Information Processing and Management 61 (2024) 103831

Contents lists available at ScienceDirect

Information Processing and Management

journal homepage: www.elsevier.com/locate/ipm

Homogeneous graph neural networks for third-party library recommendation

Duantengchuan Li [a], Yuxuan Gao [a], Zhihao Wang [a,*], Hua Qiu [a], Pan Liu [b,*], Zhuoran Xiong [c], Zilong Zhang [a]

[a] School of Computer Science, Wuhan University, Wuhan 430072, China
[b] National Engineering Research Center for E-Learning, Central China Normal University, Wuhan 430079, China
[c] Electrical and Computer Engineering, McGill University, Montreal, Canada

ARTICLE INFO

Keywords:
Recommender system
Graph neural network
Mobile application
Third-party library

ABSTRACT

During mobile application development, developers often use various third-party libraries to expedite the development process and enhance application functionality. Real datasets often show significant long-tailed distribution characteristics, where a few third-party libraries are widely adopted, while most are seldom used, leading to extreme data sparsity. This distribution phenomenon challenges recommendation algorithms, which typically recommend widely used third-party libraries for basic functionality, failing to meet developers' specific feature needs. To address these limitations, we propose HGNRec, a third-party library recommendation model based on a homogeneous graph neural network. First, to overcome the limitations of fusing heterogeneous node information, we decompose the heterogeneous graph network into two homogeneous graph networks using a statistical method. Second, the two constructed GNN models use separate aggregation and nonlinear transformation network structures for adaptive aggregation, along with edge-level and feature-level constraint methods to optimize model performance. In homogeneous graph networks, low-order and high-order neighbor information of nodes are propagated and aggregated in the same knowledge space, capturing the complex interactions among homogeneous nodes. Furthermore, we validate the superiority of HGNRec compared to several state-of-the-art methods using real datasets. Source code will be available at https://github.com/dacilab/HGNRec.

1. Introduction

In recent years, the number of mobile applications of various types and scales has grown explosively. According to data from APPBrain website,[1] as of 2023, the number of Apps has surpassed 2.59 million. The increase in the number of Apps has led to a corresponding rise in the number of Third-Party Libraries (TPL). Third-party libraries provide various functionalities, including location services, network requests, and push advertisements (Li et al., 2017). Developers can seamlessly integrate these libraries into their code without creating them from scratch. Reusing third-party libraries to enhance development efficiency and quality has become increasingly popular among developers. However, as the number of third-party libraries grows, developers find it

* Corresponding authors.
E-mail addresses: dtclee1222@whu.edu.cn (D. Li), zhihao_wang@whu.edu.cn (Z. Wang), panliu0912@gmail.com (P. Liu), zhuoran.xiong@mail.mcgill.ca (Z. Xiong), zilongzhang1022@whu.edu.cn (Z. Zhang).

[1] https://www.appbrain.com/stats/number-of-android-apps.

https://doi.org/10.1016/j.ipm.2024.103831
Received 18 March 2024; Received in revised form 9 June 2024; Accepted 7 July 2024

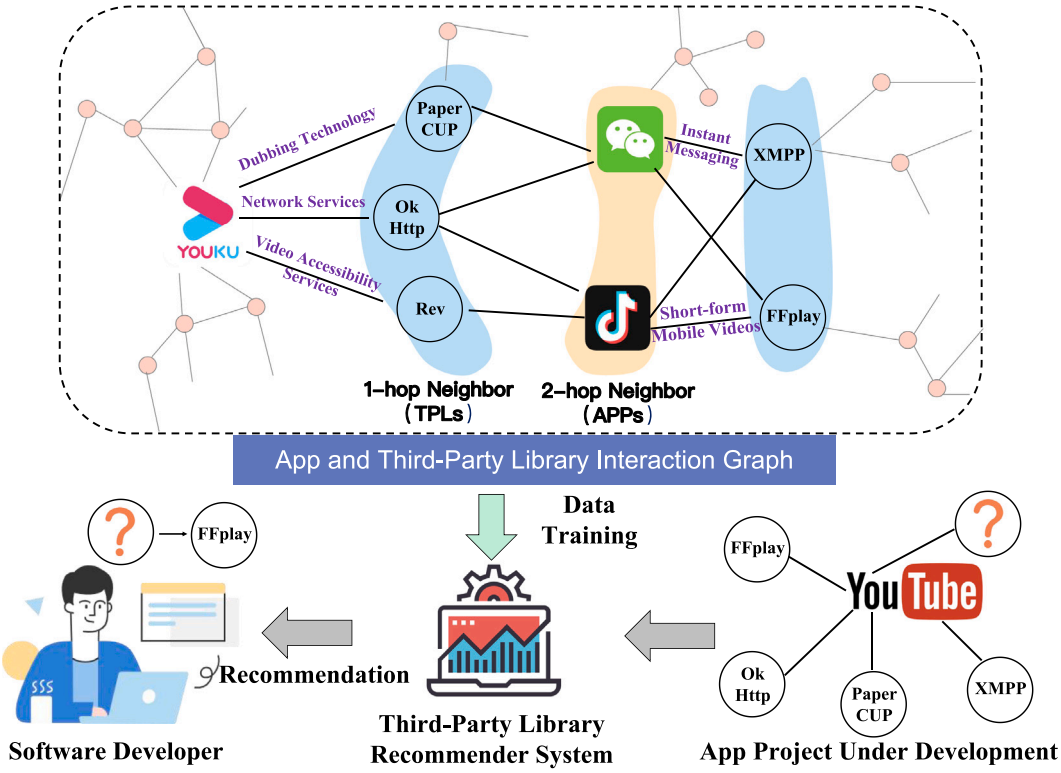

Fig. 1. Recommendation for software developer during App development process.

increasingly challenging to identify suitable ones to meet their requirements (Zhan et al., 2022). Therefore, it is crucial to provide developers with reliable and efficient algorithms for recommending third-party libraries.

Collaborative filtering (Liu, Zheng, Li, Shen et al., 2022) is an effective recommendation algorithm (Liang, Zhang, Wang, & Lu, 2024; Liu, Li, Wang, Wang et al., 2024) that uses historical behavior data to learn the similarity between users (Liu, Zheng, Li, Zhang et al., 2022) or items for recommendation. Inspired by this, LibRec (He & McAuley, 2016) combines association rule mining with collaborative filtering methods to recommend third-party libraries. Subsequently, CrossRec (Nguyen, Di Rocco, Di Ruscio, & Di Penta, 2020) further incorporates dependency sets from the development process to construct third-party library expressions and complete the recommendation. To address the singularity of recommended third-party libraries, LibSeek (Hidasi, Karatzoglou, Baltrunas, & Tikk, 2015) uses matrix factorization and adopts personalized weight mechanisms and neighborhood information.

With the development of deep learning (Zhao et al., 2024) and graph representation learning technologies (Li, Shi et al., 2024; Li, Xia et al., 2024; Li et al., 2023; Wang et al., 2023; Wu, Li, Lin & Zhang, 2021), considering that the recommendation process is actually a kind of graph topology, some researchers have used graph neural networks for recommendation algorithms. NGCF (Quadrana, Karatzoglou, Hidasi, & Cremonesi, 2017) was the first to combine graph neural networks with collaborative filtering to improve traditional collaborative filtering methods. Inspired by NGCF, GRec (Li et al., 2021) models mobile App third-party libraries and their interactions as a heterogeneous graph, using graph neural networks to mine high-order interaction information for recommending third-party libraries to developers. Graph-based recommendation methods can capture nonlinear relationships in user-item behavior data and explicitly encode collaborative information as high-order connections. This fills the gap in traditional collaborative filtering methods, where embedding functions lack explicit encoding of critical collaborative information.

However, in the context of recommending third-party libraries to mobile app developers, current graph neural network-based algorithms overlook an essential issue: aggregating feature vectors of apps and third-party libraries. App nodes and third-party library nodes are heterogeneous and belong to different knowledge bases. Previous work on graph networks involves heterogeneous graph information aggregation. First, first-order adjacency information, and more generally, odd-order adjacency information, often comes from interactions between Apps and third-party libraries (heterogeneous nodes), involving the integration of different node feature information. This means the aggregated information may originate from two different knowledge spaces. As shown in Fig. 1, using the video playback app YOUKU as an example, its first-order neighbors are third-party libraries, and the aggregated node feature information includes technical-level functional characteristics, such as video services, network services, and audio services. Its second-order neighbors are apps, and their aggregated node feature information includes usage-level functional characteristics, such as instant messaging and short video playback. These have different practical meanings compared to the technical-level functional information from first-order neighbor nodes.

Therefore, considering these limitations, we propose a third-party library recommendation model based on homogeneous graph neural networks (HGNRec). First, influenced by classical collaborative filtering, we split the heterogeneous information network (HIN) in traditional recommendation systems into two homogeneous information networks: the App information network and the third-party library information network. This approach aims to overcome the shortcomings of the aforementioned general GNN recommendation algorithms. Secondly, the two constructed GNN models will use separate aggregation and non-linear transformation network constructions, as well as adaptive aggregation, and employ edge-level and feature-level constraint methods. In a homogeneous graph, information from low-order and high-order neighbors of nodes propagates and aggregates in the same knowledge space, thereby capturing low-order and high-order interaction information of homogeneous nodes. The main contributions of this paper are as follows:

- We propose HGNRec, a third-party library recommendation model based on homogeneous graph neural networks. HGNRec splits the interaction information between Apps and third-party libraries, converting heterogeneous graph information into two homogeneous graph neural networks for Apps and third-party libraries, respectively. This approach solves the problem of heterogeneous information aggregation, simplifies and rationalizes the modeling process, and ensures that information propagation and aggregation occur within the same knowledge space.
- We incorporate a statistically-based edge construction method for node information aggregation. This method enables each node to retain interaction relationships while excluding interference from popular top-tier third-party libraries. This method efficiently updates parameters to learn structural information in the graph, thereby improving the accuracy of HGNRec recommendations.
- We meticulously collect information and construct datasets from AppBrain. Then, we conduct experiments on datasets of real Apps and third-party libraries of various scales, verifying that HGNRec's recommendations achieve higher accuracy while maintaining diversity.

