RESEARCH-ARTICLE

A Multi-Scale Hypergraph-Based Approach for Third-Party Library Recommendation in Mobile App Development

ABHINAV JAMWAL, Indian Institute of Technology Roorkee, Roorkee, UT, India

SANDEEP SURESH KUMAR, Indian Institute of Technology Roorkee, Roorkee, UT, India

Accepted: 10 August 2025
Revised: 20 June 2025
Received: 16 January 2025

A Multi-Scale Hypergraph-Based Approach for Third-Party Library Recommendation in Mobile App Development

ABHINAV JAMWAL and SANDEEP KUMAR, Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, India

In mobile app development, selecting the right third-party libraries (TPLs) is crucial to enhance functionality, improve code quality, and speed up the development process. However, recommending appropriate TPLs remains challenging due to the complexity of app-library interactions and the need to capture high-order relationships. Existing methods, such as collaborative filtering and graph-based approaches, often fail to adequately address these complexities. To address this challenge, we propose MsRec, a multiscale hypergraph neural network-based approach for TPL recommendation. MsRec uses hypergraphs to model interactions in groups of different sizes, leading to a detailed representation of the relationship between the application and the library. By modeling the strength, category, and functionality of interactions within each category, our framework improves the accuracy and diversity of recommendations. The multiscale hypergraph structure supports fine-grained relational reasoning, making it particularly effective in this context. Extensive experiments on real-world datasets demonstrate that MsRec outperforms current state-of-the-art methods, providing relevant and diverse TPL recommendations. Additionally, our model shows strong performance on various benchmarks, highlighting its ability to effectively handle complex app-library interactions.

CCS Concepts: • **Software and its engineering → Software libraries and repositories**; • **Information systems → Recommender systems**; • **Computing methodologies → Neural networks**; • **Mathematics of computing → Hypergraphs**;

Additional Key Words and Phrases: Third-party library recommendation, mobile application development, multiscale hypergraphs, collaborative filtering, graph-based recommendation, machine learning, recommendation systems

1 INTRODUCTION

The rapid development of mobile applications has emphasized the need for more efficient development practices. Reusing third-party libraries (TPLs) [17] is a key strategy to streamline app development. These libraries provide pre-built functionalities that developers can integrate into their applications, saving time and effort while improving code quality and performance [5]. However, the vast number of available libraries and the diverse requirements of different applications make selecting the most appropriate TPL a complex task. This complexity highlights the need for advanced recommendation systems to guide developers in selecting the most appropriate TPL for their projects.

Our key insight is that recommendations for third-party libraries should not rely solely on direct or hop-based interactions. In practice, apps often exhibit shared usage behaviours at the group level, reflecting ecosystem-level trends that pairwise models or flat graphs may overlook. To better capture these dynamics, MsRec models

Authors' Contact Information: Abhinav Jamwal, IndianInstituteofTechnology,Roorkee; Sandeep Kumar, sandeep.garg@cs.iitr.ac.in, Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, Roorkee, Uttarakhand, India.

ACM 1557-7392/2025/8-ART
https://doi.org/10.1145/3764114

interactions between applications and libraries on multiple scales - from direct dependencies to group-level patterns - enabling more expressive representation in sparse and complex environments.

Existing methods for TPL recommendation, such as collaborative filtering [10], matrix factorization [7], and graph-based techniques [13], have achieved notable success. However, these approaches focus primarily on pairwise interactions [31], often overlooking the group-level dynamics inherent in app-library ecosystems. By restricting their modeling to direct links to the app library, they may miss important higher-order patterns necessary to understand real-world usage behaviors. While effective in small or densely connected datasets, these methods often struggle with sparsity, heterogeneous usage patterns, and semantic subtleties typical of real-world scenarios. Furthermore, they offer limited capacity to address challenges such as popularity bias and data sparsity, which can affect recommendation performance. These limitations highlight the need for a more expressive approach that captures fine-grained and group-level interactions, supporting robust and context-aware recommendations. A multiscale perspective, which spans direct connections to higher-order group behaviors, can provide a more comprehensive representation of interaction patterns and better reflect the diversity present in real-world ecosystems.

Although Hypergraph Neural Networks (HGNNs) [6] can model higher-order relationships, most lack the ability to operate across multiple interaction scales. It is hard for a simple HGNN to model fine-grained and broad-level interactions effectively within the same framework, thus restricting comprehensive understanding in app-library relationships. These models can also suffer from over-smoothing issues where information from different nodes becomes too similar after several layers, leading to a loss of meaningful distinctions among nodes. Thus, simple HGNNs may not deliver the nuanced representations needed in the TPL recommendation.

To address these challenges, we propose MsRec-a multiscale hypergraph neural network-based approach for TPL recommendations. This approach constructs and learns from hypergraphs that capture interactions across multiple scales, thus effectively modeling the complex high-order relationships between apps and libraries. Unlike existing models, it introduces a novel three-component representation comprising Interaction Strength, the extent of influence one entity exerts on another, the Interaction nature of the interaction, such as ”dependency” or ”compatibility”, and the Per-Category Function role describing the relationship within each category. These features enable the proposed approach to model fine-grained app-library interactions for more precise and diverse recommendations.

It is designed to capture group-level behaviours and aggregate interaction patterns on multiple scales without relying on explicit supervision. It is compatible with standard recommendation pipelines and has been evaluated on benchmark datasets. Our approach constructs three levels of hypergraph interactions: node level, capturing direct app-library links; hyperedge level, modeling interactions among sets of apps and libraries; and group level, grouping entities to reflect collective usage behaviour. This multiscale design enables the model to incorporate both detailed interaction signals and broader ecosystem trends.

We approach the TPL recommendation problem from a new perspective. In an app-library hypergraph, we represent mobile apps, TPLs, and their interactions as nodes and hyperedges. This hypergraph defines low-order app-library interaction information by the direct connections between nodes. In contrast, high-order interaction information can be extracted from the hypergraph to improve TPL recommendations. For example, consider apps A, B, and C. In the hypergraph, paths between C and A through B can identify more similar apps and TPLs. This high-order interaction information is gathered from nodes several hops away from A, like C, to formulate TPL recommendations for A.

By jointly leveraging low-order and high-order interaction patterns, MsRec generates context-aware recommendations that reflect both direct dependencies and shared behavioral trends within the app-library ecosystem. To evaluate its effectiveness, this research investigates the following research questions (RQ).

- **RQ1:** How does MsRec perform compare to existing TPL recommendation methods?

- **RQ2:** How does the depth of MsRec's architecture (in terms of the number of layers used for embedding propagation) affect its recommendation performance?
- **RQ3:** How does the dimensionality of the latent space (d) influence the ability of MsRec to learn meaningful interactions between apps and TPL?
- **RQ4:** How to do different similarity thresholds in the multiscale hypergraph construction phase influence MsRec's recommendations?

We have made several key contributions through this research to address these research questions and provide a comprehensive solution to the TPL recommendation problem.

- We propose MsRec, a multiscale hypergraph neural network-based approach for TPL recommendation, which models app-library interactions as a multiscale hypergraph, capturing low-order and high-order interactions. This framework effectively captures group-wise interactions of varying sizes, providing a robust and flexible solution for the TPL recommendation.
- To enhance the representation of group-wise interactions, we use Gaussian Mixture Models (GMM) for clustering and employ hinge loss to optimize the learning process, improving the model's ability to handle complex relationships.
- We train and evaluate the model on a large-scale public dataset containing 31,421 mobile apps, 727 distinct TPLs, and 530,198 app library usage records. Through extensive experiments, we demonstrate the ability of MsRec to provide accurate, diverse, and interpretable recommendations.

The remainder of this paper is structured as follows. Section 2 discusses related work. Section 3 motivates our study. Section 4 introduces the MsRec methodology. Section 5 presents the experimental results obtained using the proposed approach. Section 6 provides a statistical analysis of the results. Finally, Section 8 concludes the paper and outlines future directions.

2 RELATED WORK

The recommendation of third-party libraries (TPLs) for mobile applications and software projects has gained significant attention due to the growing demand for efficient software development. Various approaches have been proposed to address this problem, ranging from collaborative filtering and matrix factorization methods to graph-based and hypergraph-based approaches.

POP [25] is one of the most basic baseline approaches, which always recommends the most popular TPLs that the testing app has not used. Although simple and widely used for evaluating recommendation methods, it lacks the ability to capture complex interactions beyond mere popularity, making it insufficient for advanced recommendation tasks. LibRec [26], which combines association rule mining and collaborative filtering (CF), has been a standard method for recommending libraries to traditional Java projects. Using historical interactions, LibRec [26] improves prediction performance and partially addresses data sparsity issues. However, its reliance on pairwise relationships limits its ability to uncover higher-order app-library interactions.

CrossRec [17] is a collaborative filtering-based approach designed to recommend TPLs for open-source projects. Existing studies [13] indicate that CrossRec, like other collaborative filtering-based methods, is based primarily on low-order interactions extracted from app library usage records. This limits its ability to provide highly accurate recommendations compared to more advanced approaches that capture higher-order relationships. Other approaches, such as LibFinder [19] and LibCUP [23], focus on TPL discovery for software maintenance and adaptation. Although these methods aid in understanding library usage patterns, they also remain constrained to low-order interactions.

LibSeek [7] aims to improve the diversity and precision of the TPL recommendation for the development of mobile applications. Using matrix factorization with personalized weighting and neighborhood information, it diversifies the prediction results. By incorporating explicit and implicit information and neutralizing popularity

bias through an adaptive weighting mechanism, LibSeek enhances both accuracy and diversity in predictions. Despite these improvements, LibSeek relies on matrix factorization, which struggles to handle sparse user-item matrices effectively in highly dynamic scenarios.

GRec [13], a graph neural network (GNN)-based approach, models interaction between the app and the library by building an app-library graph [9]. It captures both low-order and high-order relationships, enabling more accurate and diverse TPL recommendations. However, GNN-based models like GRec face challenges such as over-smoothing, where node features become overly similar after several layers of propagation, resulting in the loss of meaningful distinctions. Techniques like DropEdge [21] partially mitigate this problem but remain limited in highly dense or sparse datasets.

PyRec [14] builds on the graph-based framework introduced by GRec, but incorporates contextual information into the recommendation process. Unlike GRec, which is specifically designed for app-library recommendation, PyRec focuses on project-library recommendation and embeds Python projects, TPLs, and contextual information into a knowledge graph. Using both interaction data and contextual records, PyRec improves the accuracy and diversity of the recommendation. This makes PyRec particularly effective for datasets with rich contextual relationships, though its reliance on context-dependent information may limit its generalizability to scenarios where such data are unavailable.

Hypergraph-based approaches have emerged as promising solutions for modeling high-order interactions in recommendation systems. Hypergraph Neural Networks (HGNNs) [6], introduced by Feng et al., create hypergraphs to capture relationships that involve multiple nodes simultaneously, providing more nuanced recommendations. Hypergraph Collaborative Filtering (HCCF) [29], a self-learning hypergraph model, extends this concept by encoding local dependencies and global collaborative relationships to improve recommendation precision and mitigate oversmoothing problems. Although hypergraph-based methods address several limitations of graph-based approaches, many rely on predefined hypergraph structures, which can limit their adaptability to diverse datasets.

