latent vectors[21, 22], neural networks[9, 23], and GCNs[15, 24], GRec projects each mobile app and TPL into different latent vector spaces. Formally, the embedding representations of App A and TPL L are $\boldsymbol{e}_A \in \mathbf{R}^d$ and $\boldsymbol{e}_L \in \mathbf{R}^d$, respectively, where d is the embedding size. From the perspective of representation learning, embedding representation realizes the transformation from a high-dimensional sparse feature vector to a low-dimensional dense feature vector[25]. Embedding representation provides a lower storage cost and higher expressive potential than those of one-hot encoding.

2.2.2 Information distillation

The core of GRec is to propagate information iteratively on the app-library bipartite graph. Specifically, given app A_i and TPL L_j, the information distillation from L_j to A_i can be described as

$$\boldsymbol{s}_{L_j \to A_i} = \frac{1}{\sqrt{|\mathcal{N}_{A_i}||\mathcal{N}_{L_j}|}}(\boldsymbol{W}_1 \boldsymbol{e}_{L_j} + \boldsymbol{W}_2(\boldsymbol{e}_{L_j} \odot \boldsymbol{e}_{A_i})) \tag{1}$$

where $\mathcal{N}_{A_i}$ and $\mathcal{N}_{L_j}$ are the neighbor sets corresponding to A_i and L_j, respectively, $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are the trainable feature transformation matrices, and $\odot$ is element-wise product. Through the information distillation process, the embedding representation of the ego node A_i can be updated as

$$\boldsymbol{e}_{A_i}^{(1)} = \sigma\left(\boldsymbol{s}_{A_i \to A_i} + \sum_{L \in \mathcal{N}_{A_i}} \boldsymbol{s}_{L \to A_i}\right) \tag{2}$$

where σ is a nonlinear activation function (e.g., LeakyReLU). Formally, the first item is the self-connection of the ego node A_i, and the second item is the aggregation of information from all neighbor nodes. Considering the bipartite nature of the app-library graph, the information distillation process of the library nodes is similar and will not be repeated herein.

2.2.3 Information aggregation and prediction

After passing through multiple information distillation layers, all nodes receive information from neighboring nodes while their information is also passed beyond the multihop. Intuitively, the embeddings obtained from different propagation layers carry different semantic information. For example, the first-order neighbor information of app node A is its used TPLs, and its second-order neighbors are similar mobile apps that have used the same TPLs. GRec attempts to concatenate the embedding information of the same node from different layers to obtain the final embedding representation,

$$\boldsymbol{e}_{A_i} = \boldsymbol{e}_{A_i}^{(0)} \| \boldsymbol{e}_{A_i}^{(1)} \| \cdots \| e_{A_i}^{(K)};\ \boldsymbol{e}_{L_j} = \boldsymbol{e}_{L_j}^{(0)} \| \boldsymbol{e}_{L_j}^{(1)} \| \cdots \| \boldsymbol{e}_{L_j}^{(K)} \tag{3}$$

where K is the number of information distillation layers, and $\|$ is the concatenation operation. Finally, GRec uses the inner product to obtain the matching score of TPL L_j for mobile app A_i.

3 Our Proposed Model

We introduce the proposed NLA-GNN model in this section. We first provide a general overview of the model in Section 3.1, and the proposed NLA-GNN comprises three modules, namely unilateral representation initialization, alternating information distillation, and aggregated information prediction. We then comprehensively explain these modules in Sections 3.2, 3.4, and 3.5, respectively. To facilitate a macroscopic understanding of the NLA-GNN, we provide a mathematical expression in matrix form in Section 3.6. Finally, some deep analysis and discussion of the proposed model are given in Section 3.7.

3.1 Model overview

The model framework of NLA-GNN comprises three parts, as shown in Fig. 1, to learn high-quality vector representations of mobile apps and libraries in the same feature space.

- **Unilateral representation initialization:** In the model initialization stage, NLA-GNN selects one type of node in Graph $\mathcal{G}$ and creates hidden vectors for these nodes, which are the input of the next layer.
- **Alternating information distillation:** NLA-GNN first uses neighbor node information to construct the embedding representation of another kind of node, and then uses simplified GCNs to propagate node information on the graph $\mathcal{G}$ alternately.
- **Aggregated information prediction:** NLA-GNN aggregates the embeddings learned from multiple graph convolution layers to obtain the final representation of all types of nodes. NLA-GNN then calculates the matching score between the mobile app and library through the inner product.

3.2 Unilateral representation initialization

Different from most of the traditional recommendation models[13, 15, 17], NLA-GNN only initializes the embedding of any type of nodes on the bipartite graph $\mathcal{G}$ into a d-dimensional latent space. Specifically, we take the library set $\mathcal{L}$ as an example, where the latent vector $\boldsymbol{e}_{L_i} \in \mathbf{R}^d$ denotes the embedding of a library L_i. The complete library embedding lookup table can then be defined as

$$\boldsymbol{E}_{\mathcal{L}}^{(0)} = \{\boldsymbol{e}_{L_1}, \boldsymbol{e}_{L_2}, \ldots, \boldsymbol{e}_{L_N}\} \in \mathbf{R}^{N \times d} \tag{4}$$

From the perspective of representation learning, such embeddings are mapped into a specific feature space. In the learning process, the distance between similar embeddings will be close and that between dissimilar nodes will be gradually alienated[26] to achieve the purpose of CF.

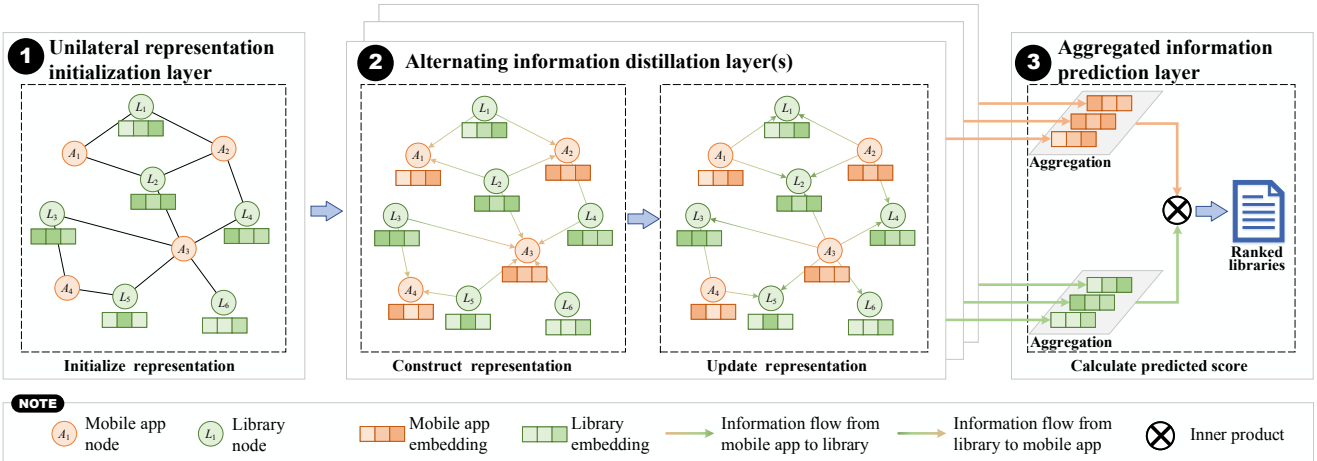


Fig. 1 Illustration of NLA-GNN model architecture.

After initializing one type of embedding (mobile app or library), uninitialized nodes can be described by their first-order neighbors in the bipartite graph seamlessly. Intuitively, we can infer the function of an app through the TPL information required by the application. For example, if an app needs to use TPLs, such as online chat and video communication, then the application should belong to social software.

3.3 Alternating information distillation

After obtaining the library embedding lookup table, we hope to use these known vectors to construct unknown terms. Specifically, given a mobile app node A_i, NLA-GNN first extracts the neighbor library nodes of node A_i on the entire graph to obtain a neighbor set $\mathcal{N}_{A_i}$. NLA-GNN then aggregates the information carried by all neighbor nodes as the representation of the central node A_i,

$$\boldsymbol{e}_{A_i}^{(1)} = \sum_{L \in \mathcal{N}_{A_i}} \frac{1}{\sqrt{|\mathcal{N}_{A_i}|}\sqrt{|\mathcal{N}_L|}} \boldsymbol{e}_L^{(0)} \tag{5}$$

where $\mathcal{N}_L$ is the neighbor set corresponding to L, and $\boldsymbol{e}_L^{(0)} \in \boldsymbol{e}_{\mathcal{L}}^{(0)}$ is the representation of library L initialized by the unilateral representation initialization layer. Instead of employing the vanilla GCN paradigm for information distillation, we use the simplified graph convolution for training, which discards redundant self-connection, feature transformation, and non-linear activation. Through such one-way information distillation, the neighbor information of the mobile app is aggregated, and a new representation $\boldsymbol{e}_{A_i}^{(1)}$ is formed for node A_i.