The structure of this paper is as follows: The Section 2 provides an overview of related work in the research field. The Section 3 introduces the detailed methodology of the HGNRec algorithm. The Section 4 presents the results from experiments using the HGNRec algorithm. Finally, the Section 5 summarizes the entire paper and discusses prospects for future work.

2. Related work

In this section, we review related work on existing third-party library recommendation algorithms.

2.1. Recommendation based on collaborative filtering

Early third-party library (TPL) recommendation algorithms for developers primarily relied on collaborative filtering, generating recommendations based on the similarity and relevance between Apps and third-party libraries. For instance, Thung, Lo, and Lawall (2013) introduced the LibRec algorithm, integrating rule-based linkage analysis with joint recommendation systems to suggest applicable external libraries to project contributors. In 2019, Nguyen et al. (2020) introduced the CrossRec algorithm, which encodes relationships between OSS artifacts using semantic graphs and employs collaborative filtering techniques for third-party library recommendations. Furthermore, to alleviate popularity bias, He et al. (2022) proposed LibSeek, which incorporates personalized weighting mechanisms and neighborhood information based on matrix factorization (Wu, He, Wu, Zhang, & Ye, 2023; Wu, Zhong, Yao, & Ye, 2022), effectively increasing the diversity of third-party library recommendations. However, these methods lack consideration for app themes. Therefore, Yu, Xia, Zhao, and Qiu (2017) proposed a third-party library recommendation algorithm that combines collaborative filtering and topic modeling techniques, recommending libraries based on similar topic distributions of Apps.

2.2. Recommendation based on deep learning

In numerous fields, deep learning strategies are utilized (Lin, Li, Chen, Li and Wu, 2024; Lin, Li, Liu et al., 2024; Liu, Li, Wang, Li and Hang, 2024) to effectively integrate raw data with various sources of information, such as text data (Li, Deng et al., 2024) and image data (Liu, Fang et al., 2022; Zhao et al., 2024), thereby better handling sparse data (Alrubaye et al., 2020). Consequently, many deep learning-based recommendation algorithms have been proposed. CDAE (Wu, DuBois, Zheng, & Ester, 2016) is a deep learning model that integrates collaborative filtering and denoising autoencoders to improve recommendation accuracy by reconstructing user-item interactions from corrupted versions. He et al. (2017) proposed the NCF general framework, which uses neural networks to process the interaction matrix between users and items. NCF strengthens the model's non-linear ability with multilayer perceptrons, improving the efficiency and effectiveness of recommendations. NCR (Chen, Shi, Li, & Zhang, 2021) combines neural collaborative filtering with reasoning capabilities to better capture latent relationships between users and items, enhancing recommendation quality. In third-party library recommendation, building upon the general NCF framework, Huang, Xia, Xing, Lo, and Wang (2018) utilized the Word2Vec algorithm (Church, 2017) to score API names and descriptions, addressing the cold-start problem. Saied et al. (2018) proposed a third-party library reuse method based on pattern mining. This method categorizes and recommends third-party libraries by extracting usage patterns and frequencies from software development projects, thereby improving the reusability of third-party libraries and the efficiency of software development. To address the deficiency of neglecting the sequence information

Table 1
Notations and description.

Notations	Description
A_i, L_j	APP nodes and third-party library nodes
Z	Evaluation metric of the relationships of nodes
$C(A_i, A_j)$	The number of common TPLs used by node A_i with A_j
$S(A_i)$	The set of App nodes that have used common TPLs with node A_i
$\mathbf{a}_i, \mathbf{l}_j$	Vector of APP node A_i and third-party library node L_j
$\mathbf{H}$	The feature vector matrices
d	Vector dimensionality
$S^*(A_i)$	The set of first order neighbor nodes at App graph nodes A_i
$\hat{r}_{ij}$	The predicted score between App node A_i and TPL node L_i
$\mathcal{L}$	Loss function

of third-party library invocation records, Sun, Liu, Cheng, Yang, and Che (2020) proposed Req2Lib. Req2Lib employs a sequence-to-sequence method to extract library invocation records and semantic information from project requirement documents. It then utilizes a pre-trained Word2Vec model for word embedding to perform third-party library recommendations. However, these deep learning-based third-party library recommendation algorithms only utilize low-order interaction information between Apps and third-party libraries, failing to leverage high-order interaction information effectively.

2.3. Recommendation based on graph structure

Since topological information exists in recommender system data sources, graph structures are often used to model these data relationships. LightGCN LightGCN (He et al., 2020) simplifies graph convolution networks by focusing solely on the essential components, significantly improving recommendation performance and efficiency. SGL (Wu, Wang et al., 2021) integrates self-supervised tasks into graph-based recommendation models to leverage both supervised and self-supervised signals, improving recommendation accuracy. Due to the connections and relationships between nodes in the graph structure, which can be utilized to obtain high-order interaction information, some graph-based third-party library recommendation algorithms have been proposed. The GRec algorithm, proposed by Li et al. (2021), models Apps, third-party libraries, and interactions between them as an App-third-party-library graph. It then utilizes a multilayer graph neural network to extract low-order and high-order interaction information and aggregates them to recommend potentially useful third-party libraries to developers. Building upon this, to enhance training efficiency, Jin, Zhang, and Zhang (2023) proposed the NLA-GNN model, which employs simplified graph convolution operations for information propagation, updating representations of all node types. Additionally, to address GRec's deficiency in capturing fine-grained information, Zhao, Zhang, Gao, Li, and Wang (2022) proposed a knowledge graph-based convolutional network approach, which completes third-party library recommendations at both the project and library levels. Although the above methods address the limitation of traditional methods in capturing high-order interaction information, they overlook the specificity of third-party library recommendations, where third-party libraries and Apps in the graph network are two different types of nodes. These methods face the issue of heterogeneous node information fusion; hence, how to model them accordingly for recommendations is a topic worthy of research.

3. Method

To overcome the issue of heterogeneous node information fusion, HGNRec splits the interaction records between Apps and third-party libraries, models them as two homogeneous interaction graphs, and inputs them into the GNN for the capture and aggregation of homogeneous node information. As shown in Fig. 2, HGNRec consists of three main modules: The Data Preparation Module, The Information Aggregation Module, and The Rating Prediction Module.

First, based on the interaction records between existing Apps and TPLs in the training set, an App-third-party-library interaction graph is generated and used as the input for the HGNRec model. To further process this bipartite graph, HGNRec splits it into an App interaction graph and a third-party library interaction graph using statistical methods. These two graphs are isomorphic, and initial vectors are generated for the App nodes and third-party library nodes in these two graphs. Next, in the Information Aggregation Module, to capture the relationships and features between nodes, HGNRec inputs the App interaction graph and the third-party library interaction graph into GNN separately to propagate and aggregate information between homogeneous nodes. In the Rating Prediction Module, rating prediction is performed based on the feature vectors of the generated App and TPL, combined with self-learned weights. The goal of this module is to recommend the top k most useful TPLs based on given Apps, assisting developers in making better choices.

The important notations and descriptions are listed in Table 1. These constituent modules are described in detail in the next subsections to better understand the operation and advantages of the HGNRec model.

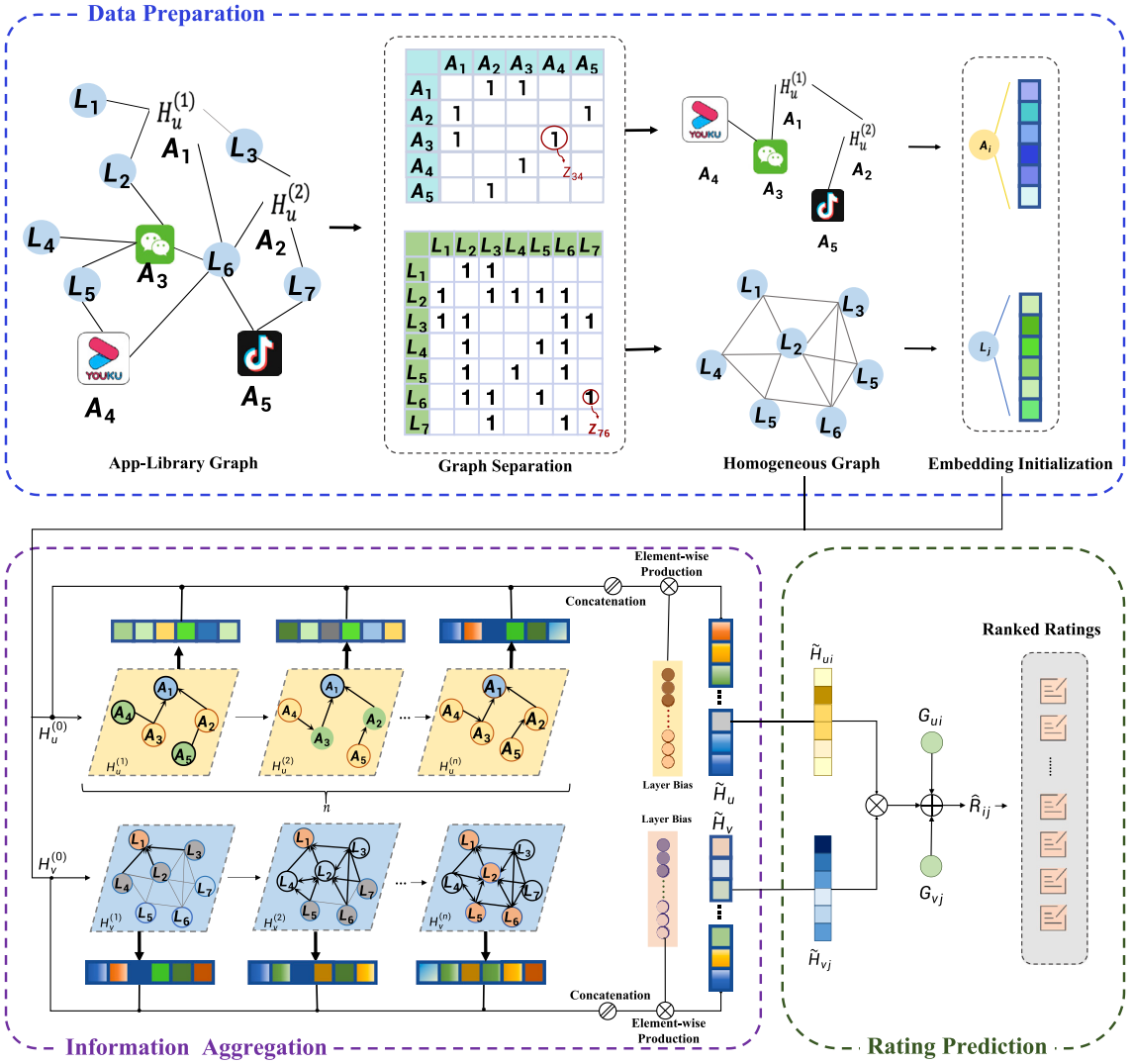

Fig. 2. The overall network architecture of HGNRec.