Building upon these advancements, our proposed MsRec introduces a novel Multi-Scale Hyper Graph Neural Network-based approach for TPL recommendation. It fundamentally differs from GRec [13] and PyRec [14] by leveraging a multiscale hypergraph framework rather than standard graph neural networks. Unlike GRec, which models pairwise app-library relationships, MsRec captures both low-order and high-order interactions across multiple group sizes, providing a more comprehensive understanding of app-library relationships. In contrast to PyRec, which relies on contextual information, it is designed to be effective even in scenarios where such information is unavailable, making it more versatile for diverse datasets. Moreover, MsRec uses a data-driven hypergraph structure that dynamically adapts to the data, allowing it to effectively model complex app-library interactions.

Unlike existing hypergraph-based methods, such as HGNNs [6] and HCCF [29], the proposed approach introduces a multiscale approach that captures interactions at three levels: direct node-level interactions, hyperedge-level interactions, and group-level interactions through clustering. This design allows it to address key challenges, such as data sparsity, popularity bias, and recommendation diversity, more effectively than other methods. Empirical validation of these claims is presented in subsequent sections to demonstrate consistent improvements over existing methods. To facilitate a detailed comparison with the state-of-the-art baselines, Table 1 outlines the distinct problems and characteristics of each method. The table highlights how our approach excels in modeling high-order interactions, handling sparse data, and mitigating popularity bias, areas where many existing techniques fall short.

By addressing the limitations of current approaches and leveraging the strengths of hypergraph neural networks, MsRec provides a comprehensive framework for TPL recommendation, significantly enhancing the accuracy and diversity of recommendations while improving the efficiency and quality of software development.

Table 1. Comprehensive Comparison of Problems and Features Addressed by MsRec and State-of-the-Art Methods

Problem/Feature	CrossRec [17]	POP [25]	LibRec [26]	LibSeek [7]	GRec [13]	MsRec
Captures high-order interactions	✗	✗	✗	✗	✓	✓
Handles data sparsity	✓	✗	✓	✓	✓	✓
Improves prediction performance	✗	✗	✓	✓	✓	✓
Handles popularity bias	✓	✗	✗	✓	✓	✓
Models group-wise interactions	✗	✗	✗	✗	✗	✓
Handles varying interaction intensities	✗	✗	✗	✗	✗	✓
Handles varying interaction categories	✗	✗	✗	✗	✗	✓
Data-driven hypergraph topology	✗	✗	✗	✗	✗	✓
Plug-and-play design	✗	✗	✗	✗	✗	✓
Robust relational reasoning	✗	✗	✗	✗	✓	✓
Comprehensive app-library interaction understanding	✗	✗	✗	✗	✓	✓
Accurate and diverse recommendations	✗	✗	✓	✓	✓	✓

3 PROBLEM STATEMENT AND MOTIVATION

3.1 Problem Statement

TPL recommendation is a system that aims to help mobile app developers identify TPLs that can improve the functionality, performance, or development efficiency of their applications. These are based on an analysis of the usage patterns of different applications of TPLs. Suppose in this perspective there is an interaction graph, as represented by the figure 1, consisting of six applications ($A_1, A_2, A_3, A_4, A_5, A_6$) with five associated TPLs (L_1, L_2, L_3, L_4, L_5), appropriately numbered.

Each edge connecting an app to a TPL means that the app uses the corresponding library. For example, the edge between A_1 and L_3 indicates that A_1 employs L_3.

From this interaction graph, the following observations can be made:

- A_1 and A_3 share common TPLs (L_1 and L_2), suggesting that these apps exhibit similar functionality or development preferences.
- A_3 utilizes L_5, which is not currently used by A_1. This observation indicates that L_5 could potentially benefit the development of A_1, making it a strong candidate for recommendation to the developers of A_1.

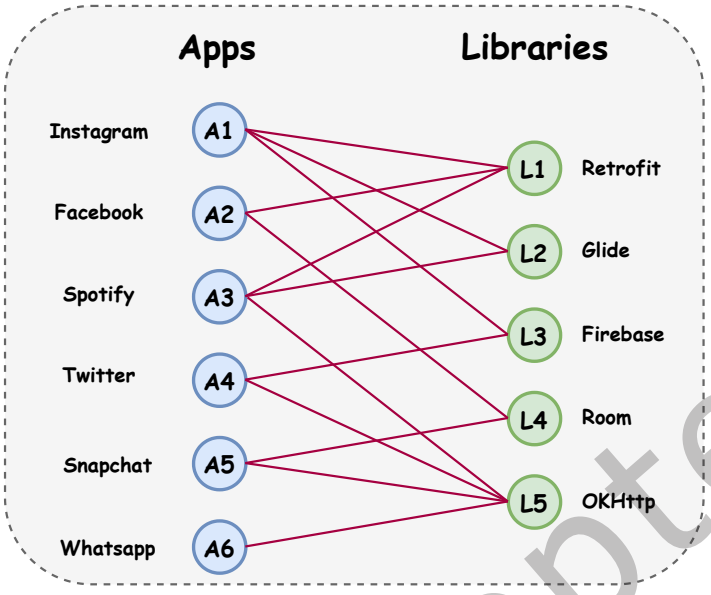

Fig. 1. An Example of Apps using TPLs

Recommending L_5 to A_1, if proven beneficial, can save developers significant time by reducing the effort required to find suitable libraries. This would ultimately improve development efficiency. In this context, MsRec provides accurate, diverse, and effective TPL recommendations, ensuring developers receive valuable suggestions tailored to their needs.

3.2 Motivation

Existing collaborative filtering (CF) techniques mainly rely on low-order connections to collect information from neighbouring nodes. Although effective in capturing direct relationships, these methods do not utilize the high-order information embedded in the app library graphs. Graph neural network (GNN)-based approaches, such as GRec [13], attempt to address this limitation by propagating information across multiple hops to capture higher-order relationships. For example, after two rounds of propagation in the interaction graph, A_1 can collect information from A_3 through shared TPL (L_1, L_2) and from A_4 and A_5 via L_5. Although A_3 provides relevant information due to its strong similarity to A_1, contributions from A_4 and A_5 introduce noise, leading to over-smoothing and reducing recommendation precision.

The key intuition behind our approach is that real-world app-library relationships are not only defined by direct or multi-hop connections but also by shared group-level behaviours. That is, apps often follow collective usage patterns, selecting similar libraries not merely due to pairwise similarity, but as part of a broader ecosystem of functional requirements. For example, groups of apps such as A_1 and A_3, which share the TPL L_1, or A_1 and A_4, which share the TPL L_3, reveal shared behaviours that cannot be fully captured by traditional methods. So, capturing these group-level interactions facilitates an extended understanding of the app-library ecosystem.

Our insight is that modeling these multiscale relationships, low-order, high-order, and group-level, within a unified framework allows us to extract both specific dependencies and generalized behaviour patterns. This is

especially important in sparse data scenarios, where direct connections may be missing, and relying solely on traditional GNN or CF signals would result in poor generalization.

- The nodes A_3 and A_4 are two hops from A_1, but their relevance to the recommendations of A_1 varies. For example, A_3 shares two TPLs (L_1 and L_2) with A_1, while A_4 shares only one TPL (L_3) with A_1, making A_3 more similar to A_1.
- Similarly, L_3 is connected to both A_1 and A_4, indicating its similarity in usage patterns. This insight improves the quality of recommendations.

By encoding these different levels of granularity within a unified hypergraph structure, our model goes beyond traditional GNN or CF methods. It captures the subtle balance between specificity (direct dependencies) and generality (usage trends), uniquely positioning it to outperform saturated baselines in accuracy, diversity, and robustness. This approach recognizes that effective TPL recommendations must reflect the collective intelligence of the app ecosystem, an insight that pairwise or hop-based methods alone cannot fully capture. By grounding our model in this intuition and operationalizing it through multiscale hypergraph learning, we offer a more nuanced and effective solution to the TPL recommendation problem. We now describe the architecture and learning mechanism of MsRec that implements this intuition.

4 PROPOSED APPROACH

MsRec employs multiscale hypergraph neural networks [6] to capture high- and low-order interactions between apps and TPLs. In this way, it can model multiscale interactions by capturing group-level behaviours between applications and TPLs. The notation used in the proposed model is detailed in Table 2. This section starts with a broad overview of the proposed model process and then provides a detailed description of each phase of the methodology.

4.1 Method Overview

To recommend potentially beneficial TPLs for a given application, such as app A_1 in Figure 1—MsRec proceeds through five structured phases, as illustrated in Figure 5. The complete pipeline, including multiscale hypergraph construction, iterative embedding refinement, and top-K recommendation generation, is outlined in Algorithm 1. This algorithm provides a detailed and reproducible framework for capturing interactions at the application, TPL, and group levels. Group-level semantics are modeled using clustering techniques like Gaussian Mixture Models (GMM), enriching the hypergraph with latent relationships not captured at lower levels. To support reproducibility and validation, we have publicly released both the MsRec implementation[1] and the MALib dataset[2], which were used in our experiments.

In the **first phase**, MsRec calculates the similarity between TPLs and apps based on their usage records in the training set. This similarity measure identifies neighbours for each application and TPL, forming the foundation for the hypergraph construction in subsequent phases. By setting thresholds on these similarity scores, it ensures that only meaningful connections are retained, allowing the model to effectively capture app-TPL relationships.

In the **second phase**, MsRec constructs three hypergraphs: G_1, G_2 and G_3. Hypergraph G_1 represents interactions among apps, G_2 represents interactions between TPLs, and G_3 captures group-level interactions between apps and TPLs. Each hypergraph is created by connecting nodes (apps or TPLs) to their neighbors through hyperedges, forming a multiscale hypergraph structure. G_3 further incorporates group-level relationships derived using clustering techniques such as Gaussian Mixture Models (GMM) [20].

In the **third phase**, the nodes representing the TPLs and the applications in the hypergraphs are embedded in a dimension-latent factor space d. These embeddings serve as the foundation for capturing multiscale interactions

[1]MsRec replication package along with dataset at Zenodo: https://doi.org/10.5281/zenodo.15286016
[2]MALib dataset is available at https://github.com/malibdata/MALib-Dataset.git

Table 2. Summary of Notations in MsRec

Notation	Description
G_1, G_2, G_3	Hypergraphs representing node-level, hyperedge-level, and group-level interactions
u, v	Mobile apps in the dataset
i, j	Third-Party Libraries (TPLs) in the dataset
$L(u)$	All the libraries used by app u
$A(i)$	All the apps using library i
H_1, H_2, H_3	Incidence matrices for the three hypergraphs G_1, G_2, and G_3
adj_{u1}, adj_{u2}	Adjacency matrices for app-level interactions
adj_{i1}, adj_{i2}	Adjacency matrices for library-level interactions
$adj_{\text{cat\_user}}, adj_{\text{cat\_item}}$	Group-level adjacency matrices for apps and libraries, respectively
x_u	Latent factor vector of app u
y_i	Latent factor vector of library i
Z	Low-dimensional representation of the interaction matrix using PCA/NMF
$\text{Sim}_{\text{App}}(u, v)$	Similarity between apps u and v based on their TPL usage
$\text{Sim}_{\text{TPL}}(i, j)$	Similarity between libraries i and j based on the apps using them
α_l	Adaptive weight applied to embedding update during training
u_l, v_l	App and TPL embeddings after l-th layer of propagation
λ_u, λ_i	Regularization hyperparameters for app and library embeddings
$\hat{R}_{i,j}$	Predicted score for the probability of TPL i being used by app j
MAP	Mean Average Precision, a metric for ranking the TPLs in recommendations
r_m	Number of TPLs removed from each testing app
K	Number of TPLs recommended for each app
k	Number of apps or TPLs selected as neighbours
α_1, α_2	Thresholds for similarity scores used to create hyperedges in the hypergraph
GMM	Gaussian Mixture Model used for clustering app and TPL interactions
NMF	Non-Negative Matrix Factorization used for reducing dimensions
$p(x)$	Probability density function used in GMM for clustering app-TPL interactions

among nodes, as described in the subsequent phases. The node-level interactions, shown in Figure 2, capture the direct relationships between entities. Each node represents an app, and hyperedges represent shared relationships or interactions between multiple apps.