Subsequently, we return to the library side after constructing the embedding representation of all mobile app nodes. The first-order neighbors of the library include concealed information, which may be used to define the library itself (what sort of app utilizes this information). Therefore, we can consider the library and app as the same sort of nodes because app embedding is built through library embeddings. We can also update the representation of library nodes by aggregating information from neighboring app nodes,

$$\boldsymbol{e}_{L_j}^{(1)} = \sum_{A \in \mathcal{N}_{L_j}} \frac{1}{\sqrt{|\mathcal{N}_{L_j}|}\sqrt{|\mathcal{N}_A|}} \boldsymbol{e}_A^{(1)} \tag{6}$$

where $\boldsymbol{e}_A^{(1)}$ is the embedding of mobile app A constructed by Eq. (5).

Example: As shown in Fig. 2, we first randomly initialize the embedding representations of all library nodes on the app-library graph. Then, we take the app

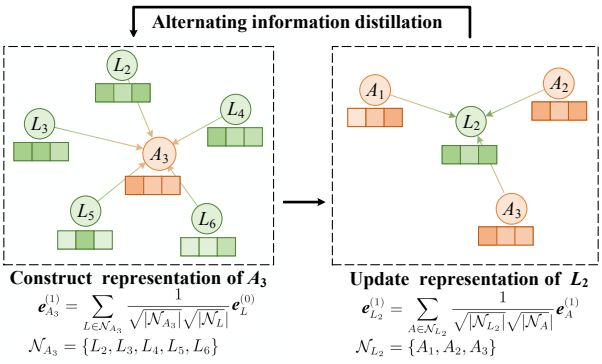


Fig. 2 Process of alternating information distillation centered on nodes A_3 and L_2 in Fig. 1.

node A_3 as an example, which aggregates the node embedding information from the neighboring nodes L_3, L_4, L_5, and L_6 to obtain its embedding representation. The viewpoint then switches to the TPL side, and the L_2 node receives the embedding representations from the neighboring nodes A_1, A_2, and A_3, which contain all the neighboring information of the app node.

Finding the structural consistency is easy based on Eqs. (5) and (6). Consistent with GRec, we attempt to stack multiple information distillation layers to propagate app and library information on the app-library graph alternately. Without loss of generality, the k-th ($k \geqslant 1$) alternating information distillation layer can be defined as follows:

$$\begin{aligned} \boldsymbol{e}_{A_i}^{(k)} &= \sum_{L \in \mathcal{N}_{A_i}} \frac{1}{\sqrt{|\mathcal{N}_{A_i}|}\sqrt{|\mathcal{N}_L|}} \boldsymbol{e}_L^{(k-1)}, \\ \boldsymbol{e}_{L_j}^{(k)} &= \sum_{A \in \mathcal{N}_{L_j}} \frac{1}{\sqrt{|\mathcal{N}_{L_j}|}\sqrt{|\mathcal{N}_A|}} \boldsymbol{e}_A^{(k)} \end{aligned} \tag{7}$$

Unlike GRec which completes the propagation over the entire app-library bipartite graph at once, NLA-GNN aggregates app and TPL embedding information in an alternating manner, which not only benefits from high-order information propagation but also effectively reduces the impact caused by the heterogeneity of the bipartite graph. In addition, the size and training speed of the model have advantages over the previous approaches because we only consider one type of node for graph convolution operation. We will verify this conclusion in the model analysis and experiment section.

3.4 Matrix forms

We perform the training process of NLA-GNN considering a single mobile app or TPL node in the previous subsections. In this Section, we provide a uniform matrix representation of all nodes to understand our model effectively. In the alternating information

distillation layer (s), we first need to define the adjacency matrix $\boldsymbol{A}$ and graph Laplacian matrix $\widetilde{\boldsymbol{A}}$ of the app-library bipartite graph,

$$\widetilde{\boldsymbol{A}} = \boldsymbol{D}^{-0.5}\boldsymbol{A}\boldsymbol{D}^{-0.5} \tag{8}$$

where $D_{ii} = |\mathcal{N}_i|$ is the diagonal degree matrix.

Notably, the complete app-library bipartite graph contains two sub-graphs, which we define as the **app library graph** and the **library app graph**. To facilitate performing alternating information distillation, we construct graph Laplacian matrices for the two subgraphs empirically,

$$\begin{aligned}\widetilde{\boldsymbol{A}}_{a-l} &= \widetilde{\boldsymbol{A}}[:M, N:],\\ \widetilde{\boldsymbol{A}}_{l-a} &= \widetilde{\boldsymbol{A}}[M:, :N]\end{aligned} \tag{9}$$

where $\widetilde{\boldsymbol{A}}_{a-l}$ and $\widetilde{\boldsymbol{A}}_{l-a}$ are the Laplacian matrices of the app library graph and the library app graph, respectively. $\widetilde{\boldsymbol{A}}[:M, N:]$ represents rows 1 to M and columns N to $M+N$ of matrix $\widetilde{\boldsymbol{A}}$. The construction process is shown in Fig. 3. The matrix forms of alternating information distillation layers are defined as follows:

$$\begin{aligned}\boldsymbol{E}_{\mathcal{A}}^{(k)} &= \widetilde{\boldsymbol{A}}_{a-l}\boldsymbol{E}_{\mathcal{L}}^{(k-1)},\\ \boldsymbol{E}_{\mathcal{L}}^{(k)} &= \widetilde{\boldsymbol{A}}_{l-a}\boldsymbol{E}_{\mathcal{A}}^{(k)}\end{aligned} \tag{10}$$

where $k \in [1, K]$, and Eq. (10) is equivalent to Eq. (7). NLA-GNN seamlessly realizes bipartite graph information distillation and high-order connectivity modeling by using only one type of embedding representation, and alternately performing graph convolution on app library and library app graphs.

3.5 Aggregated information prediction

By stacking K alternating information distillation layers, NLA-GNN finally obtains a set of mobile app embedding representations, $[\boldsymbol{E}_{\mathcal{A}}^{(1)}, \boldsymbol{E}_{\mathcal{A}}^{(2)}, \ldots, \boldsymbol{E}_{\mathcal{A}}^{(K)}]$, and a set of library embedding representations, $[\boldsymbol{E}_{\mathcal{L}}^{(1)}, \boldsymbol{E}_{\mathcal{L}}^{(2)}, \ldots, \boldsymbol{E}_{\mathcal{L}}^{(K)}]$, containing different semantic information and are used to guide the recommendation model on how to perform downstream TPL recommendation tasks.

Similar to GRec, we first aggregate each type of embeddings to obtain the final embedding representations. We then obtain the predicted scores of mobile app A_i and TPL L_j by inner product operation,

$$\begin{aligned}\hat{R}_{ij}^{\text{con}} &= \sigma\left(\sum_{k=1}^{K} \boldsymbol{e}_{A_i}^{(k)} \boldsymbol{e}_{L_j}^{(k)}\right), \text{ or}\\ \hat{R}_{ij}^{\text{sum}} &= \sigma\left(\sum_{k=1}^{K} \boldsymbol{e}_{A_i}^{(k)} \sum_{k=1}^{K} \boldsymbol{e}_{L_j}^{(k)}\right)\end{aligned} \tag{11}$$

where $\hat{R}_{ij}^{\text{con}}$ and $\hat{R}_{ij}^{\text{sum}}$ represent the predicted scores obtained by aggregation through concatenation[b] and sum, respectively, and σ is a nonlinear activation function (e.g., Sigmoid). Figuere 4 presents the process of constructing the final embedding representation and prediction using concatenation and sum operations for

[b] Note: The used derivative rules are $(\boldsymbol{e}_{A_i}^{(1)} \| \boldsymbol{e}_{A_i}^{(2)} \| \cdots \| \boldsymbol{e}_{A_i}^{(K)}) \cdot (\boldsymbol{e}_{L_j}^{(1)} \| \boldsymbol{e}_{L_j}^{(2)} \| \cdots \| \boldsymbol{e}_{L_j}^{(K)}) = \boldsymbol{e}_{A_i}^{(1)} \cdot \boldsymbol{e}_{L_j}^{(1)} + \boldsymbol{e}_{A_i}^{(2)} \cdot \boldsymbol{e}_{L_j}^{(2)} + \cdots + \boldsymbol{e}_{A_i}^{(K)} \cdot \boldsymbol{e}_{L_j}^{(K)}$.