3.1. Data preparation

The main functions of the data preparation module involve partitioning the graph and setting up initial embeddings on the original data. First, the raw interaction data is represented by a heterogeneous network, specifically an App-third-party-library interaction bipartite graph, where App nodes are denoted by A_i and third-party library nodes are denoted by L_j. This graph is then split into an App interaction graph and a third-party library interaction graph using statistical methods. Subsequently, similar to other matrix factorization-based methods, both types of nodes in the graph are embedded into a d-dimensional space, with node features represented by a d-dimensional vector.

In real-world datasets, a pronounced long-tail distribution pattern is observed among Apps and TPLs, indicating that most Apps utilize a minority of TPLs. In contrast, the majority of these libraries are only employed by a small number of Apps. Consequently, when splitting the bipartite graph of App interactions and TPLs, connections are simply established between nodes with common first-order neighbors. This leads to most nodes in the graph being interconnected, introducing global interference and reducing information accuracy. To address this issue, when splitting the graph, HGNRec introduces a metric Z_{ij} to evaluate whether the relationship between nodes is valid based on statistical principles. This metric can help filter out more relevant node pairs, thereby improving the accuracy of graph analysis. For example, the edge between app nodes A_i and A_j will only be retained when Z_{ij} is 1; otherwise, nodes A_i and A_j are not considered first-order neighbors.

$S(A_i)$ represents the set of App nodes that have used common TPLs with node A_i and $C(A_i, A_j)$ represents the number of common TPLs used by node A_i with A_j. Thus Z_{ij} can be represented as:

$$Z_{ij} = \begin{cases} 0, C(A_i, A_j) < \frac{sum(C(A_i, A_t))}{(p-1)}, t \in S(A_i) \\ 1, C(A_i, A_j) \geq \frac{sum(C(A_i, A_t))}{(p-1)}, t \in S(A_i) \end{cases}, \tag{1}$$

where $sum()$ represents the summation function for the number of p App nodes. Similarly, for the third-party library nodes L_i and L_j, Z_{ij} can be represented as:

$$Z_{ij} = \begin{cases} 0, C(L_i, L_j) < \frac{sum(C(L_i, L_t))}{(q-1)}, t \in S(L_i) \\ 1, C(L_i, L_j) \geq \frac{sum(C(L_i, L_t))}{(q-1)}, t \in S(L_i) \end{cases}, \tag{2}$$

where $S(L_i)$ represents the set of third-party library nodes used by the common App with node L_i, $C(L_i, L_j)$ represents the number of Apps that have used TPLs L_i and L_j, and q is the number of TPLs.

Fig. 2 shows the steps of graph splitting performed by HGNRec. As an example, node A_1 shares 2 TPLs, L_3 and L_6, with node A_2 and 1 third-party library, L_6, with node A_4, while node A_1 shares 1.5 TPLs with all other nodes on average. Therefore, the edge between node A_1 and node A_2 is retained (i.e., $Z_{12} = 1$). However, the edge between node A_1 and node A_4 is discarded (i.e., $Z_{14} = 0$). By repeating the above calculations several times, we obtain the interaction matrix between the Apps and the TPLs. At this point, the App and third-party library interaction bipartite graph is split into the relationship-filtered App interaction graph and the third-party library interaction graph.

To facilitate input to the neural network, HGNRec uses d-dimensional random vectors to represent app nodes and third-party library nodes. For example, app node A_i is represented by the vector $\mathbf{a}_i \in \mathbb{R}^d$, and third-party library node L_j is represented by the vector $\mathbf{l}_j \in \mathbb{R}^d$. To increase randomness and diversity, the feature vectors are randomly initialized based on a Gaussian distribution.

3.2. Information aggregation

The Information Aggregation Module receives the App interaction graph, the third-party library interaction graph, and the node feature vectors from the Data Preparation Module as input. The module uses the App interaction graph and the third-party library interaction graph to train different graph neural networks separately. Through the propagation and aggregation operations of the multilayer neural network, it conveys and integrates the low-order and high-order interaction information between nodes.

During the training process, the embedding vectors of Apps and TPLs obtained from the Data Preparation Module are denoted as $\mathbf{a}_i^{(0)}$ and $\mathbf{l}_j^{(0)}$. They constitute the initial feature matrices $\mathbf{H}_u^{(0)}$ and $\mathbf{H}_v^{(0)}$, and $\mathbf{a}_i^{(0)}$ and $\mathbf{l}_j^{(0)}$ are the ith and jth rows of $\mathbf{H}_u^{(0)}$ and $\mathbf{H}_v^{(0)}$, where $\mathbf{H}_u^{(0)} \in \mathbb{R}^{p \times d}$, $\mathbf{H}_v^{(0)} \in \mathbb{R}^{q \times d}$, d represent the dimensions of the embedding vectors, and p and q represent the number of Apps and the number of TPLs. As shown in Fig. 2, the propagation operation between single-layer networks is first elaborated and then further generalized to multi-layer graph neural networks. Finally, the outputs of each layer neural network are concatenated and multiplied by the self-learned layer weights to obtain the final feature vector matrices $\widetilde{\mathbf{H}}_u$ and $\widetilde{\mathbf{H}}_v$.

3.2.1. First order propagation

In first-order propagation, the App node and the third-party library node will each extract low-order interaction messages from its one-hop neighbors. This process can be divided into two steps: message construction and message aggregation. In the message construction step, messages from neighboring nodes are modeled. In the message aggregation step, these constructed messages from different neighbor nodes are aggregated with the node's own features. For example, in the first-order propagation phase, App node A_1 will aggregate messages from nodes A_2, A_3, and TPL node L_1 will aggregate messages from nodes L_2, L_3, L_4 in Fig. 2.

Firstly, take the App node as an illustration; during the process where this node consolidates information from another node, it first performs the message construction, and the calculation formula is as follows:

$$\mathbf{m}_{i \leftarrow r} = \frac{1}{\sqrt{|S^*(A_i)||S^*(A_r)|}} \left(\mathbf{W}_1 * \mathbf{a}_r^{(0)} + \mathbf{W}_2 * \left(\mathbf{a}_r^{(0)} \odot \mathbf{a}_i^{(0)} \right) \right), \tag{3}$$

where $S^*(A_i)$ and $S^*(A_r)$ denote the set of first-order neighbor nodes at App graph nodes A_i and A_r. $\mathbf{W}_1$, $\mathbf{W}_2$ are the weight matrices learned during the training process and they are used to control the influence of neighbor node information. $\frac{1}{\sqrt{|S^*(A_i)||S^*(A_r)|}}$ represents the Laplacian paradigm in graph convolutional networks, which is used to balance the size of the neighbor sets of different nodes. $\mathbf{a}_r^{(0)} \odot \mathbf{a}_i^{(0)}$ represents the encoding of the information about the interactions between the nodes. Here, $\odot$ denotes the product of the elements, i.e., the multiplication of the elements at the corresponding positions to capture the correlation and similarity between the nodes.

After completing the construction of messages from all the nodes in the neighborhood, these messages are converged with the information from the node itself to update the feature representation of the node. The aggregation formula is shown below:

$$\mathbf{a}_i^{(1)} = Sigmoid(\mathbf{W}_1 * \mathbf{a}_i^{(0)} + \sum_{r \in S^*(A_i)} \mathbf{m}_{i \leftarrow r}), \tag{4}$$

where $\mathbf{a}_i^{(1)}$ denotes the feature representation obtained by node A_i after the first layer of the network. $Sigmoid()$ is the activation function, which aims to scale and normalize the node features. $\mathbf{W}_1 * \mathbf{a}_i^{(0)}$ denotes the node's original feature information, and $\sum_{r \in S^*(A_i)} \mathbf{m}_{i \leftarrow r}$ denotes the aggregation of messages from the first-order neighboring nodes.

Next, the same technique is applied to revise the feature representations for the TPL nodes. Once the update is complete, the embedding vectors of the App and the TPL output by the first-order network form the first-order propagation feature matrices $\mathbf{H}_u^{(1)}$ and $\mathbf{H}_v^{(1)}$.

3.2.2. High order propagation

Based on first-order neighbor aggregation, more layers can be added to the network, with each layer utilizing the feature representation of the previous layer. The higher-order connectivity of the nodes is captured through information propagation and aggregation over many iterations so that the information of higher-order neighboring nodes is gradually fused in the feature representation of the nodes to increase the richness of the feature representation. Assuming an n-layer graph neural network is employed, in each layer, the feature representation of a node will gradually fuse information from neighboring nodes with up to n hops. In this way, the final feature representation of a node will contain information from a broader range of neighboring nodes, making the feature representation of a node more globally contextual and abundant.