Hyperedge-level interactions, depicted in Figure 3, involve relationships in which a hyperedge connects multiple libraries. This captures complex usage patterns, such as dependencies or shared functionality, within a hypergraph structure.

Group-level interactions, illustrated in Figure 4, model collective behaviours by clustering apps and TPLs into groups. This clustering, achieved using Gaussian Mixture Models (GMM) [20], captures both fine-grained and holistic patterns in the data. The clusters, represented with dotted lines in the figure, enable a comprehensive understanding of app-library relationships and support more accurate and diverse recommendations.

In the **fourth phase**, it extracts high-order neighbourhood information from the constructed hypergraphs using multiscale aggregation. This process aggregates information from neighbouring nodes through hyperedges, iteratively refining the node embeddings. The resulting embeddings capture rich multiscale relationships between applications and TPLs.

Algorithm 1 MsRec: Multiscale Hypergraph Neural Network for TPL Recommendation

Input: Interaction Matrix M, Similarity Thresholds α_1, α_2, Embedding Dimension d, Number of Layers L, Top K
Output: Top K TPL Recommendations for each App

Initialize app embeddings U_0 and TPL embeddings V_0 randomly; *//Initialize embeddings for each app pair* (A_i, A_j)
foreach *app pair* (A_i, A_j) **do**
  Compute app similarity $\text{Sim}_{\text{App}}(A_i, A_j)$ using Jaccard similarity; *//Calculate similarity for apps*
foreach *TPL pair* (L_i, L_j) **do**
  Compute TPL similarity $\text{Sim}_{\text{TPL}}(L_i, L_j)$ using Jaccard similarity; *//Calculate similarity for TPLs*
Identify neighbors based on similarity thresholds α_1 and α_2; *//Identify neighbors for both apps and TPLs*
Construct app-level and TPL-level hypergraphs G_1 and G_2; *//Build hypergraphs for both levels*
Construct group-level hypergraph G_3 using clustering techniques (e.g., GMM); *//Capture group-level interactions*
Build sparse adjacency matrices A_u, A_i, $A_{\text{cat\_user}}$, and $A_{\text{cat\_item}}$; *//Construct adjacency matrices*
for $l = 0$ *to* L **do**
  Update app embeddings $U^{(l+1)} = \sigma(A_u \cdot U^{(l)})$; *//Update app embeddings using adjacency matrices*
  Update group-level app embeddings $U^{(l+1)} = \sigma(A_{\text{cat\_user}} \cdot U^{(l+1)})$; *//Incorporate group-level interactions*
  Update TPL embeddings $V^{(l+1)} = \sigma(A_i \cdot V^{(l)})$; *//Update TPL embeddings using adjacency matrices*
  Update group-level TPL embeddings $V^{(l+1)} = \sigma(A_{\text{cat\_item}} \cdot V^{(l+1)})$; *//Incorporate group-level interactions*
Aggregate multi-layer embeddings $U_{\text{agg}} = \sum_{l=0}^{L} \alpha_l U^{(l)}$ and $V_{\text{agg}} = \sum_{l=0}^{L} \beta_l V^{(l)}$; *//Aggregate embeddings from all layers*
Normalize embeddings U_{norm} and V_{norm} for each app A_i and TPL L_j; *//Normalize embeddings for stability*
Compute relevance score $\hat{R}_{i,j} = U_{\text{norm}}(i) \cdot V_{\text{norm}}(j)$; *//Compute relevance score for ranking*
Rank TPLs for each app and recommend the top K TPLs; *//Rank and recommend top TPLs*
Update parameters using Adam optimizer with hinge loss; *//Optimize model parameters*

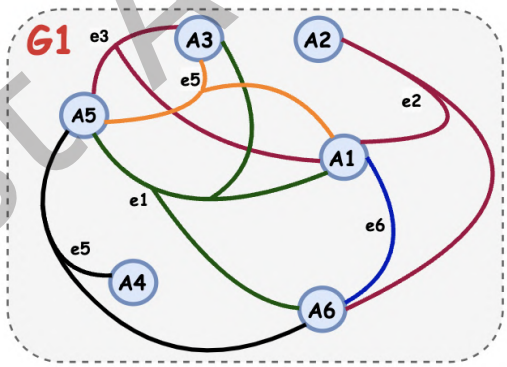

Fig. 2. Node-Level Interactions: Nodes represent apps, hyperedges capture shared interactions

In the **final phase**, MsRec updates these representations by aggregating inspired by graph-convolutional networks (GCN) [28]. Using refined node embeddings, it estimates the likelihood of each TPL being relevant for a given app and generates ranked recommendations. The recommendation process involves calculating similarity scores, ranking TPLs, and selecting the top-k TPLs for each app, excluding those already used by the app.

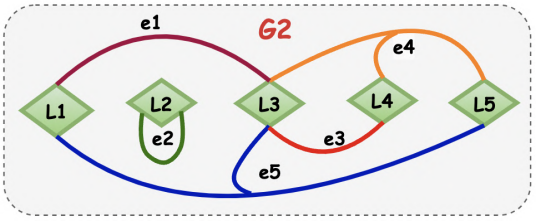

Fig. 3. Hyperedge-Level Interactions: Relationships involving multiple libraries

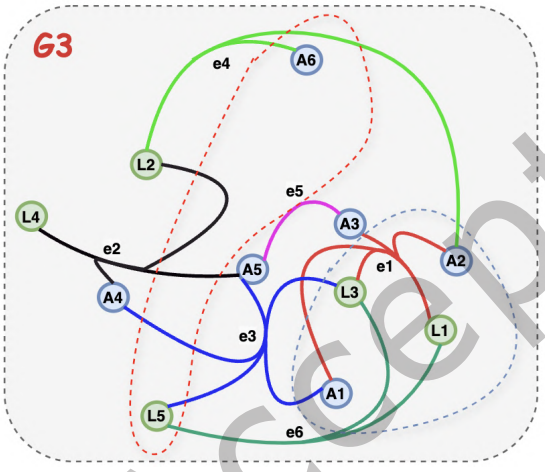

Fig. 4. Group-Level Interactions: Clustering apps and TPLs to model collective behaviours

MsRec optimizes its parameters using the Hinge loss function and the mini-batch Adam optimizer [33], ensuring efficient and robust training. This approach highlights the comprehensive capabilities of MsRec in addressing key challenges in third-party library recommendation. Specifically, it captures both low-order and high-order interactions, handles data sparsity, and mitigates popularity bias. Unlike others, it effectively models group-wise interactions, accounts for varied interaction intensities, and leverages a data-driven hypergraph topology. Its plug-and-play design and relational reasoning further support accurate and interpretable recommendations.

4.2 Phase 1: Similarity Calculation and Neighbour Identification

In the first phase, we determine the similarity between TPL and apps based on their usage records. This similarity is important for creating connections (hyperedges) in later steps. We use the Jaccard similarity [11] to measure the overlap between the sets of TPLs used by different apps and vice versa, as it effectively captures the patterns of co-usage in binary data. Furthermore, existing work [8] has demonstrated Jaccard's robustness and consistent performance in the context of third-party library recommendation.

$$\text{Sim}_{\text{App}}(A_i, A_j) = \frac{|L(A_i) \cap L(A_j)|}{|L(A_i)| + |L(A_j)| - |L(A_i) \cap L(A_j)|} \tag{1}$$

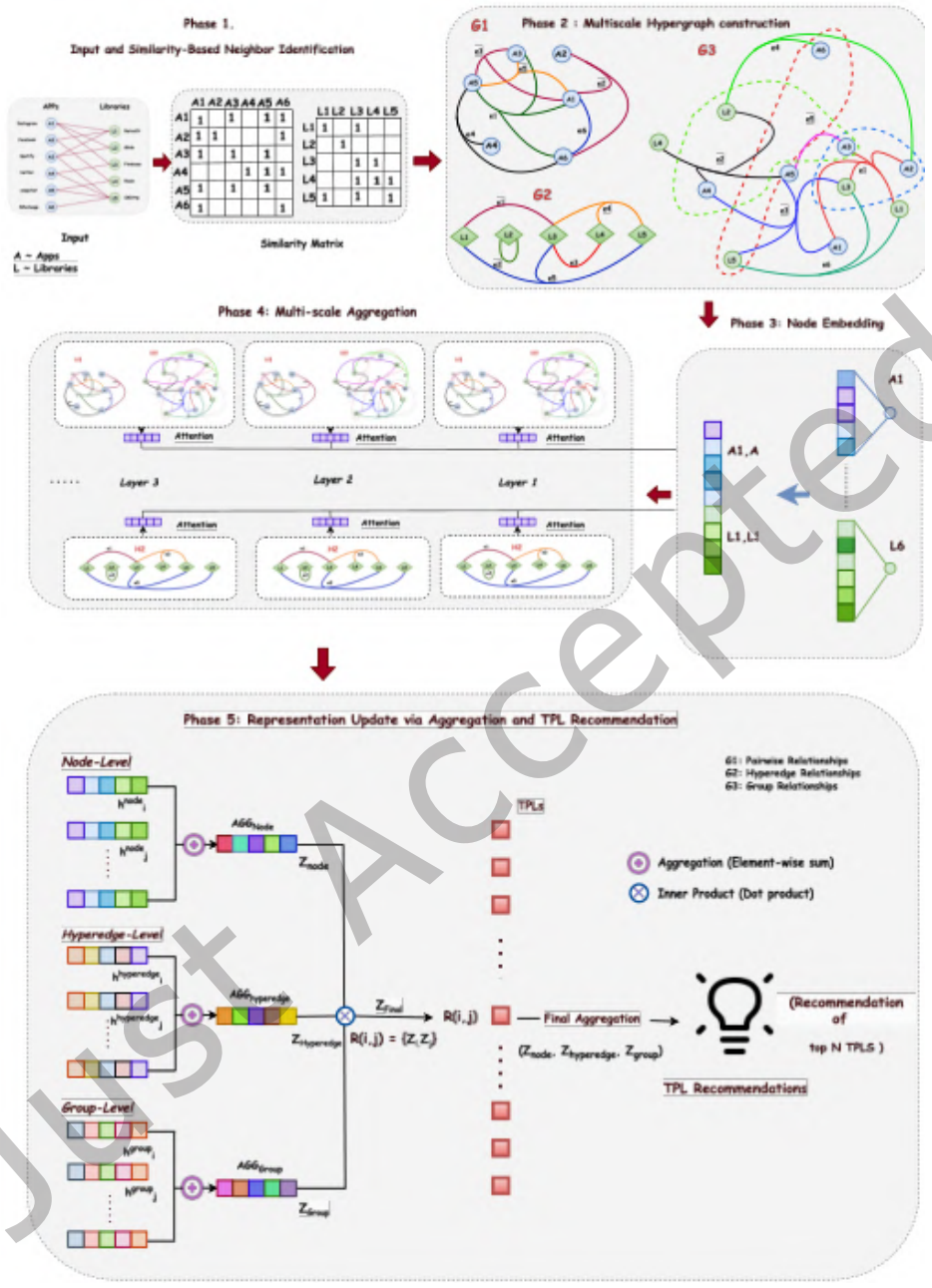

Fig. 5. : General process flow of MsRec

$$\text{Sim}_{\text{TPL}}(L_i, L_j) = \frac{|A(L_i) \cap A(L_j)|}{|A(L_i)| + |A(L_j)| - |A(L_i) \cap A(L_j)|} \tag{2}$$

where $L(A_j)$ and $L(A_i)$ represent sets of TPLs used by applications A_j and A_i, respectively, and $A(L_j)$ and $A(L_i)$ represent the sets of applications that use TPLs L_j and L_i. After calculating these similarities, we identify the most similar apps for each and the most similar TPLs for each TPL. We set thresholds α_1 and α_2 [32] to decide similarity; apps or TPLs with similarity scores above these thresholds are considered neighbours. Details on how these thresholds are chosen and validated are discussed in RQ4 in Section 5.5.