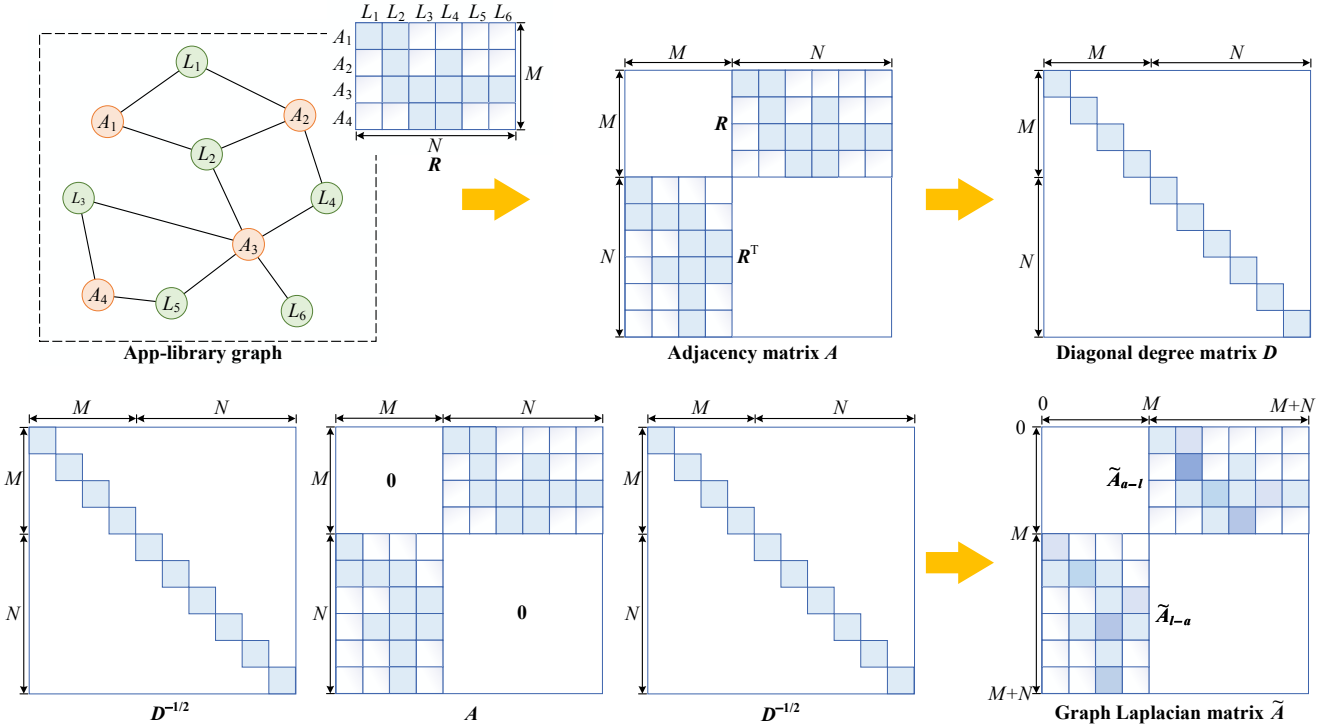
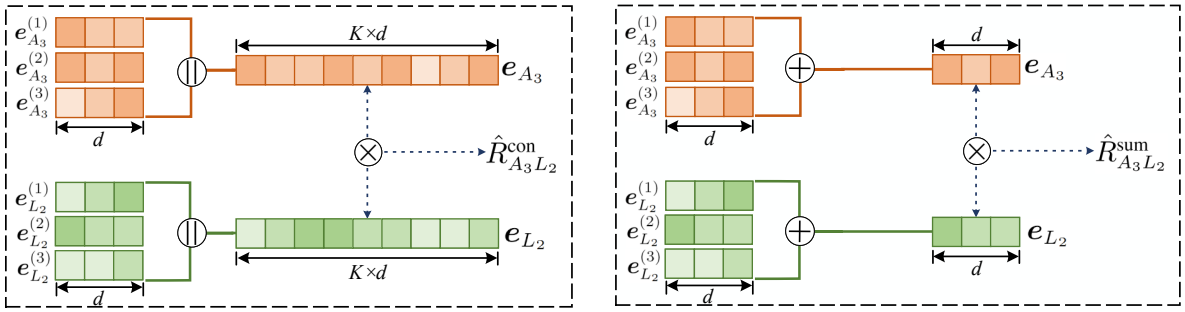


Fig. 3 Division process of app library graph and library app graph.



(a) Concatenation (b) Sum

Fig. 4 Process of aggregated information prediction centered on nodes A_3 and L_2 in Fig. 1 (activation function in Eq. (11) is omitted, and "||", "+", and "×" represent concatenation, sum, and inner product operations, respectively).

A_3 and L_2 nodes after three alternating information distillation layers.

3.6 Model training

In line with the training approach of traditional recommender systems[13, 15, 17], we divide mobile apps into random mini-batches in each training epoch. A batch is a set of app-TPL pairs that represent the history of app-library interactions. Simultaneously, we add a TPL that the app has not used as its negative sample for each pair.

To train the parameters in NLA-GNN (i.e., initialized single type embedding), a classical strategy is to introduce a pairwise loss function (such as Bayesian Personalized Ranking (BPR)[27]),

$$\max \prod_{\substack{L^+\sim U[\mathcal{N}_A]\\ L^-\sim U[\mathcal{L}/\mathcal{N}_A]}} p\left(\hat{R}_{AL^+} > \hat{R}_{AL^-}|\boldsymbol{e}_{\mathcal{L}}^{(0)}\right) = \sum_{\substack{L^+\sim U[\mathcal{N}_A]\\ L^-\sim U[\mathcal{L}/\mathcal{N}_A]}} -\ln\sigma\left(\hat{R}_{AL^+} - \hat{R}_{AL^-}\right) + \lambda||\boldsymbol{e}_{\mathcal{L}}^{(0)}||_2^2 \tag{12}$$

where L^+ and L^- represent the positive and negative TPL of app A, respectively, $U[\cdot]$ is uniform distribution, $\hat{R}_{AL^+}$ and $\hat{R}_{AL^-}$ are the predicted scores of app A for TPL L^+ and TPL L^- calculated by Eq. (11), respectively, σ is Sigmoid nonlinear activation function, and λ is a coefficient controlling the L_2 norm. Theoretically, BPR considers the relative order of observables (positive samples) and unobservables (negative samples) in the interaction, and assumes that positive samples should be assigned higher predicted scores compared to negative samples.

Example: Taking Fig. 1 as an example, the possible combination can be obtained as $\langle A_1, L_1, L_3\rangle$, where L_1 is the TPL used by A_1, which is regarded as a positive sample (L^+), while L_3 is not used by A_1, which is regarded as a negative sample (L^-).

3.7 Model analysis

3.7.1 Model size

In contrast to previous works[13], NLA-GNN utilizes only one type of embedding for model training. Taking the TPL nodes as an example, the size of the embedding table $\boldsymbol{E}_{\mathcal{L}}^{(0)}$ is N. Since we do not introduce any additional auxiliary modules, which leaves NLA-GNN with only $N \times d$ parameters while GRec has $(M+N)\times d + 2Kd^2$ model parameters because it has two types of embedding representations as well as multiple feature transformation matrices, where d is embedding size.

3.7.2 Time complexity

NLA-GNN is a TPL recommendation model based on GCN, whose training time overhead mainly comes from iterative graph convolution operations. Considering the size of the constructed app-library bipartite graph, we use a mini-batch strategy for training. We divide the training data in each epoch into $|E|/B$ groups with a predetermined batch-size B, and perform graph convolution operations on each group separately to update the embedding representations of all app and library nodes on the graph. In each batch, the graph convolution computation (Eq. (10)) must be completed first, and the time complexity to complete this step is $O(K|E|d)$. Finally, the overall time complexity of the aggregation process on mobile app-library bipartite graph is $O\left(K|E|^2d/B\right)$, where K, $|E|$, B, and d indicate the number of alternating information distillation layers, the number of interactions of app-library $\mathcal{G}$, the batch size, and embedding size, respectively. The time complexity of NLA-GNN is at the same level as that of the previous graph-based recommendation models.

3.7.3 Relation with GRec

GRec is a TPL recommendation model based on graph structure containing all the contents of basic GCN, including self-connection, feature transformation, and nonlinear activation (refer to Eqs. (1) and (2)). It is worth noting that some related studies[16, 17] have pointed out that nonlinear activation and feature transformation are not applicable to recommender systems, which may lead to training slowdown and performance loss. Inspired by these works, NLA-GNN discards the feature transformation process and disseminates node information in an iterative and alternating manner. For comparison, we rewrite the alternating information distillation process of NLA-GNN as follows:

$$\begin{aligned} \boldsymbol{e}_{A_i}^{(k)} &= \sum_{L\in\mathcal{N}_{A_i}} s_{L\to A_i} = \sum_{L\in\mathcal{N}_{A_i}} \frac{1}{\sqrt{|\mathcal{N}_{A_i}||\mathcal{N}_L|}}\boldsymbol{e}_L^{(k-1)}, \\ \boldsymbol{e}_{L_j}^{(k)} &= \sum_{A\in\mathcal{N}_{L_j}} s_{A\to L_j} = \sum_{A\in\mathcal{N}_{L_j}} \frac{1}{\sqrt{|\mathcal{N}_A||\mathcal{N}_{L_j}|}}\boldsymbol{e}_A^{(k)} \end{aligned} \quad (13)$$

Compared with the model structures of GRec (refer to Eqs. (1) and (2)), NLA-GNN eliminates the processes of self-connection, feature transformation, and nonlinear activation. Obviously, NLA-GNN can be regarded as a light GRec.

4 Experiment

In this section, we want to verify the validity of the proposed model through a series of comparative experiments and parametric analysis. We first present a detailed description of the experiments, including datasets, comparison models, and training and evaluation strategies.

4.1 Datasets

To maintain consistency with previous works[5, 13], we choose the MALib dataset[♮] to conduct our experiments. The MALib dataset is a publicly available real-world dataset that includes name information and interaction data with 61 722 android applications, 827 TPLs, and 725 502 app and library usage records. To ensure the quality of recommendations, we use a 10-core setting (i.e., filter out apps or libraries with less than ten interactions).