Specifically, the feature $\mathbf{a}_i^{(n)}$ of the A_i node at the nth layer of the network is computed as follows:

$$\mathbf{a}_i^{(n)} = Sigmoid(\mathbf{W}_1^{(n)} * \mathbf{a}_i^{(n-1)} + \sum_{r \in S^*(A_i)} \mathbf{m}_{i \leftarrow r}^{(n)}). \tag{5}$$

The higher-order aggregated message definition in the above formulation can be obtained by extending the first-order propagated message definition, which can be finally illustrated as follows:

$$\mathbf{m}_{i \leftarrow r}^{(n)} = \frac{1}{\sqrt{|S^*(A_i)||S^*(A_r)|}} \left(\mathbf{W}_1^{(n)} * \mathbf{a}_r^{(n-1)} + \mathbf{W}_2^{(n)} * \left(\mathbf{a}_r^{(n-1)} \odot \mathbf{a}_i^{(n-1)} \right) \right), \tag{6}$$

where $\mathbf{W}_1^{(n)}$ and $\mathbf{W}_2^{(n)}$ are the self-learned weight matrices from training, and $\mathbf{a}_r^{(n-1)}$, $\mathbf{a}_i^{(n-1)}$ are both the feature vectors output from the $n-1$th layer of the network. In this way, the feature representation of node A_i in the nth layer network is obtained.

Similarly, the TPL node features L_j at the nth layer of the network can be computed as follows:

$$\mathbf{l}_j^{(n)} = Sigmoid(\mathbf{W}_1^{(n)} * \mathbf{l}_j^{(n-1)} + \sum_{r \in S^*(L_j)} \mathbf{m}_{j \leftarrow r}^{(n)}), \tag{7}$$

where the definition of the higher-order aggregated message of the TPL node is aligned with the TPL node and can be denoted as:

$$\mathbf{m}_{j \leftarrow r}^{(n)} = \frac{1}{\sqrt{|S^*(L_j)||S^*(L_r)|}} \left(\mathbf{W}_1^{(n)} * \mathbf{l}_r^{(n-1)} + \mathbf{W}_2^{(n)} * \left(\mathbf{l}_r^{(n-1)} \odot \mathbf{l}_j^{(n-1)} \right) \right). \tag{8}$$

Following the above calculations, the feature matrices from layer 2 to layer n are obtained separately, denoted as $\mathbf{H}_u^{(2)}, \ldots, \mathbf{H}_u^{(\mathrm{n}-1)}$, $\mathbf{H}_u^{(\mathrm{n})}$ and $\mathbf{H}_v^{(2)}, \ldots, \mathbf{H}_v^{(\mathrm{n}-1)}, \mathbf{H}_v^{(\mathrm{n})}$.

Taking the TPL node L_1 in Fig. 2 as an example, the third-order propagation process with path $L_6 \rightarrow L_5 \rightarrow L_2 \rightarrow L_1$. Along this path, higher-order feature information is injected into the feature representation of node L_1 through multiple message construction and aggregation operations.

3.2.3. Aggregation

After completing the first-order and higher-order propagation operations, the primary node attributes sourced from the data preparation process are merged with the sequential outputs of the graph neural network layers, and the final feature depiction of each node is formed. In an interaction graph, neighbors with closer proximity are considered to have a stronger correlation, and the inter-node distance should also be considered when aggregating the nodes' feature representations from each layer. Therefore, HGNRec learns a global layer weight vector $\mathbf{w}_l$ during training, where each term is a scalar between (0,1) to evaluate how the output from each layer contributes to the final depiction of the nodes' features.

The aggregation process unfolds in two phases: initially, features across layers are unified, followed by the multiplication of these unified vectors with their corresponding layer weights. Taking App node A_i as an example, its initial vector $\mathbf{a}_i^0$ is obtained through the data preparation module. Through the first-order propagation and higher-order propagation, the features $\mathbf{a}_i^1, \mathbf{a}_i^2, \ldots, \mathbf{a}_i^n$ in the total n-layer network can be obtained. The feature representation obtained by the aggregation of node A_i is denoted as $\mathbf{a}_i^*$, which is computed as follows:

$$\mathbf{a}_i^* = \left(\mathbf{a}_i^0 \parallel \mathbf{a}_i^1 \parallel \cdots \parallel \mathbf{a}_i^n \right) \otimes \mathbf{w}_l, \tag{9}$$

where $\parallel$ represents the vector concatenating operation.

Similarly, the feature representation of the aggregated TPL node L_j is denoted as $\mathbf{l}_j^*$ and defined as follows:

$$\mathbf{l}_j^* = \left(\mathbf{l}_j^0 \parallel \mathbf{l}_j^1 \parallel \cdots \parallel \mathbf{l}_j^n \right) \otimes \mathbf{w}_l. \tag{10}$$

The feature representations of each App node and TPL node are calculated for the aggregation operation, and the final App feature matrix $\widetilde{\mathbf{H}}_u$ and TPL feature matrix $\widetilde{\mathbf{H}}_V$ can be obtained.

3.3. Ratings prediction

The feature representations of App and TPL nodes are obtained through the Information Aggregation Module, which will be the input to the score prediction module. In the HGNRec method, the score of an App node and TPL node pair consists of the feature representation product and the global offset of the node. The feature representation product measures the match between an App node and a TPL node by calculating the inner product of their feature vectors. To address the bias problem in recommender systems, a global offset is introduced for each App and TPL, which reflects the average tendency of Apps to use TPLs and TPLs to be used by Apps. By integrating the feature representation product and the global offset, the HGNRec method can more accurately assess the degree of association between App nodes and TPL nodes, thus providing more precise and personalized recommendation results. Specifically, the score between App node A_i and TPL node L_i can be calculated as follows:

$$\hat{r}_{ij} = \mathbf{a}_i^* \cdot \mathbf{l}_j^* + b_{Ui} + b_{Vj}, \tag{11}$$

where $\cdot$ denotes the inner product operation of the vector, $\mathbf{a}_i^*$ is the feature representation obtained by the aggregation of node A_i, $\mathbf{l}_j^*$ is the feature representation of the aggregated TPL node L_j, b_{Ui} is the global offset of the App node, and b_{Vj} is the global offset of the TPL node.

3.4. Optimization

HGNRec constructs the scoring loss to learn the model parameters using the cross-entropy loss function widely used in recommender systems. Predicting the App and TPL association score can be interpreted as predicting the probability of whether there is a connection between an App node and a TPL node. Specifically, the cross-entropy loss function is based on the concept of cross-entropy in information theory, which measures the probability distribution of the model's output against the probability distribution of the actual score to calculate the loss. The objective function is as follows:

$$\mathcal{L} = -\sum_{i=0}^{p}\sum_{j=0}^{q} r_{ij} \cdot \log\left(\hat{r}_{ij}\right) + \left(1 - r_{ij}\right) \cdot \log\left(1 - \hat{r}_{ij}\right) + \lambda\|\theta\|_2^2, \tag{12}$$

where r_{ij} is the true rating of App node A_i and TPL node L_j, i.e., the rating is 1 if there is an interaction between the two, and 0 vice versa. λ is the L2 regularization parameter set, and θ denotes the set of all trainable parameters in the model. The training process of the HGNRec model is as illustrated by the algorithm 1.

Algorithm 1: The training procedure for HGNRec.

Input : The App set $\mathcal{A}$, the TPL set $\mathcal{L}$ and the heterogeneous interaction graph $\mathcal{G}(\mathcal{A}, \mathcal{L})$, epoch number N_e, batch number N_b

```
Calculating Z_ij and splitting graph as Section 3.1 ;
Initialize App embedding A and TPL embedding L randomly ;
for i ← 1 to N_e do
    Sample a mini batch from A and L ;
    for j ← 1 to N_b do
        m^(n)_{i←r}, m^(n)_{j←r} ← High Order Message Propagation // Obtain message from neighbor ;
        a^(n)_i, l^(n)_j ← Feature Updating after n-layer network // Introduce multi-layer message ;
        a*_i, l*_j ← Feature Aggregation // Combine multi-layer feature;
        Calculate loss as Equation (12) and update model parameters ;
    end
end
```

Output: HGNRec model

4. Experiments

To evaluate the HGNRec technique, we conducted experiments on three real-world datasets to investigate the subsequent three research questions:

- RQ1: How is the effectiveness of HGNRec compared to other third-party library?
- RQ2: Does the isomorphic interaction graph construction method based on average threshold improve the performance of HGNRec?
- RQ3: What is the effect of different hyper-parameter configurations, like embedding vector dimensions, the number of layers in the graph network, and the activation function selection, on the effectiveness of HGNRec?

4.1. General settings

4.1.1. Evaluation metrics

In the experiment, to provide a comprehensive assessment of the recommended outcomes of HGNRec, we carefully selected four widely used indicators and compared them with other related studies. The following are the selected indicators and their meanings:

- Mean Precision (MP)
  In each experiment, each App will be given a list of TPL recommendations, and the precision represents the proportion of TPLs that are truly relevant to the recommendation results. Based on the precision, MP represents the average precision of all recommendation lists in an experiment. It can be formulated as:

$$MP@K = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \frac{|T_{ground} \cap T_K|}{K}, \tag{13}$$

  where T_{ground} denotes the ground-truth TPLs of APP a in the test set, T_K represents the TPLs in the recommend list.
- Mean Recall (MR)
  The recall represents the ratio of truly relevant TPLs in the recommendation results to all TPLs removed from the corresponding App. Based on the recall rate, MR represents the average recall rate of all recommendation lists in one experiment. It can be formulated as:

$$MR@K = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \frac{|T_{ground} \cap T_K|}{|T_{ground}|}. \tag{14}$$

- Hit Rate (HR)
  HR represents the ratio of the number of Apps from TPLs in the test set that appear in the recommended list to the total number of Apps. It can be formulated as:

$$HR@K = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} hit, hit = \begin{cases} 0, & T_{gound} \cap T_K = \emptyset \\ 1, & else \end{cases}, \tag{15}$$

  where hit denotes whether the recommend list contains the ground-truth TPLs.
- Normalized Discounted Cumulative Gain (NDCG)
  NDCG is the Normalized Discounted Cumulative Gain, a crucial metric for evaluating the quality of recommendation result ranking. NDCG measures the quality of the TPL recommendation results by considering the relationship between the relevance of the recommendation results and the ranking position, and the closer the value is to 1, the higher the accuracy of the recommendation result ranking is. It can be formulated as:

$$NDCG@K = \frac{1}{|\mathcal{A}|} \sum_{a \in A} \frac{DCG@K}{IDCG@K}, \tag{16}$$

$$DCG@K = \sum_{j=1}^{K} \frac{rel\,(j)}{log_2\,(j+1)}, rel\,(j) = \begin{cases} 1, & TPL_j \in T_{ground} \\ 0, & else \end{cases}, \tag{17}$$

  where $rel\,(j)$ denotes whether TPL_j is one of the TPLs in T_{ground}, $IDCG@K$ is the ideal value of recommendation results where the ground-truth TPLs rank first.