4.3 Phase 2: Multiscale Hypergraph Construction and Group-Level Interaction

In this phase, we construct three hypergraphs: G_1, G_2, and G_3. These hypergraphs are based on the neighbouring nodes identified for libraries and apps in the last step. Hypergraph G_1 consists of only app nodes, while G_2 consists of only TPL nodes. Hypergraph G_3 captures group-level interactions among apps and TPLs, leveraging the results of Non-Negative Matrix Factorization (NMF) [12], Principal Component Analysis (PCA) [1], and Gaussian Mixture Model (GMM) [20] clustering to define relationships that are not apparent at individual or pairwise levels.

We initially place app nodes in the hypergraph $G_1 = (V_1, E_1)$, where V_1 is the set of app nodes and E_1 is the set of hyperedges. The incidence matrix H_1 with dimensions $|V_1| \times |E_1|$ represents the relationships between the nodes of the app and the hyperedges. The weighted incidence matrix H_1 is defined as:

$$H_1(A_i, e_j) = \begin{cases} \text{Sim}_{\text{App}}(A_i, A_j) & \text{if } \text{Sim}_{\text{App}}(A_i, A_j) \geq \alpha_1, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

where $A_i \in V_1$ is the i-th app node in G_1 and $e_j \in E_1$ represents the j-th hyperedge that connects A_j to A_i, given $A_i \in N(A_j)$. $\text{Sim}_{\text{App}}(A_i, A_j)$ represents the calculated similarity between A_i and A_j, which serves as the weight of vertex A_j on hyperedge e_j. α_1 is the similarity threshold above which nodes are considered connected.

Similarly, for hypergraph $G_2 = (V_2, E_2)$, where V_2 is the set of TPL nodes and E_2 is the set of hyperedges. The incidence matrix H_2 with dimensions $|V_2| \times |E_2|$ represents the relationships between TPL nodes and hyperedges. The weighted incidence matrix H_2 is defined as:

$$H_2(L_i, e_j) = \begin{cases} \text{Sim}_{\text{TPL}}(L_i, L_j) & \text{if } \text{Sim}_{\text{TPL}}(L_i, L_j) \geq \alpha_2, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Where $L_i \in V_2$ denotes the i-th TPL node in G_2 and $e_j \in E_2$ represents the j-th hyperedge connecting L_i and L_j, given $L_i \in N(L_j)$. The similarity $\text{Sim}_{\text{TPL}}(L_i, L_j)$ between L_i and L_j is assigned as the weight of the vertex L_i on the hyperedge e_j. α_2 is the similarity threshold that determines when the nodes are considered connected.

Hypergraph G_3 aims to capture group-level interactions between applications and TPLs, which are not apparent at the individual or pair level. This hypergraph is constructed using features extracted through NMF [12] and PCA [1], followed by clustering through GMM [20]. For NMF, the interaction matrix V is decomposed into two nonnegative matrices W and H:

$$V \approx WH \tag{5}$$

where V is the interaction matrix that represents the interactions between applications and TPL, W represents the latent factors of applications and H represents the latent factors of TPLs. For PCA, the interaction matrix X is projected onto a lower-dimensional space using the transformation matrix W:

$$Z = XW \tag{6}$$

where X is the interaction matrix representing the interactions between the applications and the TPL, W is the transformation matrix for PCA and Z is the lower-dimensional representation of the interaction data.

Using these reduced-dimensional features, we apply Gaussian Mixture Models (GMM) to identify clusters of similar apps and TPLs. GMM models the data as a mixture of Gaussian distributions to identify clusters.

$$P(x) = \sum_{k=1}^{K} \pi_k N(x \mid \mu_k, \Sigma_k) \tag{7}$$

where $P(x)$ is the probability density function of the data point x, π_k is the weight of the k-th Gaussian component, μ_k is the mean vector of the k-th Gaussian component, Σ_k is the covariance matrix of the k -th Gaussian component and K is the number of Gaussian components (clusters). The Expectation-Maximization (EM) algorithm is used to estimate these parameters.

The adjacency matrices for group-level interactions are then constructed based on the clustering results. For the TPL nodes, the adjacency matrix $\text{adj}_{\text{cat\_item}}$ is defined as

$$\text{adj}_{\text{cat\_item}} = \begin{cases} 1 & \text{if label}(i) = \text{label}(j), \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

where $\text{label}(i)$ is the cluster label of node i obtained from GMM clustering. For app nodes, the adjacency matrix $\text{adj}_{\text{cat\_user}}$ is similarly defined:

$$\text{adj}_{\text{cat\_user}} = \begin{cases} 1 & \text{if label}(i) = \text{label}(j), \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

These adjacency matrices are converted to sparse tensor representations for efficient operations with PyTorch, ensuring that group-level interactions are effectively captured and used in the subsequent phases of the model. By incorporating this multiscale approach, MsRec leverages the structural diversity of the data to better represent the relationship between the app and the library in the learning process.

4.4 Phase 3: Node Embedding

In this phase, our aim is to represent both applications and TPLs as vectors in a latent space, capturing their essential characteristics for subsequent similarity calculations and predictions. This representation is crucial for the model to learn and make accurate recommendations. Here, each app and TPL is embedded in a latent factor space of dimension d, where the relationships between apps and TPLs can be effectively modeled.

We define the embeddings for apps (users) and TPLs (items). Let n_{apps} be the total number of applications, n_{tpls} be the total number of TPLs, and d be the dimension of the latent space. We initialize two embedding matrices: $U \in \mathbb{R}^{n_{\text{apps}} \times d}$ for app embeddings and $I \in \mathbb{R}^{n_{\text{tpls}} \times d}$. These initial embeddings are learned through a trainable embedding layer, typically initialized using a method such as Xavier initialization [4], ensuring that the variance of the weights is maintained. This initialization is expressed as:

$$U_0 = \text{Initialize}(n_{\text{apps}}, d), \quad I_0 = \text{Initialize}(n_{\text{tpls}}, d)$$

Specifically, let u_i be the d-dimensional embedding of the i-th app, and v_j be the d-dimensional embedding of the j-th TPL.

These latent factor representations, initially randomized, aim to capture the characteristics of TPLs or apps. Such features might encompass performance, data storage methods, functionality, security attributes, performance,

compatibility, universality, etc. The inner product of the latent feature vectors for library L_j and app A_i can serve as an indicator of the potential usage of library L_j in app A_i. The computational formula for this likelihood is as follows.

$$\hat{R}_{i,j} = \vec{A}_i \cdot \vec{L}_j \tag{10}$$

where the dot $(\cdot)$ denotes the inner product operator. A higher value of $\hat{R}_{i,j}$ shows a higher probability of TPL L_j being used in app A_i.

To incorporate the high-order neighbourhood data derived from hypergraphs G_1, G_2, and G_3, we iteratively update the embeddings using the adjacency matrices constructed in phase 2. Let adj_{u1} and adj_{u2} be the adjacency matrices for apps, adj_{i1} and adj_{i2} be the adjacency matrices for TPLs, and adj_{item} and adj_{user} be the group-level adjacency matrices for TPLs and apps, respectively.

To prevent overfitting and ensure the generalizability of the embeddings, a regularization term is incorporated into the loss function during training:

$$\text{Reg}(U, I) = \lambda_u \|U\|_F^2 + \lambda_i \|I\|_F^2 \tag{11}$$

where λ_u and λ_i are regularization hyperparameters, and $\|\cdot\|_F$ denotes the Frobenius norm. This regularization term is combined with the main loss function, hinge loss [24], to form the overall objective function:

$$\mathcal{L} = \text{Hinge Loss} + \text{Reg}(U, I) \tag{12}$$

By incorporating these multiscale interactions and high-order neighborhood information, the final embeddings for both apps and TPLs capture a comprehensive view of their relationships and interactions. These representations are intended to support accurate and diverse TPL recommendations.

4.5 Phase 4: Multi-scale Aggregation

In this phase, we use constructed hypergraphs to derive high-level neighborhood details for each node, encompassing both applications and TPLs. This phase aggregates information from neighboring nodes through the hyperedges, iteratively refining the node embeddings.

For app embeddings, the process begins with the initial embeddings u_0 from phase 3. In each layer l of the neural network, the embeddings are updated by aggregating information from the hyperedges. The iterative update process involves two main steps: generating hyperedge representations from node embeddings and then updating node embeddings based on these hyperedge representations.

First, we propagate the embeddings of the app through the adjacency matrices adj_{u1} and adj_{u2} to capture first- and second-scale interactions:

$$t_u^{(l)} = \text{adj}_{u1} \cdot u_{l-1} \tag{13}$$

$$t_u^{(l)} = \text{adj}_{u2} \cdot t_u^{(l)} \tag{14}$$

Next, we incorporate the third scale of interaction by propagating through the group-level adjacency matrix $\text{adj}_{\text{cat\_user}}$:

$$t_{\text{cat\_u}}^{(l)} = \text{adj}_{\text{cat\_user}} \cdot t_u^{(l)} \tag{15}$$

We then apply dropout to prevent overfitting and improve generalization:

$$t_{\text{cat\_u}}^{(l)} = \text{Dropout}(t_{\text{cat\_u}}^{(l)}) \tag{16}$$

The embeddings are updated using adaptive weights α_l:

$$u_l = \alpha_l \cdot t^{(l)}_{\text{cat\_u}} + (1 - \alpha_l) \cdot t^{(l)}_u \tag{17}$$

This process is repeated for each layer l, where l ranges from 1 to L (the total number of layers). The final app embeddings u_{final} are obtained by averaging the embeddings from all layers:

$$u_{\text{final}} = \frac{1}{L} \sum_{l=0}^{L} u_l \tag{18}$$

A similar process is followed for TPL embeddings. The initial embeddings v_0 are updated in each layer l propagating through the adjacency matrices adj_{i1} and adj_{i2}:

$$t^{(l)}_i = \text{adj}_{i1} \cdot v_{l-1} \tag{19}$$

$$t^{(l)}_i = \text{adj}_{i2} \cdot t^{(l)}_i \tag{20}$$

The embeddings are further refined by propagating through the group-level adjacency matrix $\text{adj}_{\text{cat\_item}}$:

$$t^{(l)}_{\text{cat\_i}} = \text{adj}_{\text{cat\_item}} \cdot t^{(l)}_i \tag{21}$$

Dropout is applied to prevent over-fitting:

$$t^{(l)}_{\text{cat\_i}} = \text{Dropout}(t^{(l)}_{\text{cat\_i}}) \tag{22}$$

The combined embeddings are then updated using adaptive weights α_l:

$$v_l = \alpha_l \cdot t^{(l)}_{\text{cat\_i}} + (1 - \alpha_l) \cdot t^{(l)}_i \tag{23}$$

This process is repeated through each layer l, resulting in a final collection of embeddings $\{v_0, v_1, \ldots, v_L\}$. The ultimate TPL embeddings v_{final} are determined by averaging the embeddings of all layers:

$$v_{\text{final}} = \frac{1}{L+1} \sum_{l=0}^{L} v_l \tag{24}$$

By integrating multiscale interactions and capturing high-order neighbourhood information, the resulting embeddings for both apps and TPLs offer a detailed view of their relationships and interactions. This representation is designed to leverage the rich multiscale connections within the dataset for downstream recommendation.