Consistent with the training strategy of traditional recommendation models, we divide the complete dataset into training and testing sets. Specifically, we randomly select r ($\in [1, 3, 5]$) interacted TPLs for each mobile app in the complete app-library interaction records as its corresponding testing set, and the rest as the training set. In the training process, we first train all models on the training set and subsequently calculate the matching scores of each app for all libraries. Next, we filter and sort the list of matching scores, and select the top-N ($\in [5, 10]$) libraries as the final recommendation list. For the convenience of follow-up, we name the datasets divided by the three training/testing sets MALib#1, MALib#2, and MALib#3. The statistical information of each dataset is shown in Table 2.

4.2 Baselines

To demonstrate the effectiveness of proposed model, we compare NLA-GNN with the following baselines:

- **Popular:** This method mechanically recommends the most popular TPLs for each mobile app without considering personalization.
- **MF[27]:** The method is standard matrix factorization for CF task, which the interaction matrix is decomposed into independent latent vectors, and the matching score is obtained by the inner product.
- **LibRec[8]:** The method is a hybrid approach that combines association rule mining and CF for the TPL recommendation task.
- **LibSeek[5]:** The method is a state-of-the-art approach for TPL recommendation task. It additionally considers the popularity bias and long-tail effect of historical interactions on top of the original MF model.
- **GRec[13]:** The method is a state-of-the-art graph-based method for TPL recommendation task. It propagates the information of mobile app and library nodes by stacking multiple graph convolution layers, thus capturing the high-order relationships between the

[♮] https://github.com/malibdata/MALib-Dataset

Table 2 Statistics of the MALib datasets.

Dataset	Number of apps	Number of TPLs	Size of training sets	Size of testing sets	Sparsity
MALib#1	31 421	727	499 107	31 091	0.978 15
MALib#2	31 421	727	436 956	93 242	0.980 87
MALib#3	31 421	727	374 771	155 427	0.983 59

Note: Sparsity refers to the proportion of zero elements in the interaction matrix $\boldsymbol{R}$. Considering a recommendation scenario with M mobile apps and N TPLs, with $|E|$ observed interactions between them, the sparsity is defined as $1 - |E|/(M \times N)$.

mobile app and library.

Notably, we propose two schemes of information aggregation for NLA-GNN in Section 3.5. That is, we use concatenation and sum operations to obtain the final node representations, which are named NLA-GNN$_{\text{cat}}$ and NLA-GNN$_{\text{sum}}$, respectively.

4.3 Experiment settings

We implement NLA-GNN in PyTorch✠. All experiments are conducted on a single Linux server with Intel Xeon Silver 4214R CPU, 128 G RAM, and 4 NVIDIA GeForce RTX 3080 GPU. For a fair comparison, the embedding size is fixed to 128 for the performance comparison experiment. We optimize the model with Adam[28] and use the Xavier method[29] to initialize the model parameters. And the batch size is fixed to 4096 for all datasets. We follow the suggested settings in the authors' original papers and use a grid search to choose the optimum hyper-parameters for all methods: the learning rate is tuned in the range $[0.0001, 0.0005, 0.001, 0.005]$; the coefficient of L_2 normalization is tuned in the range $[0, 10^{-6}, 10^{-5}, 10^{-4}, \ldots, 0.1, 1]$; the dropout ratio is tuned in $[0, 0.1, \ldots, 1.0]$. We set the number of GCN layers to 3 for GRec, and the weighted matrix size of each GCN layer is also set to 128, which is used as suggested by the original GRec paper[13]. For MF, GRec, and NLA-GNN, we randomly select apps in each training batch, along with their corresponding positive and negative samples, and optimize the model parameters by BPR loss.

4.4 Evaluation protocols

We adopt widely-used evaluation metrics to evaluate the performance of top-N recommendation: Recall, Precision, F1-Score, and Mean Average Precision (MAP), computed by the all-ranking protocol[15, 17].

• **Recall@N:** Recall refers to the proportion of positives predicted to be correct in all samples. Given a mobile app A, the Recall@N can be defined as

$$\text{Recall@N}_A = \frac{|\mathcal{R}_{\text{N}}(A) \cap \mathcal{T}(A)|}{|\mathcal{T}(A)|} \tag{14}$$

where $\mathcal{R}_{\text{N}}(A)$ is the top-N recommendation list for app A and $\mathcal{T}(A)$ is the testing set of app A.

• **Precision@N:** Precision refers to the proportion of the number of correctly predicted samples in the total number of samples, which can be defined as

$$\text{Precision@N}_A = \frac{|\mathcal{R}_{\text{N}}(A) \cap \mathcal{T}(A)|}{|\mathcal{R}_{\text{N}}(A)|} \tag{15}$$

• **F1-Score@N:** F1-Score comprehensively considers the two metrics, which is the summed average of Precision and Recall,

$$\text{F1-Score@N}_A = 2 \times \frac{\text{Precision@N}_A \times \text{Recall@N}_A}{\text{Precision@N}_A + \text{Recall@N}_A} \tag{16}$$

• **MAP@N:** Average Precision (AP) is used to measure the rationality of the ranking position of the recommended results, and MAP is the mean value of AP,

$$\text{MAP@N}_A = \frac{1}{\sum_{i=1}^{N} \text{rel}(i)} \sum_{i=1}^{N} \frac{\sum_{j=1}^{i} \text{rel}(i)}{i} \times \text{rel}(i) \tag{17}$$

where $\text{rel}(i)$ is a binary indicator that indicates whether the library at position i of the recommendation list is in app A's testing set. If $\mathcal{R}_{\text{N}}(A)[i] \in \mathcal{T}(A)$, we set $\text{rel}(i) = 1$, and 0 otherwise.

We calculate four metrics independently for each mobile app in testing set, and then get the overall evaluation scores of the recommendation model by average operation.

4.5 Performance comparison

4.5.1 Overall comparison

Table 3 reports the performance of NLA-GNN$_{\text{cat}}$, NLA-GNN$_{\text{sum}}$, and other baselines on three datasets. And we can draw the following conclusions.

Our proposed NLA-GNN$_{\text{cat}}$ and NLA-GNN$_{\text{sum}}$ achieve significant performance improvements on all evaluation metrics for all datasets. Taking NLA-GNN$_{\text{sum}}$ as an example, it outperforms Popular, MF, LibRec, LibSeek, and GRec by 81.82%, 20.11%, 39.80%, 16.25%, and 4.47% with respect to Recall@5 on MALib#3 datasets, respectively. This shows the rationality and generalization of the proposed NLA-GNN. We attribute the performance improvements to the following: (1) NLA-GNN enables seamless information transfer between two types of node information on the app-library bipartite graph by stacking multiple alternating information distillation layers. This not only explicitly models the higher-order connectivity of the app-library, but also eliminates the negative impact of the heterogeneity of the bipartite graph. (2) By discarding the redundant design in graph convolution and initializing only one node embedding, NLA-GNN is easier to fit limited sparse samples, which is consistent with previous studies.

✠ https://pytorch.org/

Table 3 Overall performance comparison. The performance of the proposed model on each dataset is highlighted in bold, and the best score in baselines is underlined.

Dataset	Method	Top-5				Top-10			
		Precision	Recall	F1-Score	MAP	Precision	Recall	F1-Score	MAP
MALib#1	Popular	0.0753	0.3765	0.1255	0.2840	0.0457	0.4565	0.0831	0.2949
	MF	0.1326	0.6629	0.2210	0.5150	0.0754	0.7544	0.1371	0.5274
	LibRec	0.1267	0.6335	0.2112	0.4622	0.0668	0.6682	0.1215	0.4669
	LibSeek	0.1348	0.6741	0.2247	0.5236	0.0755	0.7553	0.1373	0.5346
	GRec	0.1521	0.7607	0.2536	0.6269	0.0828	0.8283	0.1506	0.6360
	NLA-GNN$_{cat}$	**0.1540**	**0.7699**	**0.2567**	**0.6307**	**0.0833**	**0.8329**	**0.1515**	**0.6394**
	NLA-GNN$_{sum}$	**0.1544**	**0.7721**	**0.2574**	**0.6322**	**0.0835**	**0.8345**	**0.1517**	**0.6409**
MALib#2	Popular	0.2147	0.3579	0.2684	0.5931	0.1341	0.4468	0.2063	0.5682
	MF	0.3497	0.5904	0.4387	0.7209	0.2095	0.7068	0.3229	0.6829
	LibRec	0.2789	0.4648	0.3486	0.6883	0.1542	0.5142	0.2373	0.6864
	LibSeek	0.3710	0.6183	0.4637	0.7280	0.2158	0.7193	0.3320	0.6971
	GRec	0.4099	0.6915	0.5142	0.7977	0.2337	0.7879	0.3602	0.7605
	NLA-GNN$_{cat}$	**0.4163**	**0.7026**	**0.5223**	**0.8051**	**0.2347**	**0.7922**	**0.3617**	**0.7710**
	NLA-GNN$_{sum}$	**0.4176**	**0.7046**	**0.5237**	**0.8080**	**0.2350**	**0.7930**	**0.3621**	**0.7742**
MALib#3	Popular	0.3383	0.3383	0.3383	0.7413	0.2180	0.4360	0.2907	0.6813
	MF	0.5052	0.5121	0.5083	0.7838	0.3238	0.6555	0.4333	0.7253
	LibRec	0.4400	0.4400	0.4400	0.6922	0.2434	0.4868	0.3245	0.6890
	LibSeek	0.5291	0.5291	0.5291	0.7896	0.3293	0.6587	0.4391	0.7396
	GRec	0.5868	0.5945	0.5902	0.8397	0.3613	0.7312	0.4834	0.7856
	NLA-GNN$_{cat}$	**0.6062**	**0.6140**	**0.6097**	**0.8540**	**0.3680**	**0.7445**	**0.4923**	**0.8034**
	NLA-GNN$_{sum}$	**0.6073**	**0.6151**	**0.6108**	**0.8567**	**0.3683**	**0.7452**	**0.4927**	**0.8054**

Comparing all baseline models horizontally, Popular only mechanically recommends the most popular TPLs, which undoubtedly goes against the original intention of personalized recommender system. The performance of MF, LibRec, and LibSeek are at the same level. Although LibRec combines association rule mining and LibSeek additionally considers popularity bias and long-tail effect, they fail to consider the high-order relationship between app and library, resulting in sub-optimal performance. In addition, the performance improvement of NLA-GNN$_{cat}$ and NLA-GNN$_{sum}$ in MALib#3 is greater than that in MALib#2, while the performance improvement of MALib#2 is higher than that of MALib#1. This shows that NLA-GNN has a strong anti-sparsity ability when the training set is sparser.