When calculating the above metrics in the experiments, the top K TPLs with $K \in \{5, 10\}$ were recommended for each test App rating ranking, respectively.

4.1.2. Datasets

We collect information about 667,233 Android mobile Apps from the AppBrain website, containing entries such as the name, classification, and description of the App with the TPLs. AppBrain is the leading source for information about Android apps. It maintains information about all apps on Google Play, resources for developers, and statistics about the Android ecosystem. Each App record contains its name, category, TPLs, and other meta information. To ensure the validity of the data, we manually cleaned the App-TPL usage records and finally obtained 572 different TPLs as well as 1,139,323,322 App-TPL usage records, and the distribution of the usage relationship is shown in Fig. 3.

In order to effectively evaluate the HGNRec method, we divide three branch datasets based on the crawled total dataset. Firstly, we filter out the Apps with TPL usage records, then we extract 1% and 4% of the Apps using the random sampling method to construct App_1% and App_4% datasets and select all the Apps categorized as sports to construct the App_Sports dataset, and the specific data of the three datasets are shown in Table 2. In the experiment, following the evaluation method of NGCF (Wang, He, Wang, Feng, & Chua, 2019), for each dataset, we randomly allocate 80% historical interaction data of each App to the training set, with the remainder used as the test set to assess effectiveness.

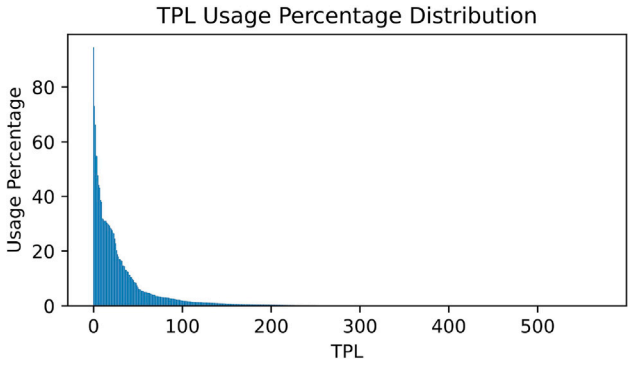

Fig. 3. Distribution of App-TPL Usage Relationships.

Table 2
Statistics details of the datasets for third-party library recommendation.

Dataset	Apps	TPLs	Interactions	Density
App_1%	7565	470	129,581	0.9636
App_4%	30,258	517	516,859	0.9669
App_Sports	23,186	457	469,356	0.9557

4.2. RQ1: Performance comparison

To verify the efficacy of HGNRec's suggestions, we chose the five benchmark methods outlined below, which contain the baseline method with the current state-of-the-art TPL recommendation methods:

- **POP**: The method ranks the popularity of TPLs according to their usage history with the App, and recommends the most popular TPLs for the test App that it does not use. We use this method as a baseline method for evaluation.
- **BPR** (Rendle, Freudenthaler, Gantner, & Schmidt-Thieme, 2009): The approach is based on the idea of pairwise comparison, using matrix decomposition to obtain the features of the App and TPLs, and ranking the TPLs by Maximum Bayesian posterior probability to produce recommendation results.
- **NGCF** (Wang et al., 2019): The approach uses a multilayer graph neural network to capture the cooperative signals and higher-order connectivity information in the interaction graph to complement the node features for recommendation.NGCF is regarded as a classic among recommendation algorithms that utilize graph-based frameworks.
- **CrossRec** (Nguyen et al., 2020): CrossRec is a collaborative filtering-based recommender system, utilizing the inverse document frequency to calculate similarity among Apps.
- **LibSeek** (He et al., 2022): This method is an advanced algorithm designed to provide TPL recommendations for Android Apps. LibSeek is based on matrix decomposition and incorporates a personalized weighting mechanism with neighborhood information to recommend applicable TPLs.
- **Grec** (Li et al., 2021): It models the interaction between the Apps and TPLs as an interaction graph, and uses a multi-layer graph neural network to extract higher-order domain information from the graph.
- **HGNRec**: The model proposed in this paper. HGNRec splits the interaction information between Apps and third-party libraries, namely heterogeneous graph information, into two homogeneous graph neural networks for Apps and third-party libraries, respectively. Then, the low-order and high-order neighbor information of nodes are propagated and aggregated in the same knowledge space.

To maintain comparative fairness, the parameter settings for all methods in the experiments are aligned with those specified in Grec. The embedding space dimension is fixed at 128, the number of graph neural network layers is 3, the Adam method is used for optimization and the learning rate is set to 0.0002, and the batch size is fixed at 512.

Table 3 presents the experimental results of our method HGNRec with the remaining five methods on three datasets at vs. time respectively, where the optimal results are bolded. Drawing from these results, we arrive at the following insights:

- POP performs significantly behind the other methods on all datasets, which may be because it only relies on the popularity of TPLs to accomplish recommendations. The popular TPLs tend to be general-purpose tool libraries, which are difficult to satisfy the targeted functionality needs of App developers in the real world during the development process.BPR outperforms POP in all cases, proving the importance of implicit feedback features. However, there is still an obvious gap between the results of BPR and the other four methods, indicating that the collaborative filtering algorithm is insufficient to extract the complex invocation relationships between App and TPLs and cannot accurately portray their features.

Table 3
Performance results of different methods.

Dataset	Models	K = 5				K = 10			
		MP	HR	MR	NDCG	MP	HR	MR	NDCG
App_1%	POP	0.4254	0.8565	0.4640	0.4995	0.304	0.8875	0.6265	0.5916
	BPR	0.4382	0.8183	0.4601	0.5465	0.2862	0.8427	0.5714	0.6257
	LibSeek	0.5598	0.9493	0.6418	0.6812	0.3572	0.9677	0.7703	0.7561
	NGCF	0.5653	0.9577	0.6506	0.7541	0.3662	0.9755	0.7919	0.7719
	CrossRec	**0.6140**	0.9362	0.2536	0.6851	**0.4611**	0.9558	0.3585	0.5511
	Grec	0.5859	0.9619	0.6728	0.8442	0.3743	0.9781	0.8078	0.8976
	HGNRec	0.5926	**0.9621**	**0.6783**	**0.8533**	0.3766	**0.9787**	**0.8094**	**0.9017**
App_4%	POP	0.3345	0.8018	0.4899	0.4834	0.2308	0.8297	0.6100	0.5492
	BPR	0.3523	0.8174	0.5122	0.5092	0.2373	0.8509	0.6327	0.5738
	LibSeek	0.4004	0.8528	0.5833	0.5737	0.2567	0.9013	0.7085	0.6349
	NGCF	0.4422	0.9112	0.6669	0.7034	0.2804	0.9399	0.7862	0.7261
	CrossRec	**0.4814**	0.8220	0.2956	0.5518	**0.3505**	0.8378	0.3848	0.4348
	Grec	0.4653	0.9208	0.6964	0.7955	0.2897	0.9458	0.8073	0.8404
	HGNRec	0.4750	**0.9274**	**0.7096**	**0.8182**	0.2924	**0.9506**	**0.8144**	**0.8584**
App_Sports	POP	0.3568	0.8012	0.4514	0.4690	0.2504	0.8331	0.5758	0.5391
	BPR	0.3632	0.8006	0.4565	0.5104	0.2465	0.8193	0.5614	0.5811
	LibSeek	0.4722	0.9007	0.6192	0.6187	0.3145	0.9353	0.7596	0.6956
	NGCF	0.5089	0.9318	0.6733	0.7525	0.3338	0.9575	0.8112	0.7687
	CrossRec	0.5049	0.9094	0.3228	0.5758	**0.3905**	0.9227	0.4332	0.4701
	Grec	0.5120	0.9325	0.6777	0.8027	0.3332	0.9567	0.8115	0.8523
	HGNRec	**0.5208**	**0.9340**	**0.6886**	**0.8216**	0.3372	**0.9623**	**0.8203**	**0.8673**

- Compared with POP and BPR, the effect of LibSeek gains a substantial improvement. This verifies that the introduction of an adaptive weighting mechanism based on matrix decomposition effectively alleviates the problem of popularity bias of TPLs and improves the representation learning between App and TPLs. However, LibSeek only uses the low-order interaction relationship between App and TPLs and ignores the high-order connectivity information, so there is still room for improvement in representation learning.
- NGCF obtains significantly superior results to LibSeek on all datasets, which validates the importance of higher-order connectivity information in learning App and TPL features. Further, GRec models Apps, TPLs, and their interactions as App-TPL graphs and uses NGCF to extract both low-order and advanced neighbor data from them, thereby improving feature detection. CrossRec only utilizes the interactions to calculate similarities among Apps. Although CrossRec can improve the MP because it may recommend popular items, it does not perform well on other metrics.
- HGNRec achieves consistently the best performance on most metrics of all datasets. Specifically, HGNRec gains 1.9% to 44.8%, and 2.1% to 56.3% on the MR@5 and NDCG@10 metrics on the App_4% dataset, respectively. By splitting the App-TPL interaction graph and constructing the App-App interaction graph and the TPL-TPL interaction graph, HGNRec can extract and converge information of the same knowledge type using a multi-layer graph neural network. It verifies the importance of considering the type of node information when performing graph node feature aggregation. In addition, compared to Grec, HGNRec considers the use of a weighted splicing mechanism when aggregating the layers of the graph neural network, while Grec only directly splices the outputs of each layer to complete the aggregation. This suggests that features from different propagation layers have different correlations with nodes. In addition to this, HGNRec incorporates global offset features from App and TPLs. The improvement over NGCF and Grec, which also use graph neural networks, suggests that explicitly modeling the recommendation global bias in the node features results in a better learning representation.