4.6 Phase 5: Representation Update via Aggregation and TPL Recommendation

In this final phase, we integrate multiscale embedding aggregation and third-party library (TPL) recommendations to generate precise and relevant suggestions for app developers.

The representations of nodes, derived from various layers in the embedding process, are aggregated to form comprehensive embeddings for both apps and TPLs. The aggregation leverages the strengths of multiscale embeddings, capturing low-order, high-order, and group-level interactions among nodes. For applications, the embeddings of each layer are denoted as $\{u_0, u_1, \ldots, u_L\}$, and for TPLs, they are $\{v_0, v_1, \ldots, v_L\}$.

Inspired by Graph Convolutional Networks (GCN) [30], we employ a weighted sum aggregation method to combine embeddings from different layers. The aggregated representation for apps, u_{agg}, is computed as:

$$u_{\text{agg}} = \sum_{l=0}^{L} \alpha_l u_l \tag{25}$$

Similarly, the aggregated representation for TPLs, v_{agg}, is:

$$v_{\text{agg}} = \sum_{l=0}^{L} \alpha_l v_l \tag{26}$$

In both equations, $\alpha_l = \frac{1}{L+1}$ ensures equal contribution from all layers. These aggregated embeddings encapsulate rich multi-scale information, making them robust and comprehensive for recommendation tasks.

With the aggregated representations finalized, the TPL Recommendation process begins. Using aggregated embeddings u_{agg} for applications and v_{agg} for TPLs, we estimate the likelihood that each TPL will be integrated into a given application. The likelihood is determined by the inner product of the app's and TPL's aggregated embeddings:

$$\hat{R}_{i,j} = u_{\text{agg}} \cdot v_{\text{agg}} \tag{27}$$

Here, $\hat{R}_{i,j}$ represents the predicted score for TPL L_j being used in the app A_i. A higher score indicates a higher likelihood of recommendation.

The recommendation process involves the following steps:

(1) **Score Calculation:** Compute the inner product scores $\hat{R}_{i,j}$ for all TPLs L_j for the target app A_i.
(2) **Ranking TPLs:** Rank the TPLs according to their scores in descending order.
(3) **Exclude Used TPLs:** Remove any TPLs already utilized by the target app A_i, ensuring new and potentially valuable suggestions.
(4) **Select Top-k TPLs:** Choose the highest ranked TPLs in the list to serve as the final recommendations, where k represents the desired number of recommendations.

The final recommended list of TPLs $\{L_{j1}, L_{j2}, \ldots, L_{jk}\}$ is then provided to the developers of the target app A_i. These recommendations are generated based on the most relevant and useful libraries identified through learned representations.

By consolidating the aggregation and recommendation processes, this final phase enables MsRec to utilize nuanced embeddings of both apps and TPLs, capturing intricate relationships and interactions within the hypergraph. This design aims to support precise and diverse TPL suggestions without assuming the effectiveness of the outcome at this stage.

5 EXPERIMENTAL EVALUATION

This section outlines our experiment settings and assesses MsRec's performance through four research questions. Each research question evaluates a specific aspect of MsRec's capability and performance.

5.1 Experimental Setup

The evaluation is structured around the following research questions.

(1) How does MsRec perform compare to existing TPL recommendation methods?
(2) How does the depth of MsRec's architecture (in terms of the number of layers used for embedding propagation) affect its recommendation performance?
(3) How does the dimensionality of the latent space (d) influence the ability of MsRec to learn meaningful interactions between the apps and TPL?

(4) How to do different similarity thresholds in the multiscale hypergraph construction phase influence MsRec's recommendations?

MsRec was implemented using a state-of-the-art GNN framework [13]. The experiments were performed on a computer with a 14th generation Intel® CoreTM i7 processor, 16 GB DDR5-4800 RAM, an NVIDIA® Quadro® T1000 GPU and a 512 GB SSD. The system was running on Windows 11 Pro with PyTorch 1.3.1, NumPy 1.18.1, SciPy 1.3.2, and Scikit-learn 0.21.3.

The MALib[2] dataset used in our experiments was introduced by He et al. [7]. It contains 61,722 Android apps collected from the Google Play Store via the AndroZoo [2] repository, with third-party libraries (TPLs) extracted using LibRadar [16]. To address LibRadar's limitations, such as mistakenly identifying subfolders of a single TPL as different libraries, the dataset was manually refined by aligning library identifiers with repositories from Maven[3] and GitHub[4], resulting in 827 distinct TPLs and 725,502 app-library usage records. The dataset retained only those apps that used 10 or more TPLs, producing a benchmark of 31,432 applications and 752 distinct libraries, with a total of 537,011 app-library interactions. This curated data set has been widely used to evaluate TPL recommendation approaches.

To mimic real-world scenarios, we used cross-validation and varied parameters r_m and K. The parameter r_m specifies the number of TPLs removed from each test application, and K denotes the number of TPLs recommended for each application. We tested with $r_m \in \{1, 3, 5\}$ and $K \in \{5, 10\}$. In each experiment, r_m TPLs were randomly removed from each of the 31,432 testing apps, and MsRec was used to recommend a list of K TPLs for each application. We evaluated the effectiveness of MsRec by checking whether removed TPLs had been recommended. For each experiment, 31,432 recommendation lists were generated, and we ran each setting 50 times, reporting average results.

To evaluate MsRec, we use a set of metrics that are higher when performance is better, and we adopt the same implementations as in existing research papers to ensure consistency and avoid reimplementation-related validity threats. MP (mean precision) [18] calculates the proportion of relevant recommended TPLs on all lists, while MR (mean recall) [26] is responsible for calculating the number of correctly recommended TPLs from those that have been removed. In addition, the mean F1 score (MF) [27] is a harmonic mean between precision and recall in each list. Furthermore, Mean Average Precision (MAP) [15] measures the ability of MsRec to classify removed TPLs within the recommendation lists, while Coverage (COV) [13] measures the diversity of the recommendations by calculating the distinct TPLs recommended within all lists with respect to all TPLs within the dataset. These metrics allow for an assessment of both the precision and diversity of the MsRec recommendations.

5.2 RQ1: How does MsRec compare to existing TPL recommendation methods?

To evaluate the effectiveness of MsRec, we compare it with five other state-of-the-art TPL recommendation methods. These methods were selected because they represent various approaches, including popularity-based, collaborative filtering, cross-domain, diversification, and graph-based methods. By comparing MsRec to these methods, we can comprehensively assess its performance in terms of recommendation precision, diversity, and robustness.

- **CrossRec** [17]: A proposed CF-based technique to recommend TPLs for open-source projects. However, it suffers from popularity bias and fails to capture high-order relationships, leading to less diversity in recommendations.
- **POP** [25]: This baseline approach always recommends the most popular TPLs not used by the testing app. Although effective for commonly used libraries, it lacks personalization and does not adapt to unique project needs.

[3]https://mvnrepository.com/
[4]https://github.com/

- **LibRec** [26]: Combines association rule mining and collaborative filtering (CF) to make recommendations for traditional Java projects. However, it is limited by its inability to model complex dependencies beyond direct associations.
- **LibSeek** [7]: A state-of-the-art method specifically designed to recommend TPLs for Android applications, using matrix factorization to identify potentially useful TPLs. However, it does not incorporate higher-order dependencies as effectively as MsRec.
- **GRec** [13]: A Graph Neural Network-based recommendation framework that captures high-order interactions through a graph structure. Although effective, it does not utilize a multiscale hypergraph approach like MsRec, limiting its diversity in recommendations.

To conduct a fair comparison, the parameter settings of each competing approach are exactly the same as those in [17], [26], [7], and [13], respectively. In MsRec, we set the number of layers in the hypergraph neural network to 3, each with 128 dimensions. The size of each vector of latent factors was also set to 128. Table 3 compares the average performance of all competing approaches. We can see that MsRec consistently achieves the highest scores in all evaluation metrics, including MP, MR, MF, MAP, and COV.

Table 3. Performance Comparison of Different Approaches

Dataset	Approaches	K = 5					K = 10				
		MP	**MR**	**MF**	**MAP**	**COV**	**MP**	**MR**	**MF**	**MAP**	**COV**
$r_m = 1$	CrossRec [17]	0.0031	0.0155	0.0052	0.0061	0.0472	0.0069	0.0687	0.0125	0.0130	0.0783
	POP [25]	0.0753	0.3765	0.1255	0.2840	0.0316	0.0457	0.4565	0.0831	0.2949	0.0465
	LibRec [26]	0.1267	0.6335	0.2112	0.4622	0.2921	0.0668	0.6682	0.1215	0.4669	0.2990
	LibSeek [7]	0.1348	0.6741	0.2247	0.5236	0.3346	0.0755	0.7553	0.1373	0.5346	0.3960
	GRec [13]	0.1521	0.7607	0.2536	0.6269	0.6948	0.0828	0.8283	0.1506	0.6360	0.7918
	MsRec	**0.1628**	**0.8140**	**0.2713**	**0.6893**	**0.9102**	**0.0866**	**0.8664**	**0.1575**	**0.6963**	**0.9638**
$r_m = 3$	CrossRec [17]	0.0187	0.0312	0.0234	0.0299	0.0896	0.0220	0.0734	0.0339	0.0439	0.1508
	POP [25]	0.2147	0.3579	0.2684	0.5931	0.0322	0.1341	0.4468	0.2063	0.5682	0.0455
	LibRec [26]	0.2789	0.4648	0.3486	0.6883	0.2916	0.1542	0.5142	0.2373	0.6864	0.2936
	LibSeek [7]	0.3710	0.6183	0.4637	0.7280	0.3245	0.2158	0.7193	0.3320	0.6971	0.3907
	GRec [13]	0.4099	0.6915	0.5142	0.7977	0.6849	0.2337	0.7879	0.3602	0.7605	0.7824
	MsRec	**0.4530**	**0.7636**	**0.5681**	**0.8436**	**0.8996**	**0.2487**	**0.8381**	**0.3833**	**0.8136**	**0.9474**
$r_m = 5$	CrossRec [17]	0.0342	0.0342	0.0342	0.0596	0.1701	0.0371	0.0743	0.0495	0.0780	0.2560
	POP [25]	0.3383	0.3383	0.3383	0.7413	0.0316	0.2180	0.4360	0.2907	0.6813	0.0449
	LibRec [26]	0.4400	0.4400	0.4400	0.6922	0.2885	0.2434	0.4868	0.3245	0.6890	0.2992
	LibSeek [7]	0.5291	0.5291	0.5291	0.7896	0.3141	0.3293	0.6587	0.4391	0.7396	0.3796
	GRec [13]	0.5868	0.5945	0.5902	0.8297	0.6571	0.3613	0.7312	0.4834	0.7856	0.7536
	MsRec	**0.6576**	**0.6655**	**0.6611**	**0.8808**	**0.8414**	**0.3864**	**0.7815**	**0.5169**	**0.8350**	**0.9363**

These results, as shown in Table 3, highlight the effectiveness of the proposed approach in multiple evaluation metrics. The significantly higher Mean Precision (MP), Mean Recall (MR), and Mean F1 Score (MF) confirm that it

Table 4. Friedman Test Results

Metric	H-stat	P-value	Alpha
MP	23.97	0.00022	0.05
MR	23.97	0.00022	0.05
MF	23.97	0.00022	0.05
MAP	23.97	0.00022	0.05
COV	23.97	0.00022	0.05

Table 5. Wilcoxon Signed-Rank Test Results with Bonferroni Correction

Comparison	p-value	Effect Size	α (Corrected)	Sig. Diff.
MsRec vs CrossRec [17]	0.007812	0.9405	0.01	Yes
MsRec vs POP [25]	0.007812	0.9405	0.01	Yes
MsRec vs LibRec [26]	0.007812	0.9405	0.01	Yes
MsRec vs LibSeek [7]	0.007812	0.9405	0.01	Yes
MsRec vs GRec [13]	0.007812	0.9405	0.01	Yes

not only retrieves relevant TPLs more accurately than existing approaches, but also maintains a strong balance between precision and recall. This balance ensures that the model avoids overly generic or niche recommendations, making its output both relevant and practical for developers.