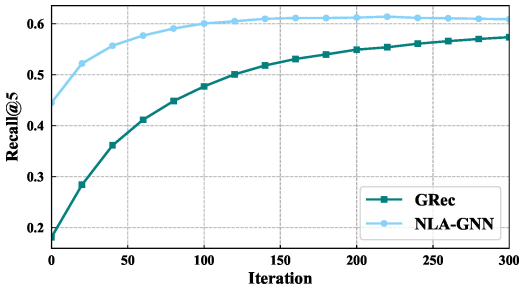
An interesting phenomenon is that, in most cases, the performance of NLA-GNN$_{sum}$ is better than that of NLA-GNN$_{cat}$. One possible reason is that the concatenation operation will significantly increase the dimension size of the final representation of the mobile app and library nodes. The inner product operation with too large a dimension may restrict the correct prediction of the model, while the sum operation integrates the embedding information of different semantic layers in a consistent dimension, which is helpful for the learning of downstream recommendation tasks.

4.5.2 In-depth comparison with GRec

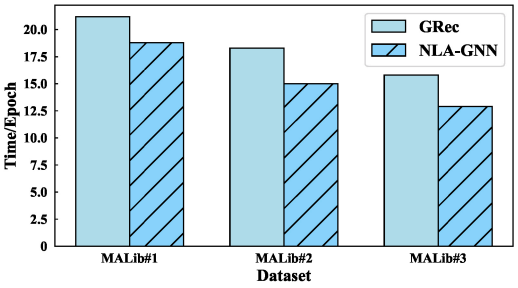
In this section, we perform detailed comparisons of NLA-GNN with GRec, the current state-of-the-art graph-based TPL recommendation model containing three parts: convergence speed, training speed, and model size.

(1) Convergence speed: Taking MALib#3 dataset as an example, we provide the training curves (Fig. 5a) of NLA-GNN and GRec, in which the ordinate represents the Recall@5 of the corresponding model on the testing set. Intuitively, the efficiency of model training can be effectively improved by abandoning the redundant design in GCN and using only one type of embedding for representation learning.

(2) Training speed: Figure 5b provides a comparison of the training speed of GRec and NLA-GNN on the three datasets under the same experimental environment and parameter settings. Not surprisingly, since NLA-GNN utilizes only the most basic light graph convolution paradigm to propagate information, the training speed of NLA-GNN has an advantage over GRec for the same experimental setup, which is especially important for recommender systems with high



(a) Training curve



(b) Training speed

Fig. 5 Detailed comparison between GRec and NLA-GNN, (a) training efficiency and (b) training speed per epoch.

requirements for real-time performance.

(3) Model size: Another concern is the model size, i.e., the number of parameters that can be trained in the model. Figure 6 shows the visualization results of comparing the model size of GRec with that of NLA-GNN when embedding size $d = 128$. Consistent with the analysis in Section 3.7.1, NLA-GNN initializes only one type of node (e.g., library) and does not introduce additional feature transformation matrices in each graph convolution layer. Thus the number of parameters of NLA-GNN is far less than that of GRec (44.28 times smaller). Furthermore, the analysis of model size can also support that of convergence speed (Fig. 5a) and training speed (Fig. 5b). That is, an excessive number of model parameters may not bring additional benefit to the model, but rather affect the training and convergence speed of the model.

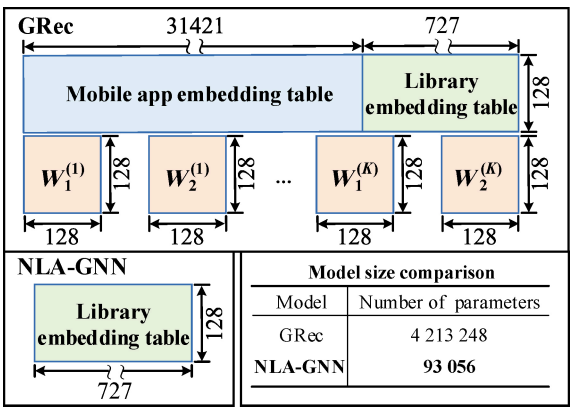


Fig. 6 Comparison of model parameters between GRec and NLA-GNN when embedding size $d = 128$.

4.6 Parameter analysis

As previously analyzed, NLA-GNN is an extremely simplified model. The hyper-parameters that can be modified are the number of alternating information distillation layers K, embedding size d, and regularization coefficient λ. In this part, we focus on these parameters to analyze their impact on the performance of NLA-GNN.

4.6.1 Impact of layer numbers

A key factor in the remarkable success of GCN is the capability to encode the high-order connectivity of nodes on the graph explicitly. In this part, we study the effect of different numbers of alternate information distillation layers on the performance of NLA-GNN. Specifically, we fix other parameters and adjust the number of layers K in the range of 1 to 5. The performance of NLA-GNN$_{\text{cat}}$ and NLA-GNN$_{\text{sum}}$ on the three datasets with different K is provided in Table 4. The results reveal the following observations.

The trends for NLA-GNN$_{\text{cat}}$ and NLA-GNN$_{\text{sum}}$ are generally consistent across the three datasets, with the performance starting to increase as the number of layers rises and is particularly pronounced at low layers. This finding suggests that NLA-GNN can benefit from multi-layer graph convolution operation. Through K-layer information propagation, each mobile app and library node can obtain node information beyond K-hops. Meanwhile, its node information is also passed beyond K-hops, which helps break the information blockage and allows the model to learn collaborative signals that contribute to downstream recommendation tasks.

By contrast, the performance improvement starts to slow down and even degrades in some cases as the number of layers increases further. The possible reasons are presented as follows. (1) Excessively deep model structures may allow nodes to over-propagate and aggregate high-order information, which may introduce noisy information that can interfere with the normal training of the model. (2) Stacking too many graph convolution layers may over-smooth the node embedding and result in limited differentiation of node representation, i.e., the over-smoothing problem that has received wide attention in graph structure representation learning[14, 30, 31].

Table 4 Performance comparison with respect to layers. The best performance of the model on each dataset is highlighted in bold. "Prec" is an abbreviation for Precision to save space.

Dataset	Layer	NLA-GNN$_{cat}$				NLA-GNN$_{sum}$			
		Prec@5	Recall@5	F1-Score@5	MAP@5	Prec@5	Recall@5	F1-Score@5	MAP@5
MALib#1	1 Layer	0.1527	0.7636	0.2545	0.6256	0.1527	0.7636	0.2545	0.6256
	2 Layers	0.1531	0.7658	0.2552	0.627	0.1534	0.7672	0.2557	0.6273
	3 Layers	0.1532	0.7663	0.2554	0.6281	0.1541	0.7705	0.2568	0.6303
	4 Layers	**0.1540**	**0.7699**	**0.2567**	**0.6307**	**0.1544**	**0.7721**	**0.2574**	**0.6322**
	5 Layers	0.1532	0.7680	0.2553	0.6291	0.1540	0.7699	0.2566	0.6301
MALib#2	1 Layer	0.4134	0.6969	0.5186	0.7902	0.4134	0.6969	0.5186	0.7902
	2 Layers	0.4139	0.6983	0.5192	0.7953	0.4152	0.7011	0.5208	0.8006
	3 Layers	**0.4164**	**0.7026**	**0.5224**	**0.8051**	0.4171	0.7036	0.5232	0.8061
	4 Layers	0.4160	0.7018	0.5219	0.8013	**0.4176**	**0.7046**	0.5237	**0.8080**
	5 Layers	0.4156	0.7017	0.5214	0.7993	**0.4176**	0.7044	**0.5238**	0.8053
MALib#3	1 Layer	0.6013	0.6091	0.6048	0.8476	0.6013	0.6091	0.6048	0.8476
	2 Layers	0.6024	0.6103	0.6059	0.8499	0.6047	0.6125	0.6081	0.8494
	3 Layers	**0.6062**	**0.6140**	**0.6097**	**0.8540**	0.6073	0.6151	0.6108	0.8567
	4 Layers	0.6057	0.6135	0.6091	0.8530	0.6074	0.6153	0.6109	0.8567
	5 Layers	0.6054	0.6132	0.6089	0.8503	**0.6089**	**0.6167**	**0.6124**	**0.8587**

In addition, it is worth noting that when $K = 1$, NLA-GNN is equivalent to completing an app embedding construction process and then obtaining the prediction score through the inner product. Compared with MF and LibSeek, NLA-GNN-1 has made significant improvements on all datasets, which shows the rationality of initializing only one type of node.