4.3. RQ2: Effect of homogeneous graph construction methods

As introduced in the Data Preparation module of the Methods section, in order to rationally split the App-TPL interaction graph into 2 isomorphic interaction graphs, HGNRec chooses a statistical method based on average thresholding to construct the interaction relationships between App and App, and TPL. To investigate the impact of different isomorphic graph edge construction methods on HGNRec's recommendation results, we try three other different construction methods: complete retention of all interactions, random sampling, and weighted edge construction, and conduct experiments on the App_1% dataset with reference to the parameter settings of RQ1, and the results of the experiments are presented in Fig. 4.

We find that the statistical method based on average thresholding obtains significantly better results than the remaining three edge construction methods for all three metrics, MP, HR, and MR, for either K = 5 or 10. This advantage is due to the fact that the average number of co-interacting objects between Apps and TPLs is calculated and used as a threshold for determining whether to retain App-to-App and TPL-to-TPL interactions. In this way, we are able to exclude interference from those header generic TPLs that are widely used, while retaining interaction objects related to specific functions. In general, these objects with stronger function-specific relevance enable HGNRec to more accurately extract the features of Apps and TPLs, and more efficiently produce recommendation results that satisfy the targeted functional needs of App developers.

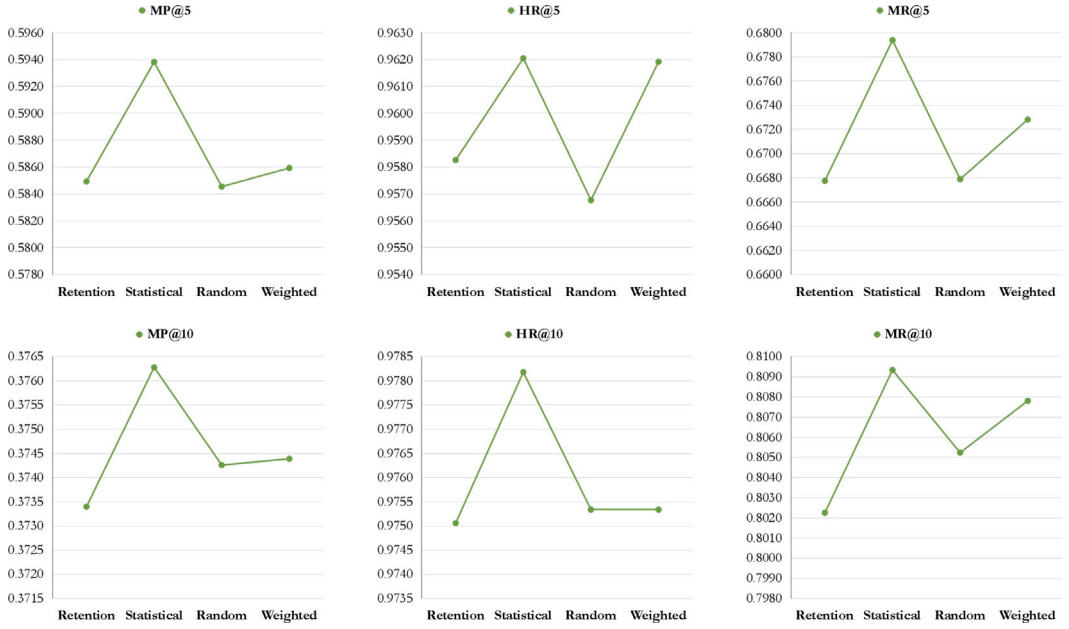

Fig. 4. Comparison of the results of the homogeneous compositional edge construction methods.

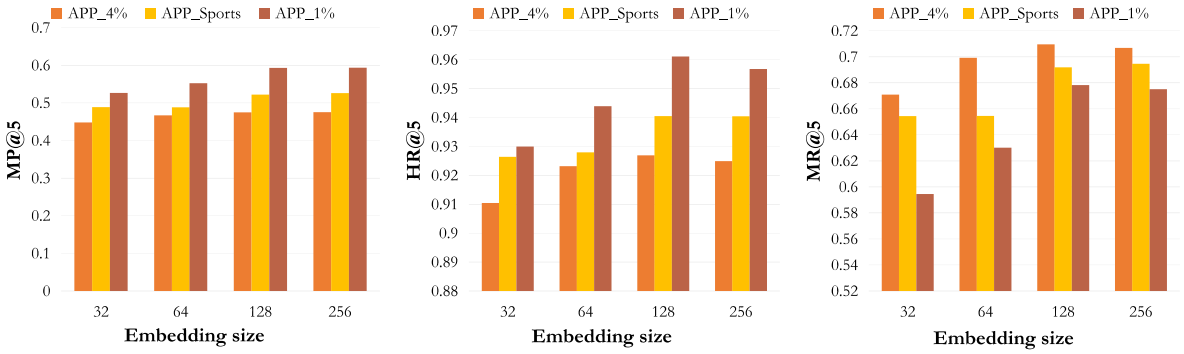

Fig. 5. Effect of embedding dimension d on HGNRec recommendation results.

4.4. RQ3: Study of HGNRec

As shown in Fig. 2, HGNRec is composed of three modules: Data Preparation, Information Aggregation, and Rating Prediction. For the Data Preparation Module, we have discussed the impact of different isomorphic interaction graph construction methods on recommendation results in RQ2. In this section, we will discuss some settings, such as the choice of vector embedding dimension and activation function, in the modules of Information Aggregation and Rating Prediction, and their effects on the recommendation results of HGNRec.

4.4.1. Effect of embedding dimension

HGNRec embeds Apps and TPLs as low-dimensional vectors in the feature space during training to represent their features and interactions. The embedding vector dimension is one of the key parameters affecting the recommendation results of the model. A low embedding dimension may lead to information loss and limit the model's representation capability, while a high embedding dimension may lead to increased computation and storage costs, and the vector space may become too sparse, resulting in over-fitting of the recommendation results. To investigate how varying embedding dimensions impact HGNRec's recommendation accuracy, we adjusted the dimension d between 32 and 256. The experimental results are displayed in Fig. 5, with the best results for each dataset marked accordingly.

When d increases, the performance of HGNRec on the MP@5 metric increases in all three datasets as a consequence. However, when considering the HR@5 metric, the performance of HGNRec takes the maximum value at $d = 128$. This may be due to the fact that the addition of the embedding dimension improves the representational power and diversity of results of HGNRec and increases

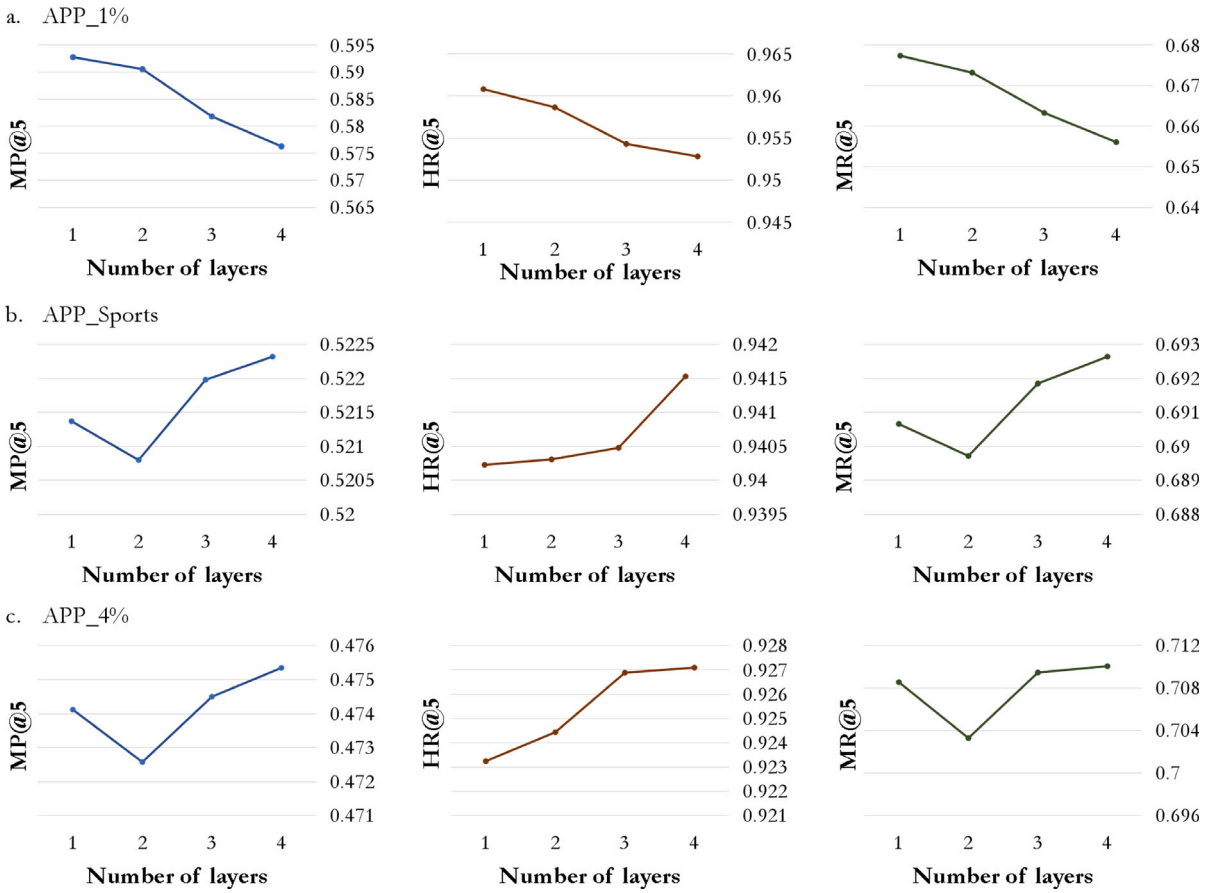

Fig. 6. Effect of the number of GNN layers n on HGNRec recommendation results.

the average accuracy of the recommendation results. However, the increase of embedding dimension may lead to the increase of feature error of those Apps with extremely sparse interaction data with TPLs, which leads to the decrease of hit rate. Interestingly, as far as the MR@5 metric is concerned, the effect of d varies across datasets. On the App_1% and App_4% datasets, the performance of HGNRec is optimal at $d = 128$, while on the App_Sports dataset, the performance of HGNRec is optimal at $d = 256$. We analyze that this could be attributed to the fact that Apps in the sports category have richer features and thus higher dimensions achieve better performance, which is also in line with our previous experimental results that the MP@5 metric is optimal at $d = 256$ on the App_Sports dataset.