The proposed model outperforms POP by 537.21%, 77.91%, 2190.47%, 47.31%, and 11.48% in terms of MP, MR, MF, MAP, and COV, respectively. Similarly, it surpasses LibRec by 48.62%, 49.53%, 48.36%, 31.64%, and 1894.05% in the same metrics. These improvements underscore the performance gains achieved by MsRec over traditional methods.

When $r_m = 1$ and $K = 5$, MsRec achieves its most substantial performance gains. Specifically, it outperforms POP, LibRec [26], CrossRec [17], LibSeek [7], and GRec [13] by 654.49%, 69.25%, 5697.58%, 53.20%, and 12.39% in MP, respectively. Even in a more challenging scenario ($r_m = 5$, $K = 10$), MsRec maintains its advantage with 448.40%, 82.52%, 814.49%, 42.66%, and 10.26% higher MP than the same baselines.

In particular, the method demonstrates a notable architectural advancement over GRec [13], which is the most comparable baseline. Although GRec captures high-order information through pairwise edges within a flat graph, it lacks the expressiveness to model group-wise dependencies. In contrast, MsRec constructs layered hyperedges that span multiscale interactions, enabling richer propagation of contextual signals and more nuanced semantic understanding across the app-library space. This structural advantage contributes to both higher accuracy and improved recommendation diversity.

One key strength is the ability to mitigate popularity bias, as evidenced by its consistently higher COV values compared to LibRec and GRec. Unlike POP [25], which strongly favors frequently used TPLs, MsRec provides a more balanced mix of popular and niche libraries. This diversity makes it particularly valuable in real-world development, where flexibility and innovation are often constrained by the overreliance on widely adopted libraries. Furthermore, it exhibits strong resilience to data sparsity. Its stable performance in increasing values of r_m particularly at $r_m = 5$ demonstrates its robustness when fewer training interactions are available. This is a critical property for practical scenarios where the interaction between the app and the library may be limited or incomplete.

Collectively, these findings emphasize that the proposed solution not only achieves superior performance in terms of accuracy and coverage, but also overcomes key limitations such as popularity bias and data sparsity. Together, these strengths make MsRec a compelling solution for real-world third-party library recommendations.

We used Friedman [3] and Wilcoxon signed rank tests [22] to validate the performance improvements of MsRec over the other methods. The Friedman test was chosen because it is a non-parametric statistical test specifically designed to detect differences in treatments in multiple test attempts, making it suitable for comparing the rankings of multiple algorithms with multiple evaluation metrics. The results of the Friedman test on the five evaluation metrics (MP, MR, MF, MAP, and COV) are summarized in Table 4, which shows strong evidence against the null hypothesis with $p < 0.001$. The Wilcoxon signed rank test was selected to perform pairwise comparisons between MsRec and each baseline method, helping to identify which differences are statistically significant. We also report the effect size of type r, calculated as $r = \frac{|Z|}{\sqrt{N}}$, where Z is the statistic of the standardized test and N is the number of paired observations. The p-values and effect sizes are summarized in Table 5, with all results showing $p < 0.01$ and effect sizes greater than 0.94, indicating large and significant differences. This statistical framework allows global and pairwise validation of performance improvements. The following hypotheses were tested:

- *H0: There is no significant difference between MsRec and the other methods*
- *H1: MsRec performs significantly better than the other methods.*

To ensure statistical validation, we applied a Bonferroni correction to control for multiple pairwise comparisons. Since we compared MsRec against five baseline methods, the original significance threshold of $\alpha = 0.05$ was divided by 5, resulting in a corrected threshold of $\alpha = 0.01$. This correction reduces the likelihood of Type I errors that can arise when multiple hypotheses are tested.

A p-value below this corrected threshold ($p < 0.01$) indicates statistical significance and supports the rejection of the null hypothesis (H0). In contrast, a p-value above 0.01 implies that the observed difference may be due to chance. The size of the effect r, as defined earlier, helps quantify the practical relevance of these differences.

As shown in Table 5, the p-values for all five comparisons are below 0.01 and the corresponding effect sizes are consistently large ($r \approx 0.94$). These results confirm that MsRec significantly outperforms all baseline methods in the evaluated metrics.

Surprisingly, MsRec can greatly diversify its recommendation results while achieving high recommendation accuracy, indicated by its significant advantage in COV over competing approaches. We find that the COV of POP is particularly low in all cases. The reason is that POP always recommends a few of the most popular TPLs that the testing app has not used. Thus, the other less popular TPLs are rarely recommended. This is a critical limitation as recommending only popular TPLs is not beneficial to developers. In contrast, the proposed method diversifies the recommendations by recommending both popular and less popular TPLs, indicated by its highest COV values in all cases.

Unlike LibRec, CrossRec and LibSeek, which use only a small portion of low-order app-library interactions, the proposed approach captures both local and high-order signals distilled from the hypergraph structure. This comprehensive use of interaction information improves its effectiveness, as seen in the highest scores in both precision and diversity of recommendations.

5.3 RQ 2: How does the depth of MsRec's architecture (regarding the number of layers used for embedding propagation) affect its recommendation performance?

To investigate the impact of embedding propagation layers on the performance of MsRec, we conducted a series of experiments by varying the number of layers (n) from 1 to 5. Each experiment was evaluated using multiple metrics: Recall, Precision, MAP, Coverage, and MF Score. We evaluated this for two lengths of the recommendation list: $K = 5$ and $K = 10$. The results, shown in the plots in Figure 6, reveal the following trends.

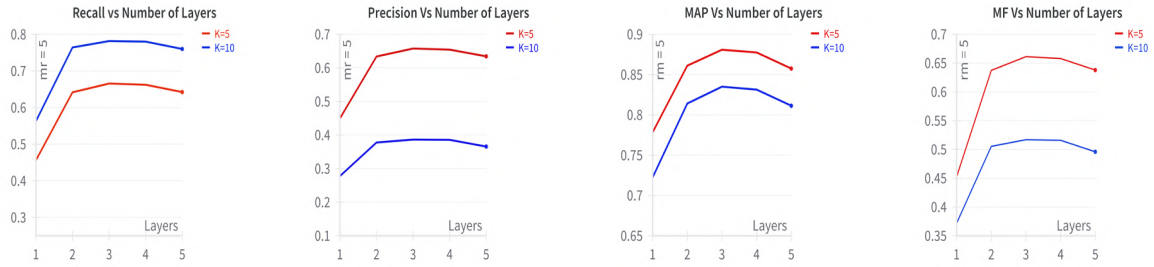

Fig. 6. Impact of the number of embedding propagation layers on MsRec performance metrics for $K = 5$ and $K = 10$. (a) Recall vs Number of Layers. (b) Precision vs Number of Layers. (c) MAP vs Number of Layers. (d) MF vs Number of Layers.

When the number of layers increased from 1 to 3, there was a significant improvement in all metrics. For example, for $K = 5$, the recall increased from 0.456 to 0.666, the precision increased from 0.450 to 0.658, and the MAP increased from 0.778 to 0.881. These improvements suggest that incorporating high-order app-library interactions via additional propagation layers effectively enhances recommendation quality. Similarly, for $K = 10$, metrics such as recall improved from 0.563 to 0.782, precision from 0.278 to 0.386, and MAP from 0.722 to 0.835. The additional layers allowed the model to leverage more intricate relationships within the data, thus improving its performance.

However, the performance metrics decreased slightly when the number of layers increased from 3 to 4 to 5. For example, for $K = 5$, the recall decreased from 0.666 to 0.642, and the MAP decreased from 0.881 to 0.857. This suggests that adding too many layers may introduce noise from distant and less relevant nodes, which can dilute the effectiveness of the learned embeddings. A similar trend was observed for $K = 10$. Recall slightly decreased from 0.782 to 0.760, and MAP from 0.835 to 0.811. The diminishing returns indicate that while high-order information is valuable, overly large n values may incorporate excessive noise, reducing the model's precision and recall. An optimal depth of three layers appears to offer the best trade-off between representational richness and noise avoidance, as evidenced by the maximum metric scores at $n = 3$ for both $K = 5$ and $K = 10$.

5.4 RQ 3: How does the dimensionality of the latent space (d) influence MsRec's ability to learn meaningful interactions between apps and TPL?

MsRec embeds mobile apps and TPLs as latent factors of dimension d to represent their specific features, such as functionality, performance, and compatibility. To study the impact of different values of d on performance, we vary d from 32 to 512. Figure 7 shows the experimental results. When d increases, the model performance in metrics increases. For example, when $K = 5$, $r_m = 5$ and $d = 32$, it achieves 0.647, 0.768, 0.639, 0.380, and 0.911 in MP, MR, MF, MAP and COV, respectively. When d increases to 256, the values become 0.661, 0.774, 0.653, 0.383, and 0.919 in MP, MR, MF, MAP, and COV, respectively, that is, 2. 31%, 0. 81%, 2. 31%, 0. 81%, and 0. 81% higher than when $d = 32$. The reason is that a higher dimensionality of the latent space allows the proposed approach to model more potential latent features that reflect the relationships between apps and TPLs. These richer representations enable the model to assess the potential usefulness of each TPL for a given app with greater precision. Thus, it can recommend TPLs more effectively with a higher d.

Another interesting observation is that when d increases from 32 to 64, performance increases significantly, as indicated by improvements in all metrics. This trend continues as d increases from 64 to 128 and further to 256, showing consistent performance gains. However, the performance increase slows slightly when d is increased from 256 to 512. In practice, a proper value of d can be identified by experiments.

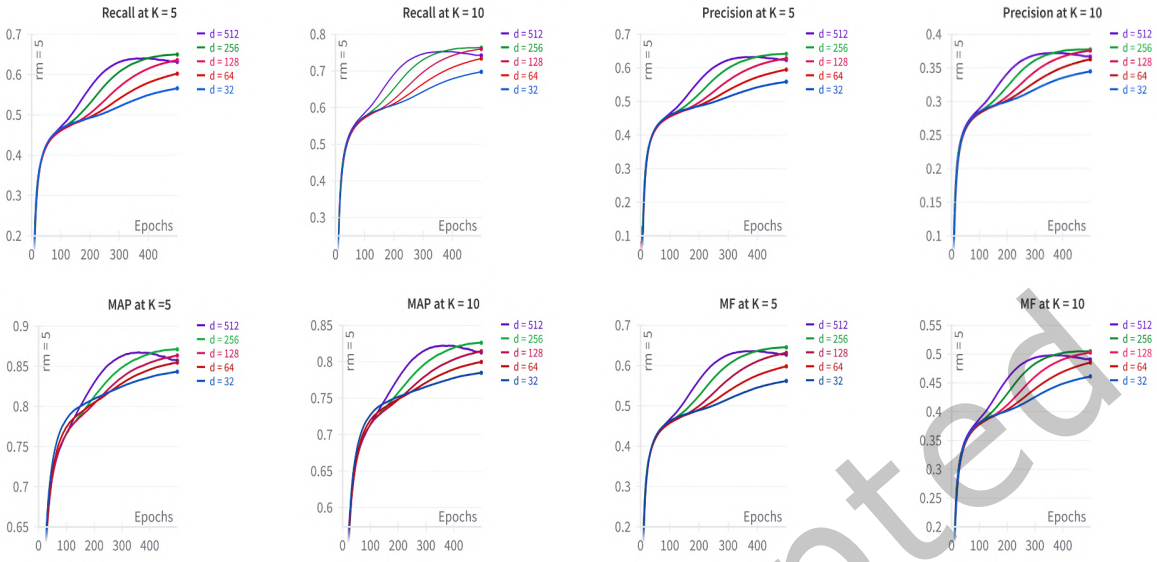

Fig. 7. Impact of dimensionality of latent space on MsRec performance metrics for $K = 5$ and $K = 10$.