4.6.2 Impact of embedding dimension

To investigate the effect of embedding size on the performance of NLA-GNN, we fix other irrelevant parameters and set the number of alternating information distillation layers to 3. Subsequently, we change the embedding size $d = [32, 64, 128, 256, 512]$ into NLA-GNN$_{cat}$ and NLA-GNN$_{sum}$. Figure 7 plots the effect of embedding size d against Recall@5, Precision@5, F1-Score@5, and MAP@5 on three datasets, and the following findings are presented.

The performance of NLA-GNN$_{cat}$ and NLA-GNN$_{sum}$ improves as the dimensions become increasingly large, implying that large embedding dimensions aid representation learning. This finding is easy to comprehend because each dimension of the node's representation reflects a node feature; thus, additional dimensions allow the recommender system to learn more important information regarding the node and its neighborhoods.

Large embeddings are beneficial for representation learning, but they can markedly increase the time overhead of model training. Furthermore, when the embedding dimension is too large, the performance gain is restricted, implying that massive scale parameters might interfere with the normal training of the model and should thus not be encouraged.

4.6.3 Impact of regularization coefficient

The last parameter worthy of attention is the L_2 regularization coefficient λ, which, as a part of the loss function, plays a vital role in alleviating the over-fitting of the model to the training set. In this part, we first fix other parameters and set the number of layers to 3; we then set λ in the range of $[0, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}]$. Figure 8 plots the performance of NLA-GNN$_{cat}$ and NLA-GNN$_{sum}$ corresponding to different values of λ in three datasets and obtains the following findings.

The performance of NLA-GNN$_{cat}$ and NLA-GNN$_{sum}$ shows consistency across all datasets. The performance of the model first improves as the λ increases, indicating that the regularization of appropriate strength helps model training. However, the performance of the model starts to decrease sharply after the λ increases further, indicating that an overly large regularization term can seriously interfere with model parameter updates.

In addition, it is worth noting that even if λ is set to 0, NLA-GNN can still train effectively and achieve satisfactory performance. This finding indicates that NLA-GNN is not easily affected by over-fitting. A key reason is that NLA-GNN only initializes one type of node, facilitating its easy training and convergence, and does not need to use additional anti-over-fitting mechanisms such as dropout, so as to ensure model simplification.

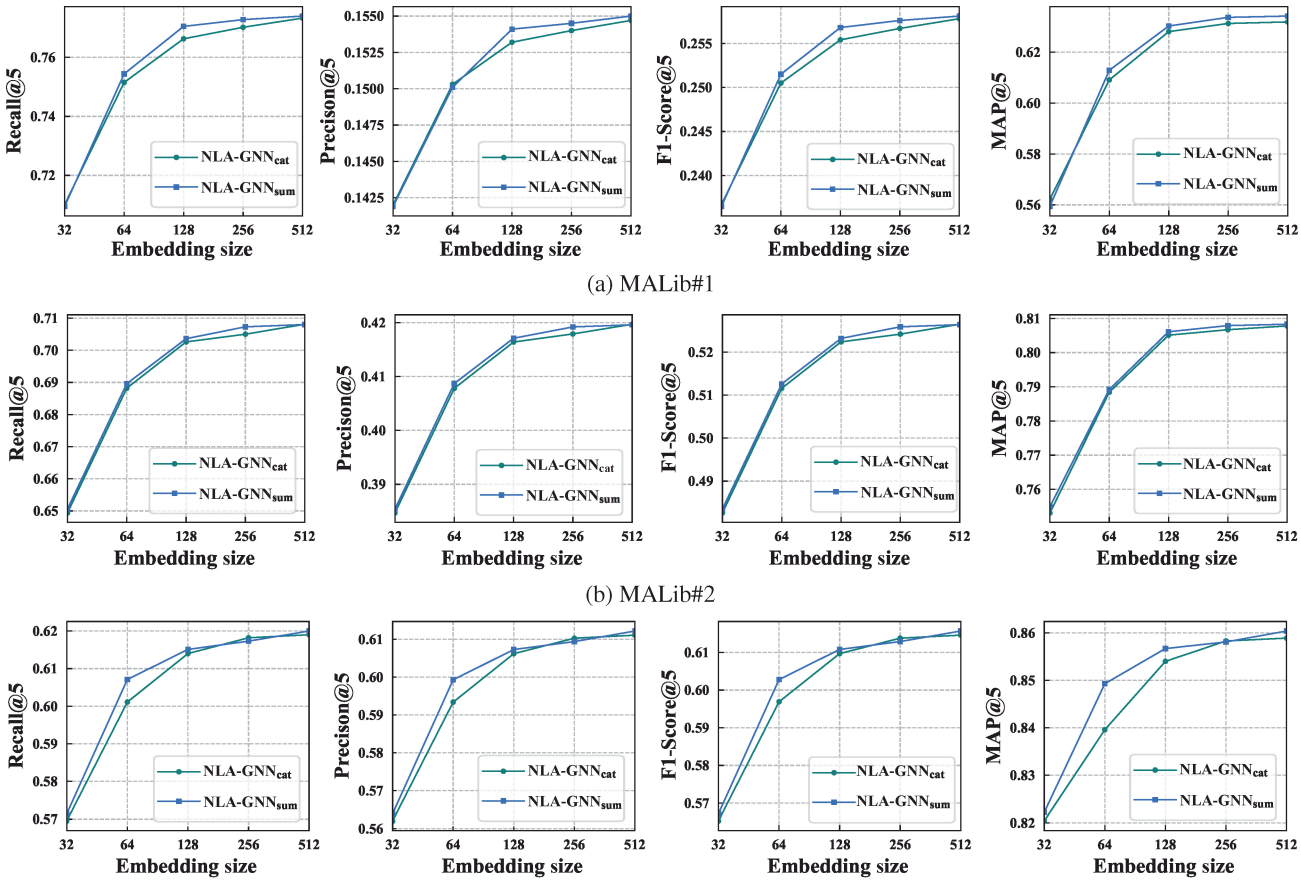


Fig. 7 Recall@5, Precision@5, F1-Score@5, and MAP@5 curves of NLA-GNN$_{cat}$ and NLA-GNN$_{sum}$ with respect to different embedding sizes on three datasets (results on the Top-10 task show the same trend and are omitted herein for space).

4.7 Threats to validity

4.7.1 Internal validity

A main threat to the internal validity lies in the possible correlation between the performance improvement of NLA-GNN and the negative impact of graph neural networks, and the heterogeneity of bipartite graphs on recommender systems. To minimize this threat, we first perform a detailed comparison with GRec considering training speed, convergence speed, and model size. This experiment can effectively verify that NLA-GNN has rapid training and a small cost, which is consistent with our hypothesis in the Introduction section. In addition, we analyze the effect of the number of layers on the performance of NLA-GNN in the parametric analysis. Such an analysis allows us to observe whether NLA-GNN can benefit from multi-layer graph convolution operations. The experimental results show that NLA-GNN can achieve effective performance as the number of layers increases. This finding indicates that NLA-GNN can adapt well to bipartite graph structures and learn the necessary knowledge from them.

4.7.2 External validity

A main threat to the external validity is the MALib dataset used in our experiments, which is a real dataset from Google Play with 31 432 real applications corresponding to TPLs, and which is open source. In order to minimize the threat, we adopt this dataset and use it to simulate different scenarios in the real world. Specifically, we remove different proportions of TPLs as the ground-truth and recommend different numbers of TPLs in the testing phase, which can simulate real-world scenarios with different sparsity and sizes of user requirements.