4.4.2. Effect of layer numbers

In order to obtain higher-order connectivity information in App-App interaction graphs and TPL-TPL interaction graphs, HGNRec employs multi-layer GNN for graph node feature extraction. To explore the variation of HGNRec's performance under different GNN layers, we set the number of network layers of HGNRec within the range to carry out the experiments and keep the default parameter settings in RQ1 in each experiment. Fig. 6 illustrates the experimental results. From the analysis of the three datasets, we make the following deductions:

- On the App_1% dataset, all the observable metrics, that is, MP@5, HR@5, and MR@5, achieve the best results at $n = 1$. Combined with the data in Table 1, this may be due to the smaller size of App_1%, where the use of more layers of GNN resulted in the model overfitting the training data, leading to a degradation in the performance of HGNRec on the test data. This observation emphasizes the need for caution in choosing the number of GNN layers when using the HGNRec method in practice, especially when the dataset size is small. To determine the optimal number of GNN layers, experiments and adjustments need to be made based on the actual dataset.
- On the App_Sports dataset and App_4% dataset, we observe that all metrics have optimal results at $n = 4$. The experimental data indicate that adding more layers to the graph network for these datasets enhances the recommendation performance of HGNRec. This is because the higher dimensional feature space allows HGNRec to model more features from Apps and TPLs. The propagation and aggregation of data from more distant nodes can capture the dependencies and global information between the App and the TPLs more effectively, thus improving the accuracy of the recommendation results.

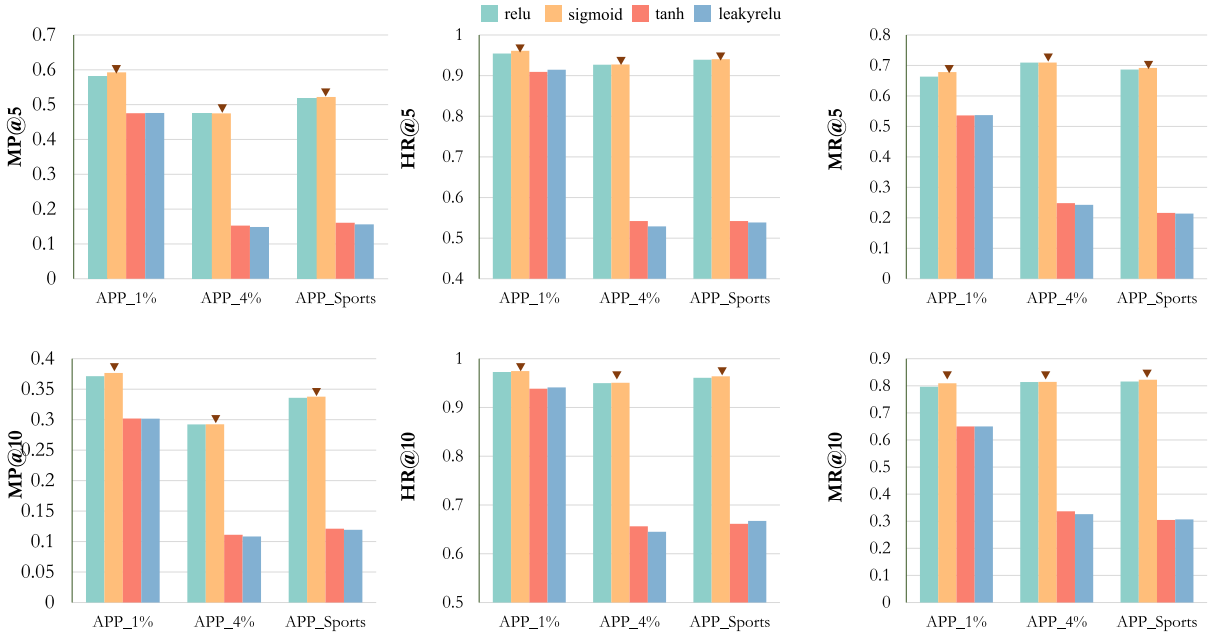

Fig. 7. Effect of activation function on HGNRec recommendation results.

4.4.3. *Effect of activation function*

Activation functions play an important role in graph neural networks to enhance the model's representation of nonlinear features. To help HGNRec better capture the complex relational features of App and TPLs, we add activation functions after each layer of GNN in the Information Aggregation module. There are several choices of mainstream activation functions, which have different features and transformations. To explore the impact of different feature activation functions on the HGNRec recommendation results, we select four activation functions, relu, sigmoid, tanh and leakrelu, and launch experiments on each of the three datasets. The default parameter settings in RQ1 are kept in each experiment. The experimental findings are presented in Fig. 7, with triangles indicating the superior outcomes.

By comparison experiments, we can observe that the sigmoid activation function achieves better results on all datasets, while tanh and leakyrelu obtain bad performance. This may be due to the fact that the sigmoid activation function normalizes the features and avoids the effect of too large or too small feature values for TPLs with extreme usage, such as those used by 95% of the Apps versus those used by 0.01% of the Apps.

Another notable finding is that employing various activation functions in the App_1% and App_Sports datasets markedly influences the recommendation performance of HGNRec. Taking the MR@10 metric as an example, using the sigmoid function improves 1.60% and 0.86% relative to the relu function on the App_1% and App_Sports datasets, respectively, while it brings only 0.074% improvement on the App_4% dataset. This difference may be due to the fact that the App_1% and App_Sports datasets have a more pronounced continuity relationship. The smoothing of the sigmoid function helps to capture this type of data with a continuity feature, which leads to a more significant enhancement in the recommendation results.

4.4.4. *Effect of global bias*

In the Rating Prediction module of HGNRec, for the common recommendation bias problem in recommender systems, we introduce a self-learning global offset to model the average tendency of App usage and TPLs being used. To verify whether global offsets alleviate the recommendation bias problem and improve the final recommendation results of HGNRec, The ablation experimental results are displayed in Table 4.

The experimental results show that when the global bias in App and TPL ratings is explicitly removed, the recommendation effect decreases significantly. This proves that HGNRec uses self-learning scalars to model the recommendation bias of Apps and TPLs, which can significantly enhance the accuracy of the recommendations. In addition, combining the results in Table 3, HGNRec still exhibits better performance than the other compared methods even when the global offset is removed. This also empirically shows that avoiding heterogeneous node information fusion can effectively improve recommendation results.

5. Discussion

5.1. *Results analysis*

HGNRec takes the heterogeneous App interaction information graph with third-party libraries, splits it into isomorphic graphs, and combines it with a statistically-based edge construction method for node information aggregation. By comparing HGNRec with

Table 4
Experimental results of global offset ablation.

Dataset	Bias	K = 5				K = 10			
		MP	HR	MR	NDCG	MP	HR	MR	NDCG
App_1%	×	0.5869	0.9494	0.6565	0.9519	0.3707	0.9656	0.7888	0.9724
	✓	0.5926	0.9621	0.6783	0.8533	0.3766	0.9787	0.8094	0.9017
App_4%	×	0.4693	0.9245	0.6944	0.8092	0.28595	0.94163	0.80628	0.84533
	✓	0.4750	0.9274	0.7096	0.8182	0.2924	0.9506	0.8144	0.8584
App_Sports	×	0.5151	0.9390	0.6791	0.8087	0.3366	0.9596	0.8114	0.8550
	✓	0.5208	0.9340	0.6886	0.8216	0.3372	0.9623	0.8203	0.8673

state-of-the-art third-party library recommendation methods, the experimental results demonstrate its superiority. HGNRec better aggregates information under the same modeling space, outperforming the baseline approach on several datasets in the overall comparison. In RQ2, it is experimentally demonstrated that the statistically based homomorphic edge construction method is better at excluding interference from popular head third-party libraries. In addition, with parameter tuning, HGNRec can achieve excellent performance on datasets of various magnitudes and categories. Comparing HGNRec with the baseline method, the main advantages of our work are: (1) By splitting the heterogeneous graph interaction information between Apps and third-party libraries into homogeneous graphs, our method can overcome the difficulties in aggregating the node information of the heterogeneous graphs; (2) The statistically-based edge construction method for node aggregation allows each node to preserve the interactions to exclude the interferences from the third-party libraries of the popular headers; (3) The parameters of our method can be easily adjusted to adapt to different sizes and types of datasets with generalization.

5.2. Theoretical and practical implications

From a theoretical perspective, our proposed HGNRec optimizes the node information aggregation based on the homomorphic graph network to better utilize the interaction information between App and third-party libraries. First, HGNRec splits the heterogeneous interaction information graph of the App with third-party libraries into interaction homography graphs. Subsequently, HGNRec combines the statistically-based edge construction method for node information aggregation, which effectively excludes the interaction interference from popular third-party libraries. In addition, the offset introduced by HGNRec can effectively characterize the calling tendency of each different dataset. HGNRec was evaluated with state-of-the-art third-party library recommendation methods, and the results demonstrate its superior performance.

In the past, in traditional graph-based third-party library recommendation methods, the aggregated information may come from the data in two nodes that are not in the same knowledge space, which leads to a decrease in the recommendation effect. Moreover, these methods are easily influenced by popular third-party libraries, resulting in a single recommendation result. Therefore, this paper mainly splits heterogeneous interaction graph information into homogeneous graphs based on homogeneous graph networks, which makes the modeling and aggregation space unified, and uses a statistical-based method to construct the edges of homogeneous graphs, which reduces the noise influence caused by popular third-party libraries. In this way, our method is able to achieve better recommendation results in real datasets.

6. Conclusion

We propose HGNRec, a third-party library recommendation model based on homogeneous graph neural networks.HGNRec solves the heterogeneous information aggregation problem by splitting the heterogeneous graph of third-party library-App interaction information into two homogeneous graph neural networks for the App and the third-party library, respectively. Further, it combines a statistically-based edge construction method for node information aggregation, which allows each node to retain interaction relations that exclude interference from popular head third-party libraries. By comparing HGNRec with state-of-the-art recommendation algorithms, the experimental results demonstrate its superiority.