5.5 RQ 4: How to do different similarity thresholds in the hypergraph construction phase influence MsRec's recommendations?

The threshold parameter α is a crucial factor affecting the performance of MsRec. It's important to note that the optimal value of α may differ based on the specific dataset. We explore how this threshold affects MsRec performance when applied to the rm=5 dataset. Initial experiments allowed us to determine an appropriate range for α, which lies between 37% and 55%.

We first set the value of α_1 at 0.45 and varied the value of α_2 from 0.37 to 0.54 in increments of 0.04. Similarly, we then set the value of α_2 at 0.45 and varied the value of α_1 from 0.37 to 0.54 in increments of 0.04. The results are illustrated in Figure 8. These experiments found that the optimal values for α_1 and α_2 are 0.45.

In general, the influence of α on the performance of the model is significant. This parameter determines node similarity, affecting the overall performance of the model.

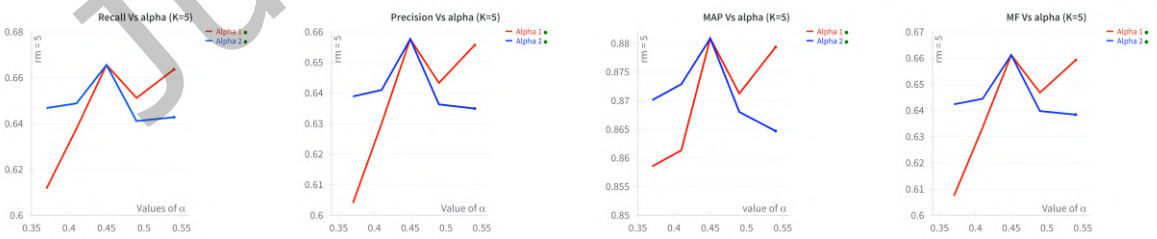

Fig. 8. Impact of the similarity threshold α on MsRec performance metrics for the rm=5 dataset. (a) Recall vs α. (b) Precision vs α. (c) MAP vs α. (d) F1 Score vs α.

Figure 8 shows that performance improves as the value of α increases from 0.37 to 0.45. This indicates that raising the threshold value to an appropriate level can improve performance. However, when the α value exceeds 0.45, the overall performance of the model deteriorates. This shows that too high similarity thresholds can harm the performance of MsRec. This trend highlights the importance of carefully tuning the α parameter to achieve optimal performance.

6 STATISTICAL ANALYSIS

This section presents a series of experiments to evaluate the performance of MsRec in the context of recommendations from the third-party library (TPL) for mobile applications. The experiments are structured around four key areas: ablation testing, robustness testing, scalability testing, and complexity analysis - each capturing different aspects of MsRec's effectiveness and flexibility. All evaluations were carried out using the same dataset and hyperparameter configuration($r_m = 5, K = 5$), as defined in the Experimental Setting section. This setup reflects a more challenging and realistic scenario, where each application has fewer interactions with the library observed. It enables a more meaningful evaluation of the contribution of MsRec in component terms and its ability to perform reliably under moderate sparsity.

6.1 Ablation Study

To evaluate the contribution of each component in MsRec, we conducted an ablation study, systematically removing them and observing the impact on performance. The first is the *full model*, which includes all core components: Hypergraph Level 1, Hypergraph Level 2, and Regularization. The second variant removes *Hypergraph Level 1*, which captures fine-grained group interactions (for example, app-app similarity). The third excludes *Hypergraph Level 2*, responsible for modeling high-order interaction between the app and the library. The fourth disables *Regularization*, which helps prevent overfitting and ensures stable training.

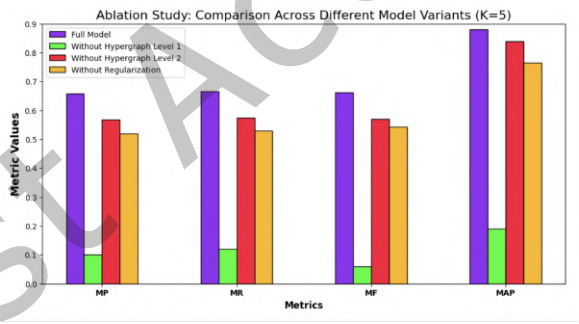

Fig. 9. Ablation Study: Comparison Across Different Model Variants

As shown in Figure 9, the Full Model achieves the best performance in all metrics (MP, MR, MF, MAP), confirming the value of each component. Removing Hypergraph Level 1 results in the largest drop in MP and MF, underlining the importance of modeling local group interactions. Excluding Hypergraph Level 2 moderately degrades performance, indicating that high-order interactions, while helpful, are less critical than group-level semantics. Disabling regularization causes a slight decrease in all metrics, which shows its role in improving generalization.

These results confirm that each component contributes meaningfully, and their combined effect is essential for optimal performance and robustness.

6.2 Robustness Testing

The robustness test evaluates the ability of MsRec to maintain performance in real-world scenarios where the data may be noisy or incomplete. To simulate such conditions, we introduced controlled noise into the interaction matrix by randomly flipping a portion of positive (1) and negative (0) entries. This procedure creates realistic data incompleteness, mimicking errors or missing values that often occur in practical settings.

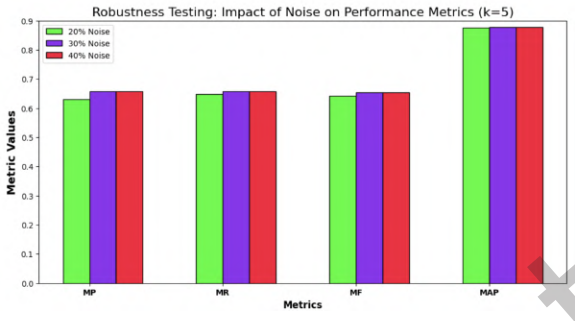

Fig. 10. Robustness Testing: Comparison of MP, MR, MF, and MAP under varying noise levels, showcasing MsRec's resilience to data imperfections.

We tested three increasing noise levels—20%, 30%, and 40% while keeping the configuration fixed at $r_m = 5$, $K = 5$ to maintain consistency with other studies. This setup also ensures that the evaluation occurs under already sparse interaction conditions, further stressing the robustness of the model. Performance was measured using four key metrics: Mean Precision (MP), Mean Recall (MR), Mean F1 Score (MF) and Mean Average Precision (MAP).

As shown in Figure 10, MsRec maintains consistently high performance at all noise levels. Even as the noise increases from 20% to 40%, the variations in MP, MR, and MF are minimal, and the MAP remains above 0.86. This trend underscores MsRec's robustness, as the model effectively mitigates the impact of corrupted interactions without significant performance degradation.

This stability is attributed to MsRec's multiscale hypergraph structure, which captures high-order semantic relationships and provides redundancy in learning patterns. As a result, even when some input signals are distorted or missing, the model continues to generate accurate and reliable TPL recommendations.

6.3 Scalability Study

This experiment investigates how MsRec scales with increasing data set sizes, which is a critical factor for real-world deployment where the volume of data can vary significantly. To ensure comparability and maintain a realistic sparse environment, we evaluated MsRec performance in five user groups - 5,000, 8,000, 15,000, 20,000 and 25,000 apps - under a fixed configuration of $r_m = 5$ and $K = 5$. This setup allows us to isolate and assess the direct influence of the size of the data set on the performance of the recommendation.

As shown in Figure 11, MsRec demonstrates steady and consistent performance improvements in all key metrics (MP, MR, MF, and MAP) as the number of apps increases. This trend confirms MsRec's ability to effectively utilize additional interaction data to refine its recommendations. MAP shows the highest relative gain, indicating enhanced ranking precision with larger datasets.

These results highlight MsRec's scalability and robustness in handling larger datasets, reinforcing its suitability for real-world, large-scale third-party library recommendation tasks.

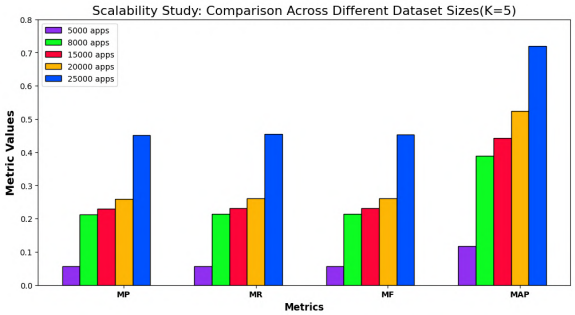

Fig. 11. Scalability Study: Comparison Across Different Dataset Sizes for (K=5)

6.4 Complexity Analysis

This analysis evaluates the practical feasibility of MsRec by comparing its computational cost -measured in training time and memory usage. These metrics are crucial to understanding the benefits of implementation, especially in large-scale environments. We compared the time complexity of MsRec with state-of-the-art methods, including LibSeek and GRec, to highlight the differences in resource requirements. Both training time and memory usage are essential factors for assessing the scalability of these models, particularly when applied to large datasets in a production environment.

Table 6. Computational Complexity Comparison

Model	Training Time (s)	Memory Usage (MB)
MsRec	47,116	3,224
GRec	13,456	778
LibSeek	314.1	12,670

As shown in Table 6, MsRec achieves high accuracy but requires more training time (47,116 seconds), reflecting the computational overhead introduced by its multiscale hypergraph architecture. Despite this, its memory usage remains moderate (3,224 MB), especially compared to LibSeek, which uses 12,670 MB due to MATLAB's matrix operations. GRec strikes a balance with lower memory usage (778 MB) and moderate training time (13,456 s).

These results emphasize the trade-offs in resource efficiency: MsRec is more time-intensive but offers a scalable memory profile, making it suitable for offline batch recommendations where accuracy and robustness are prioritized. In contrast, LibSeek is faster but memory-hungry, and GRec offers a middle ground.

6.5 **Threats to Validity**

Although our experimental evaluation demonstrates the effectiveness of MsRec in recommending third-party libraries (TPLs), we acknowledge several potential threats to validity that may influence the generalizability and reproducibility of our findings.

Internal Validity Internal validity concerns arise from potential biases in the dataset and the experimental setup. - The training and testing datasets were derived from the MALib dataset, which may not fully capture the diversity of real-world applications. Although cross-validation was used to mitigate biases, the inherent characteristics of the data set could still influence the results. - Hyperparameter tuning and model configuration

were conducted on the basis of specific experimental conditions. Suboptimal tuning may have affected the performance of MsRec or competing methods.