4.7.3 Construct validity

The main threat comes from the capability of the chosen baseline to demonstrate the effectiveness of NLA-GNN. To minimize the threat, different types of models are selected for comparison. Specifically, Popular recommends only the most prevalent TPLs. MF and LibSeek consider only interactions in the mobile app-TPL co-occurrence matrix. LibRec is a classical library recommendation algorithm, and GRec

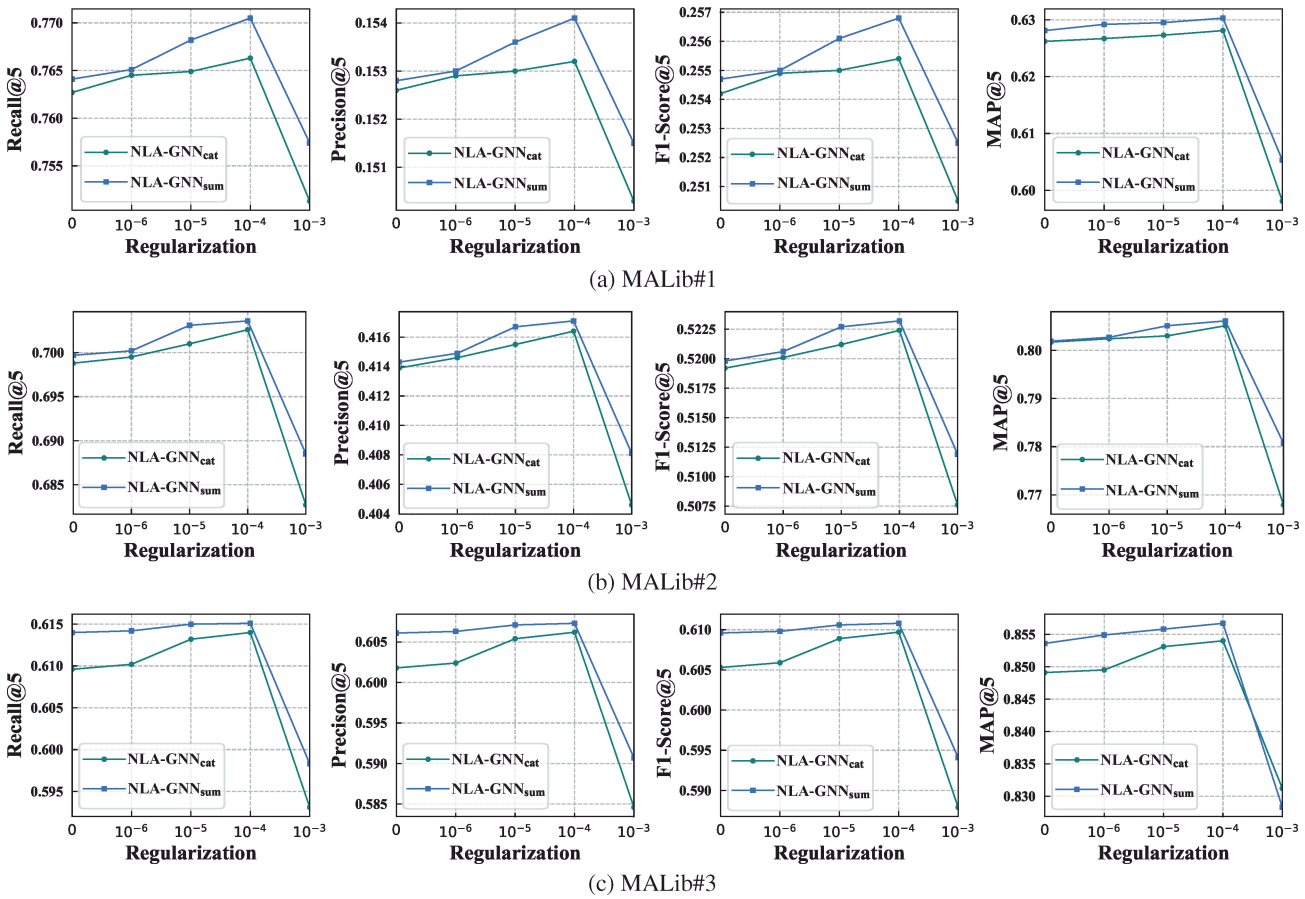


Fig. 8 Recall@5, Precision@5, F1-Score@5, and MAP@5 curves of NLA-GNN$_{cat}$ and NLA-GNN$_{sum}$ with respect to different regularization coefficient on three datasets (results on the Top-10 task show the same trend and are omitted herein for space).

additionally considers multiple layers of interactions. By comparing with these different types of approaches, it can be effectively verified that NLA-GNN has additional advantages while retaining the advantages of these models. For example, one-layer NLA-GNN (Section 4.6.1) demonstrates strong performance gains over MF and LibSeek, which demonstrates the effectiveness of a single type of representation learning.

4.7.4 Conclusion validity

The main threat is whether our conclusions are statistically significant and credible. We take three steps to minimize the threat. First, we use a consistent experimental setup and dataset, which is mainly for fairness. We also run each experiment 50 times and report mean results to ensure the credibility and non-coincidence of the results. Second, we conduct comparative experiments using the open-source code of the baseline model and maintain the consistency of conclusions with previous works. Third, we provide a detailed description of the proposed model, the parameter description, and the parameter analysis, which will help the subsequent research and replication work.

5 Related Work

With the fast expansion of the Internet and electronic devices in recent years, developers have begun to provide increased attention to mobile application development. Researchers have also explored how developers choose appropriate apps from an expanding number of off-the-shelf TPLs to decrease repetitious effort and accomplish agile development and speedy delivery of customer needs. In such a context, the concepts of TPL recommendation[8] and API recommendation[32] are proposed and have been investigated in two conceptual directions.

The first type of work used traditional machine learning paradigms, such as clustering[33, 34], similarity measures[35, 36], or feature matching[37, 38], to find similar modules. For example, LibPecker[38] designs an adaptive class matching module to measure the similarity of different TPLs. Another research direction is realized with the help of recommendation techniques. CF[9, 23, 39],

the core idea of modern recommender systems, assumes that users (items) with behavioral (feature) similarities will show consistent preferences for items (users). In the TPL recommendation scenario, users and items represent mobile applications and TPLs, respectively. Inspired by traditional recommender systems, many excellent works have been produced in the field of TPL recommendation. For example, LibRec[8] is a pilot study that combines association rule mining and CF to find TPLs that the target project may use in the future. However, this approach can only recommend highly popular libraries. Unlike LibRec, CrossRec[40] additionally considers the dependencies of the project under development, thus providing highly relevant recommendation results to the project. Subsequently, LibSeek[5] is a novel TPL recommendation model that uses the matrix factorization technique[11] that has made a big splash in recommender systems to decompose the app-library interaction matrix into two single hidden vectors that can be used to compute the similarity scores between the app and the library through the inner product operation. In addition, this model considers the popularity bias[10] and the long-tail effect in recommender systems. By additionally adding an adaptive weighting mechanism to distinguish the importance of different TPLs, LibSeek achieves a better performance than traditional MF. Although both types of approaches are simple and effective, they have some inherent limitations. Specifically, content-based models often require additional side information, such as app development-related logs, library functionality, and interface definitions[7], and such auxiliary information is also not easily accessible. The CF-based approaches, on the other hand, only consider the correlation between the app and its historical interactions, while ignoring the complex relationships between app and library, app to app, and library to library over long distances[15].

In the last few years, GCNs[14, 20, 30, 31] have started to emerge and have had remarkable success in semi-supervised classification tasks. Inspired by this, GRec[13] defines TPL recommendations for the first time from a graph perspective. Specifically, GRec treats the app and library as nodes and their interaction record as an edge on the graph. GRec basically covers all techniques of GCNs, including self-connection, feature transformation, and nonlinear activation[14, 15, 24]. By stacking multiple graph convolutional layers, GRec can easily learn the high-order connectivity between the app and library, which is impossible with the traditional TPL recommendation model. However, as described in the previous sections, GRec still has limitations and there is still a considerable room for improvement. The proposed NLA-GNN further simplifies the framework of GRec by removing redundant modules and steps and initializing only one type of node embedding to perform the graph convolution process. The experimental results show that NLA-GNN achieves the best performance on all datasets and has significant advantages considering training speed, convergence speed, and model size.

6 Conclusion and Future Work

In this paper, we proposed a novel NLA-GNN for TPL recommendation task. NLA-GNN further simplifies the graph convolution process based on GRec, i.e., it removes the redundant processes of self-connection, feature transformation, and nonlinear activation, and alternatively completes the construction and update of all types of node representations on the app-library graph by only one type of node embedding. This condition enables NLA-GNN to tap into the complex high-order relationships between the app and library while keeping the model compact and eliminating the impact of heterogeneity of bipartite graphs. We have conducted extensive comparison experiments and parameter analyses on a real large-scale dataset MALib. The experimental results show that the proposed NLA-GNN can more accurately recommend the preferred TPLs for developers than the state-of-the-art baseline models, and has significant advantages in terms of training speed, convergence speed, and model size. In the future, we will further investigate the complex high-order relationship between the app and library, and expect to distinguish the degree of contribution of different neighbor nodes to the central node by introducing an attention mechanism and graph attention network. In addition, a feasible direction is to integrate the feature information of the mobile app and TPL (such as geographic location, publisher, and type label, etc.) to further enhance the performance and empower the interpretability of the recommendation results.

Acknowledgment

This work was supported by the Key Project of Nature Science Research for Universities of Anhui Province of China (No. KJ2020A0657), the National Natural Science Foundation of China (Nos. 62272001, 61872002, and 62276146), and the University Collaborative Innovation Project of Anhui Province (No. GXXT-2021-087).

References

[1] Y. Zhang, B. Hu, and Y. Zhang, Model-driven open ecological cloud enterprise resource planning, *Int. J. Web Serv. Res.*, vol. 18, no. 3, pp. 82–99, 2021.

[2] P. Salza, F. Palomba, D. Di Nucci, A. De Lucia, and F. Ferrucci, Third-party libraries in mobile apps, *Empir. Softw. Eng.*, vol. 25, no. 3, pp. 2341–2377, 2020.

[3] M. A. Saied and H. Sahraoui, A cooperative approach for combining client-based and library-based API usage pattern mining, in *Proc. 24th Int. Conf. Program Comprehension* (*ICPC*), Austin, TX, USA, 2016, pp. 1–10.