We hope that this paper will provide new ideas for modeling third-party library-App interactions. Although this paper better aggregates node information through a homography network, it uses random initialization for node information, which does not effectively utilize the description information in the actual scene. Besides, introducing graph splitting brings the extra time cost, which can be further optimized. In subsequent work, we plan to incorporate details about Apps and external libraries to improve the aggregation of information and the clarity of our suggestions.

CRediT authorship contribution statement

Duantengchuan Li: Writing – review & editing, Writing – original draft, Supervision, Software, Methodology, Formal analysis, Data curation, Conceptualization, Funding acquisition, Project administration. **Yuxuan Gao:** Writing – original draft, Software, Formal analysis. **Zhihao Wang:** Writing – review & editing, Software, Methodology, Conceptualization, Data curation, Writing – original draft. **Hua Qiu:** Software, Investigation, Formal analysis, Data curation. **Pan Liu:** Conceptualization, Supervision, Writing – review & editing. **Zhuoran Xiong:** Validation, Visualization, Writing – review & editing. **Zilong Zhang:** Software, Writing – review & editing.

Data availability

Data will be made available on request.

Acknowledgments

The authors would like to thank the following people for their support and work in building the dataset for this article: Zhixiong Wu (2019 undergraduate students, School of Computer Science, Wuhan University, Wuhan), Anbite Bo (2019 undergraduate students, School of Computer Science, Wuhan University, Wuhan). In addition, we also thank Duantengchuan Li for his overall planning of the entire GooglePlay dataset project. His continuous efforts are the key to the proper implementation of the whole project.

References

Alrubaye, H., Mkaouer, M. W., Khokhlov, I., Reznik, L., Ouni, A., & Mcgoff, J. (2020). Learning to recommend third-party library migration opportunities at the API level. *Applied Soft Computing*, *90*, Article 106140.

Chen, H., Shi, S., Li, Y., & Zhang, Y. (2021). Neural collaborative reasoning. In *Proceedings of the web conference 2021* (pp. 1516–1527).

Church, K. W. (2017). Word2Vec. *Natural Language Engineering*, *23*(1), 155–162.

He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., & Wang, M. (2020). LightGCN: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval* (pp. 639–648).

He, Q., Li, B., Chen, F., Grundy, J., Xia, X., & Yang, Y. (2022). Diversified third-party library prediction for mobile app development. *IEEE Transactions on Software Engineering*, *48*(1), 150–165.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017). Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web* (pp. 173–182).

He, R., & McAuley, J. (2016). Fusing similarity models with Markov chains for sparse sequential recommendation. In *2016 IEEE 16th international conference on data mining* (pp. 191–200).

Hidasi, B., Karatzoglou, A., Baltrunas, L., & Tikk, D. (2015). Session-based recommendations with recurrent neural networks. CoRR abs/1511.06939.

Huang, Q., Xia, X., Xing, Z., Lo, D., & Wang, X. (2018). API method recommendation without worrying about the task-API knowledge gap. In *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering* (pp. 293–304).

Jin, Y., Zhang, Y., & Zhang, Y. (2023). Neighbor library-aware graph neural network for third party library recommendation. *Tsinghua Science and Technology*, *28*(4), 769–785.

Li, D., Deng, C., Wang, X., Li, Z., Zheng, C., Wang, J., et al. (2024). Joint inter-word and inter-sentence multi-relation modeling for summary-based recommender system. *Information Processing & Management*, *61*(3), Article 103631.

Li, B., He, Q., Chen, F., Xia, X., Li, L., Grundy, J., et al. (2021). Embedding app-library graph for neural third party library recommendation. In *Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering* (pp. 466–477).

Li, D., Shi, F., Wang, X., Zheng, C., Cai, Y., & Li, B. (2024). Multi-perspective knowledge graph completion with global and interaction features. *Information Sciences*, *666*, Article 120438.

Li, M., Wang, W., Wang, P., Wang, S., Wu, D., Liu, J., et al. (2017). LibD: Scalable and precise third-party library detection in android markets. In *2017 IEEE/ACM 39th international conference on software engineering* (pp. 335–346).

Li, D., Xia, T., Wang, J., Shi, F., Zhang, Q., Li, B., et al. (2024). SDFormer: A shallow-to-deep feature interaction for knowledge graph embedding. *Knowledge-Based Systems*, *284*, Article 111253.

Li, Z., Zhang, Q., Zhu, F., Li, D., Zheng, C., & Zhang, Y. (2023). Knowledge graph representation learning with simplifying hierarchical feature propagation. *Information Processing & Management*, *60*(4), Article 103348.

Liang, R., Zhang, Q., Wang, J., & Lu, J. (2024). A hierarchical attention network for cross-domain group recommendation. *IEEE Transactions on Neural Networks and Learning Systems*, *35*(3), 3859–3873.

Lin, K., Li, Y., Chen, S., Li, D., & Wu, X. (2024). Motion planner with fixed-horizon constrained reinforcement learning for complex autonomous driving scenarios. *IEEE Transactions on Intelligent Vehicles*, *9*(1), 1577–1588.

Lin, K., Li, Y., Liu, Q., Li, D., Shi, X., & Chen, S. (2024). Almost surely safe exploration and exploitation for deep reinforcement learning with state safety estimation. *Information Sciences*, *662*, Article 120261.

Liu, H., Fang, S., Zhang, Z., Li, D., Lin, K., & Wang, J. (2022). MFDNet: Collaborative poses perception and matrix Fisher distribution for head pose estimation. *IEEE Transactions on Multimedia*, *24*, 2449–2460.

Liu, Y., Li, D., Wang, J., Li, B., & Hang, B. (2024). MDLR: A multi-task disentangled learning representations for unsupervised time series domain adaptation. *Information Processing & Management*, *61*(3), Article 103638.

Liu, B., Li, D., Wang, J., Wang, Z., Li, B., & Zeng, C. (2024). Integrating user short-term intentions and long-term preferences in heterogeneous hypergraph networks for sequential recommendation. *Information Processing & Management*, *61*(3), Article 103680.

Liu, H., Zheng, C., Li, D., Shen, X., Lin, K., Wang, J., et al. (2022). EDMF: Efficient deep matrix factorization with review feature learning for industrial recommender system. *IEEE Transactions on Industrial Informatics*, *18*(7), 4361–4371.

Liu, H., Zheng, C., Li, D., Zhang, Z., Lin, K., Shen, X., et al. (2022). Multi-perspective social recommendation method with graph representation learning. *Neurocomputing*, *468*, 469–481.

Nguyen, P. T., Di Rocco, J., Di Ruscio, D., & Di Penta, M. (2020). CrossRec: Supporting software developers by recommending third-party libraries. *Journal of Systems and Software*, *161*, Article 110460.

Quadrana, M., Karatzoglou, A., Hidasi, B., & Cremonesi, P. (2017). Personalizing session-based recommendations with hierarchical recurrent neural networks. In *Proceedings of the eleventh ACM conference on recommender systems* (pp. 130–137).

Rendle, S., Freudenthaler, C., Gantner, Z., & Schmidt-Thieme, L. (2009). BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence* (pp. 452–461).

Saied, M. A., Ouni, A., Sahraoui, H., Kula, R. G., Inoue, K., & Lo, D. (2018). Improving reusability of software libraries through usage pattern mining. *Journal of Systems and Software*, *145*, 164–179.

Sun, Z., Liu, Y., Cheng, Z., Yang, C., & Che, P. (2020). Req2Lib: A semantic neural model for software library recommendation. In *2020 IEEE 27th international conference on software analysis, evolution and reengineering* (pp. 542–546).

Thung, F., Lo, D., & Lawall, J. (2013). Automated library recommendation. In *2013 20th working conference on reverse engineering* (pp. 182–191).

Wang, X., He, X., Wang, M., Feng, F., & Chua, T.-S. (2019). Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval* (pp. 165–174).

Wang, J., Zhang, Q., Shi, F., Li, D., Cai, Y., Wang, J., et al. (2023). Knowledge graph embedding model with attention-based high-low level features interaction convolutional network. *Information Processing & Management*, *60*(4), Article 103350.

Wu, Y., DuBois, C., Zheng, A. X., & Ester, M. (2016). Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the ninth ACM international conference on web search and data mining* (pp. 153–162).

Wu, B., He, X., Wu, L., Zhang, X., & Ye, Y. (2023). Graph-augmented co-attention model for socio-sequential recommendation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *53*(7), 4039–4051.

Wu, F., Li, D., Lin, K., & Zhang, H. (2021). Efficient nodes representation learning with residual feature propagation. In *Advances in knowledge discovery and data mining* (pp. 156–167).

Wu, J., Wang, X., Feng, F., He, X., Chen, L., Lian, J., et al. (2021). Self-supervised graph learning for recommendation. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval* (pp. 726–735).

Wu, B., Zhong, L., Yao, L., & Ye, Y. (2022). EAGCN: An efficient adaptive graph convolutional network for item recommendation in social internet of things. *IEEE Internet of Things Journal*, *9*(17), 16386–16401.

Yu, H., Xia, X., Zhao, X., & Qiu, W. (2017). Combining collaborative filtering and topic modeling for more accurate android mobile app library recommendation. In *Proceedings of the 9th Asia-Pacific symposium on internetware*.

Zhan, X., Liu, T., Fan, L., Li, L., Chen, S., Luo, X., et al. (2022). Research on third-party libraries in android apps: A taxonomy and systematic literature review. *IEEE Transactions on Software Engineering*, *48*(10), 4181–4213.

Zhao, W., Wang, S., Wang, X., Li, D., Wang, J., Lai, C., et al. (2024). DADL: Double asymmetric distribution learning for head pose estimation in wisdom museum. *Journal of King Saud University - Computer and Information Sciences*, *36*(1), Article 101869.

Zhao, J.-z., Zhang, X., Gao, C., Li, Z.-d., & Wang, B.-l. (2022). KG2Lib: knowledge-graph-based convolutional network for third-party library recommendation. *Journal of Supercomputing*, *79*(1), 1–26.