External Validity External validity concerns the generalizability of our results to other datasets or domains. - The MALib dataset, while comprehensive, primarily focuses on Android apps and may not reflect the characteristics of other ecosystems, such as iOS or cross-platform applications. - MsRec performance may vary for datasets with different levels of sparsity or various patterns of app-library interaction. Future work should evaluate MsRec on additional datasets to validate its applicability across domains.

Construct Validity Construct validity is referring to whether the evaluation metrics and methods chosen adequately represent the objectives of the study. - Metrics such as MP, MR, MF, MAP, and Coverage are widely used in recommendation systems; however, they may not fully capture real-world utility, such as developer satisfaction or runtime performance of the recommended TPLs. Their relevance in previous studies guided the choice of baseline methods, but a comparison with other state-of-the-art techniques could provide deeper insight.

Reproducibility To ensure reproducibility, we have made the MsRec[1] implementation and the MALib[2] dataset publicly available. To eliminate variations due to hardware, software environments, or library versions, we have containerized our entire experimental setup using Docker. All dependencies are managed through a Dockerfile, ensuring that the experiments can be reliably replicated across different systems and over time. Our Docker configuration and scripts are provided alongside the code repository to facilitate seamless reproduction.

By acknowledging these threats to validity, we aim to encourage further exploration and validation of MsRec in diverse experimental settings and application domains.

7 DISCUSSION

Our experiments show that MsRec significantly outperforms state-of-the-art methods in third-party library (TPL) recommendation for mobile applications. By modeling node-, hyperedge-, and group-level interactions, MsRec captures local, high-order, and collective semantics often overlooked by existing methods. These multiscale patterns allow the model to uncover latent structures and functional similarities, directly contributing to more accurate and diverse recommendations. For example, a developer building a finance or e-commerce app can receive suggestions for libraries, such as analytics, payment gateways, or data encryption modules, based on usage patterns learned from similar applications, thus accelerating development and minimizing trial and error.

Despite these advantages, MsRec currently operates at the library level and does not consider specific versions. Although this simplifies modeling and improves generalizability, it may limit applicability when version compatibility or security is critical. Future work could improve MsRec with version-level granularity for more precise recommendations. The superior performance of MsRec can be attributed to several strengths. This structure not only improves predictive accuracy, but also enables the generation of interpretable recommendations, facilitating integration into real-world development workflows through transparent dependency reasoning. Furthermore, the iterative embedding update process inspired by Graph Convolutional Networks (GCNs) ensures that the final representations are enriched with multiscale interaction data, further enhancing the precision of the recommendation.

Experimental results, including robustness tests, scalability analysis, and ablation studies, further validate the capabilities of MsRec. For example, Figure 10 shows that MsRec maintains high MAP and F1 scores under noise, confirming its robustness to data imperfections. Figure 11 shows that MsRec scales well with an increase in dataset size, supporting its applicability in large-scale scenarios. Moreover, MsRec constructs multiscale hypergraphs using structured interaction data derived from standard dependency records, making it easy to integrate into automated recommendation pipelines.

These evaluations were conducted in the domain of mobile applications, where third-party libraries are widely used to support modular and feature-rich development. The structured nature of dependencies in this setting

makes it well suited for modeling complex interaction patterns. This domain provides a meaningful context to assess the performance and capabilities of MsRec.

Building on insights gained from this evaluation, future work will explore the generalizability of MsRec beyond mobile applications, specifically targeting open source projects such as Python and Java repositories. Although the multiscale hypergraph approach is inherently adaptable, cross-domain applicability may encounter challenges like differing dependency patterns or less structured metadata. Addressing these issues will be critical to further demonstrating MsRec's robustness and flexibility across diverse software ecosystems.

Furthermore, MsRec addresses popularity bias by taking advantage of group-level interactions and varying interaction intensities, resulting in balanced and diverse recommendations. As illustrated in Figure 9 and Table 1, these multiscale mechanisms enable MsRec to consistently outperform baselines even under sparse and imbalanced conditions.

8 CONCLUSION AND FUTURE WORK

In conclusion, this paper presented MsRec, a multiscale hypergraph neural network-based approach to recommend third-party libraries (TPL) in the development of mobile applications. By modeling both low-order and high-order interactions using hypergraphs, MsRec captures complex relationships between applications and libraries. Extensive experiments on real-world datasets demonstrate that MsRec consistently outperforms existing state-of-the-art methods on multiple metrics, including mean precision (MP), mean recall (MR), F1 score (MF), mean average precision (MAP) and coverage (COV), thus improving both recommendation precision and diversity. The scalability study further validates its robustness in handling large datasets, supporting its deployment in large-scale, real-world environments. In general, MsRec addresses key challenges in modeling interaction between applications and libraries and provides a comprehensive solution that can significantly benefit software development and quality assurance.

Future work will focus on refining the similarity metric, evaluating additional data sets that incorporate version-based recommendations, and incorporating user feedback to improve the quality of recommendations. Since the current evaluation is limited to mobile applications, we plan to extend MsRec to cross-domain scenarios such as Python and Java projects to assess its generalizability across diverse software ecosystems. Furthermore, exploring integration in real-world development environments will help demonstrate the practical utility and long-term scalability of MsRec.

ACKNOWLEDGMENT

Authors are thankful to the project funding under the Responsible AI Theme titled *Synthetic Data Generation* launched by IndiaAI - IBD under the Safe and Trusted Pillar of the IndiaAI Mission, MeitY, Government of India.

REFERENCES

[1] Hervé Abdi and Lynne J Williams. 2010. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics* 2, 4 (2010), 433–459.

[2] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2016. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th international conference on mining software repositories*. 468–471.

[3] Jacob Cohen, Patricia Cohen, Stephen G West, and Leona S Aiken. 2013. *Applied multiple regression/correlation analysis for the behavioral sciences*. Routledge.

[4] DeepLearning.AI. 2023. Weight Initialization for Deep Learning: AI Notes. https://www.deeplearning.ai/ai-notes/initialization/index.html

[5] Erik Derr, Sven Bugiel, Sascha Fahl, Yasemin Acar, and Michael Backes. 2017. Keep me updated: An empirical study of third-party library updatability on android. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2187–2200.

[6] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. 2019. Hypergraph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 3558–3565.

[7] Qiang He, Bo Li, Feifei Chen, John Grundy, Xin Xia, and Yun Yang. 2020. Diversified third-party library prediction for mobile app development. *IEEE Transactions on Software Engineering* 48, 1 (2020), 150–165.

[8] Abhinav Jamwal and Sandeep Kumar. 2025. Third-Party Library Recommendations Through Robust Similarity Measures. In *Proceedings of the 20th International Conference on Evaluation of Novel Approaches to Software Engineering - ENASE*. INSTICC, SciTePress, 635–642. https://doi.org/10.5220/0013366600003928

[9] Abhinav Jamwal and Sandeep Kumar. 2025. Towards an Approach for Project-Library Recommendation Based on Graph Normalization. In *Proceedings of the 20th International Conference on Evaluation of Novel Approaches to Software Engineering - ENASE*. INSTICC, SciTePress, 120–127. https://doi.org/10.5220/0013351900003928

[10] Taeyeon Ki, Chang Min Park, Karthik Dantu, Steven Y Ko, and Lukasz Ziarek. 2019. Mimic: UI compatibility testing system for Android apps. In *2019 IEEE/ACM 41st international conference on software engineering (ICSE)*. IEEE, 246–256.

[11] LearnDataSci. [n. d.]. Jaccard Similarity. https://www.learndatasci.com/glossary/jaccard-similarity/.

[12] Daniel Lee and H Sebastian Seung. 2000. Algorithms for non-negative matrix factorization. *Advances in neural information processing systems* 13 (2000).

[13] Bo Li, Qiang He, Feifei Chen, Xin Xia, Li Li, John Grundy, and Yun Yang. 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*. 466–477.

[14] Bo Li, Haowei Quan, Jiawei Wang, Pei Liu, Haipeng Cai, Yuan Miao, Yun Yang, and Li Li. 2024. Neural library recommendation by embedding project-library knowledge graph. *IEEE Transactions on Software Engineering* (2024).

[15] Zhongxin Liu, Xin Xia, David Lo, and John Grundy. 2019. Automatic, highly accurate app permission recommendation. *Automated Software Engineering* 26 (2019), 241–274.

[16] Ziang Ma, Haoyu Wang, Yao Guo, and Xiangqun Chen. 2016. Libradar: Fast and accurate detection of third-party libraries in android apps. In *Proceedings of the 38th international conference on software engineering companion*. 653–656.

[17] Phuong T Nguyen, Juri Di Rocco, Davide Di Ruscio, and Massimiliano Di Penta. 2020. CrossRec: Supporting software developers by recommending third-party libraries. *Journal of Systems and Software* 161 (2020), 110460.

[18] Phuong T Nguyen, Juri Di Rocco, Davide Di Ruscio, Lina Ochoa, Thomas Degueule, and Massimiliano Di Penta. 2019. Focus: A recommender system for mining api function calls and usage patterns. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1050–1060.

[19] Ali Ouni, Raula Gaikovina Kula, Marouane Kessentini, Takashi Ishio, Daniel M German, and Katsuro Inoue. 2017. Search-based software library recommendation using multi-objective optimization. *Information and Software Technology* 83 (2017), 55–75.

[20] Douglas A Reynolds et al. 2009. Gaussian mixture models. *Encyclopedia of biometrics* 741, 659-663 (2009).

[21] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2019. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903* (2019).

[22] Bernard Rosner, Robert J Glynn, and Mei-Ling T Lee. 2006. The Wilcoxon signed rank test for paired comparisons of clustered data. *Biometrics* 62, 1 (2006), 185–192.

[23] Mohamed Aymen Saied, Ali Ouni, Houari Sahraoui, Raula Gaikovina Kula, Katsuro Inoue, and David Lo. 2018. Improving reusability of software libraries through usage pattern mining. *Journal of Systems and Software* 145 (2018), 164–179.

[24] scikit-learn. 2024. Hinge Loss — scikit-learn 1.3.0 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.hinge_loss.html

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

[26] Ferdian Thung, David Lo, and Julia Lawall. 2013. Automated library recommendation. In *2013 20th Working conference on reverse engineering (WCRE)*. IEEE, 182–191.

[27] Yanchun Wang, Qiang He, Dayong Ye, and Yun Yang. 2017. Formulating criticality-based cost-effective fault tolerance strategies for multi-tenant service-based systems. *IEEE Transactions on Software Engineering* 44, 3 (2017), 291–307.

[28] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*. PMLR, 6861–6871.

[29] Lianghao Xia, Chao Huang, Yong Xu, Jiashu Zhao, Dawei Yin, and Jimmy Huang. 2022. Hypergraph contrastive collaborative filtering. In *Proceedings of the 45th International ACM SIGIR conference on research and development in information retrieval*. 70–79.

[30] Liang Yao, Chengsheng Mao, and Yuan Luo. 2019. Graph convolutional networks for text classification. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 7370–7377.

[31] Runlong Yu, Yunzhou Zhang, Yuyang Ye, Le Wu, Chao Wang, Qi Liu, and Enhong Chen. 2018. Multiple pairwise ranking with implicit feedback. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 1727–1730.

[32] Jiyong Zhang and Pearl Pu. 2007. A recursive prediction algorithm for collaborative filtering recommender systems. In *Proceedings of the 2007 ACM conference on Recommender systems*. 57–64.

[33] Pan Zhou, Xiaotong Yuan, Huan Xu, Shuicheng Yan, and Jiashi Feng. 2019. Efficient meta learning via minibatch proximal update. *Advances in Neural Information Processing Systems* 32 (2019).

Received 16 January 2025; revised 20 June 2025; accepted 10 August 2025