[4] E. Derr, S. Bugiel, S. Fahl, Y. Acar, and M. Backes, Keep me updated: An empirical study of third-party library updatability on android, in *Proc. 2017 ACM SIGSAC Conf. Computer and Communications Security*, Dallas, TX, USA, 2017, pp. 2187–2200.

[5] Q. He, B. Li, F. Chen, J. Grundy, X. Xia, and Y. Yang, Diversified third-party library prediction for mobile app development, *IEEE Trans. Softw. Eng.*, vol. 48, no. 1, pp. 150–165, 2022.

[6] H. Henriques, H. Lourenço, V. Amaral, and M. Goulão, Improving the developer experience with a low-code process modelling language, in *Proc. 21st ACM/IEEE Int. Conf. Model Driven Engineering Languages and Systems*, Copenhagen, Denmark, 2018, pp. 200–210.

[7] T. Ki, C. M. Park, K. Dantu, S. Y. Ko, and L. Ziarek, Mimic: UI compatibility testing system for android apps, in *Proc. 41st Int. Conf. Software Engineering*, Montreal, Canada, 2019, pp. 246–256.

[8] F. Thung, D. Lo, and J. Lawall, Automated library recommendation, in *Proc. of the 20th Working Conf. Reverse Engineering* (*WCRE*), Koblenz, Germany, 2013, pp. 182–191.

[9] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. S. Chua, Neural collaborative filtering, in *Proc. 26th Int. Conf. on World Wide Web*, Perth, Australia, 2017, pp. 173–182.

[10] H. Abdollahpouri, M. Mansoury, R. Burke, and B. Mobasher, The connection between popularity bias, calibration, and fairness in recommendation, in *Proc. 14th ACM Conf. Recommender Systems*, Virtual Event, Brazil, 2020, pp. 726–731.

[11] Y. Koren, R. Bell, and C. Volinsky, Matrix factorization techniques for recommender systems, *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[12] Y. Zhang, K. Wang, Q. He, F. Chen, S. Deng, Z. Zheng, and Y. Yang, Covering-based web service quality prediction via neighborhood-aware matrix factorization, *IEEE Trans. Serv. Comput.*, vol. 14, no. 5, pp. 1333–1344, 2021.

[13] B. Li, Q. He, F. Chen, X. Xia, L. Li, J. Grundy, and Y. Yang, Embedding app-library graph for neural third party library recommendation, in *Proc. 29th ACM Joint Meeting on European Software Engineering Conf. and Symp. Foundations of Software Engineering*, Athens, Greece, 2021, pp. 466–477.

[14] T. N. Kipf and M. Welling, Semi-supervised classification with graph convolutional networks, presented at the Int. Conf. Learning Representations, Toulon, France, 2017.

[15] X. Wang, X. He, M. Wang, F. Feng, and T. S. Chua, Neural graph collaborative filtering, in *Proc. 42nd Int. ACM SIGIR Conf. Research and Development in Information Retrieval*, Paris, France, 2019, pp. 165–174.

[16] L. Chen, L. Wu, R. Hong, K. Zhang, and M. Wang, Revisiting graph based collaborative filtering: A linear residual graph convolutional network approach, *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 1. pp. 27–34, 2020.

[17] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, LightGCN: Simplifying and powering graph convolution network for recommendation, in *Proc. 43rd Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, Virtual Event, China, 2020, pp. 639–648.

[18] J. Cao, X. Lin, S. Guo, L. Liu, T. Liu, and B. Wang, Bipartite graph embedding via mutual information maximization, in *Proc. 14th ACM Int. Conf. Web Search and Data Mining*, Virtual Event, Israel, 2021, pp. 635–643.

[19] J. Sun, Y. Zhang, W. Guo, H. Guo, R. Tang, X. He, C. Ma, and M. Coates, Neighbor interaction aware graph convolution networks for recommendation, in *Proc. 43rd Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, Virtual Event, China, 2020, pp. 1289–1298.

[20] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, Simplifying graph convolutional networks, in *Proc. 36th Int. Conf. on Machine Learning*, Long Beach, CA, USA, 2019, pp. 6861–6871.

[21] S. Kabbur, X. Ning, and G. Karypis, FISM: Factored item similarity models for top-N recommender systems, in *Proc. 19th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, Chicago, IL, USA, 2013, pp. 659–667.

[22] S. Rendle, Factorization machines with libFM, *ACM Trans. Intell. Syst. Technol.*, vol. 3, no. 3, p. 57, 2012.

[23] W. Chen, F. Cai, H. Chen, and M. De Rijke, Joint neural collaborative filtering for recommender systems, *ACM Trans. Inform. Syst.*, vol. 37, no. 4, p. 39, 2019.

[24] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, Graph convolutional neural networks for web-scale recommender systems, in *Proc. 24th ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining*, London, UK, 2018, pp. 974–983.

[25] D. Zhang, J. Yin, X. Zhu, and C. Zhang, Network representation learning: A survey, *IEEE Trans. Big Data*, vol. 6, no. 1, pp. 3–28, 2020.

[26] C. K. Hsieh, L. Yang, Y. Cui, T. Y. Lin, S. Belongie, and D. Estrin, Collaborative metric learning, in *Proc. 26th Int. Conf. World Wide Web*, Perth, Australia, 2017, pp. 193–201.

[27] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, BPR: Bayesian personalized ranking from implicit feedback, in *Proc. 25th Conf. Uncertainty in Artificial Intelligence*, Montreal, Canada, 2009, pp. 452–461.

[28] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv: 1412.6980, 2017.

[29] X. Glorot and Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in *Proc. 13th Int. Conf. Artificial Intelligence and Statistics*, Sardinia, Italy, 2010, pp. 249–256.

[30] M. Liu, H. Gao, and S. Ji, Towards deeper graph

neural networks, in *Proc. 26th ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining*, Virtual Event, CA, USA, 2020, pp. 338–348.

[31] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, Simple and deep graph convolutional networks, in *Proc. 37th Int. Conf. Machine Learning*, Vivtual Event, Austria, 2020, pp. 1725–1735.

[32] W. Zheng, Q. Zhang, and M. Lyu, Cross-library API recommendation using web search engines, in *Proc. 19th ACM SIGSOFT Symp. and the 13th European Conf. Foundations of Software Engineering*, Szeged, Hungary, 2011, pp. 480–483.

[33] A. Narayanan, L. Chen, and C. K. Chan, AdDetect: Automated detection of android ad libraries using semantic analysis, in *Proc. 9th Int. Conf. on Intelligent Sensors, Sensor Networks and Information Processing* (*ISSNIP*), Singapore, 2014, pp. 1–6.

[34] B. Liu, B. Liu, H. Jin, and R. Govindan, Efficient privilege de-escalation for ad libraries in mobile apps, in *Proc. 13th Annu. Int. Conf. Mobile Systems, Applications, and Services*, Florence, Italy, 2015, pp. 89–103.

[35] F. Thung, S. Wang, D. Lo, and J. Lawall, Automatic recommendation of API methods from feature requests, in *Proc 28th IEEE/ACM Int. Conf. Automated Software Engineering* (*ASE*), Silicon Valley, CA, USA, 2013, pp. 290–300.

[36] M. A. Saied, A. Ouni, H. Sahraoui, R. G. Kula, K. Inoue, and D. Lo, Improving reusability of software libraries through usage pattern mining, *J. Syst. Softw.*, vol. 145, pp. 164–179, 2018.

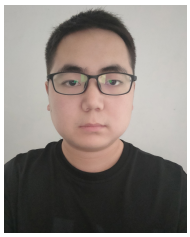
[37] M. Backes, S. Bugiel, and E. Derr, Reliable third-party library detection in android and its security applications, in *Proc. 2016 ACM SIGSAC Conf. Computer and Communications Security*, Vienna, Austria, 2016, pp. 356–367.

[38] Y. Zhang, J. Dai, X. Zhang, S. Huang, Z. Yang, M. Yang, and H. Chen, Detecting third-party libraries in android applications with high precision and recall, in *Proc. 25th Int. Conf. Software Analysis, Evolution and Reengineering* (*SANER*), Campobasso, Italy, 2018, pp. 141–152.

[39] L. Wu, X. He, X. Wang, K. Zhang, and M. Wang, A survey on accuracy-oriented neural recommendation: From collaborative filtering to information-rich recommendation, *IEEE Trans. Knowl. Data Eng.*, doi: 10.1109/TKDE.2022.3145690.

[40] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, and M. Di Penta, CrossRec: Supporting software developers by recommending third-party libraries, *J. Syst. Softw.*, vol. 161, p. 110460, 2020.

Ying Jin received the MEng degree from Hefei University of Technology, China in 2009. She is an associate professor at the School of Artificial Intelligence and Big Data, Hefei University. Her research interests include service computing, data mining, and web service.



Yi Zhang received the BEng degree in computer science and technology from Anhui University, China in 2020, where he is currently a master student at the School of Computer Science and Technology, Anhui University. His current research interests include graph learning, recommender system, and service computing.

Yiwen Zhang received the PhD degree in management science and engineering from Hefei University of Technology, China in 2013. He is currently a professor at the School of Computer Science and Technology, Anhui University. His research interests include service computing, cloud computing, and big data analysis. Please see more information on http://bigdata.ahu.edu.cn/.