# Survey and Attempted improvements on existing recommendation system

**Authors**
Wentai Li / wentai412@163.com
Yingkai Tang / tangyingkai@163.com
Weiran Su / oceansukila@163.com
Yikai Li / liyk.charlie2023@gdhfi.com
Jingyun Miao / cryangfan@163.com

## Abstract

This paper presents an exploration and enhancement of existing recommendation systems by addressing key challenges such as the cold-start problem and data sparsity. We propose DGSR++, a hybrid model that combines dynamic user-item interaction data with content-based features to enhance recommendation quality. Additionally, we develop MPKG-lite, a lightweight knowledge graph embedding model to mitigate cold-start issues. Besides, we also improved other models such as GTN and ensembled learning. The paper also evaluates the performance of ensemble learning methods. Experimental results demonstrate the superiority of DGSR++ in solving cold-start challenges and improving overall recommendation accuracy compared to traditional approaches.

## 1 Introduction

In the realm of digital information overload, recommendation systems play a crucial role as information filters, offering personalized content to users and reducing the effort required to find relevant material(Dutta). They have a significant impact on improving user experience and improving business revenue or decision making. However, these systems face challenges——the cold-start problem.This problem is when the system lacks enough data about new users or new items to generate accurate recommendations Addressing these challenges is critical to the advancement of recommendation systems and their ability to provide users with a more diverse and satisfying range of choices.

Many methods have been proposed to deal with the issue. They can be broadly categorized into two main approaches. The first is based on collaborative filtering. This is a program that utilizes user-item interactions to predict preferences and has been a cornerstone of recommendation systems due to its ability to capture implicit feedback effectively(Fayyaz Ebrhimian Nawara). In addition, the content-based filtering focuses on the attributes and content of items to generate recommendations and can be another solution. This approach is particularly useful for new users or projects with a limited interaction history, and thus complements collaborative filtering by providing initial recommendations based on project characteristics(Fayyaz Ebrhimian Nawara). As a result, both approaches have their own merits and limitations.

However, previous methods in recommendation systems often rely on strong assumptions that can be challenging to uphold in real-life scenarios. For instance, collaborative filtering methods assume that there is a large amount of user interaction data. This may not be available for new users or items and can lead to the cold-start problem(Rossen). In addition, content-based filtering methods generally assume that project features are sufficiently representative and can be precisely matched to user preferences, yet this can be difficult when the content is not descriptive enough or lacks diversity(Lops). When these assumptions are violated, previous methods may fail to provide accurate recommendations or quickly adapt to changing user preferences, which highlight the need for a more flexible recommendation system that can better deal with sparse data and dynamically evolve user interests.

Many methods are proposed to address cold start problems, especially models based on GNNs. For example, DGRN utilizes a dynamic graph framework to capture user's evolving preferences, performing excellently in changeable situation (Zhang et al.). KGAT has a great ability mitigating

sparsity problems through knowledge graph embedding (Wang et al.). However, there isn't an existing method that can work well in changeable and sparse conditions simultaneously.

## 2 Related works

### 2.1 Recommendation System

A recommendation system is a type of artificial intelligence (AI) algorithm, typically rooted in machine learning, that employs big data to suggest or recommend additional products or services to consumers. These recommendations are formulated based on various criteria, such as past purchases, search history, demographic details, and other relevant factors (Recommendation). These systems harness historical user behavior, preferences, and other ancillary data to surface products and content that are likely to resonate with individual tastes. As such, the recommendation systems play a crucial role in enhancing user experience and driving sales and acting as online salespeople who are intimately familiar with consumer history and inclinations. With the development of science and technology, several algorithms are designed, which are collaborative filtering, content-based, knowledge graph based, and hybrid recommendation systems.

For collaborative filtering. Collaborative filtering uses a matrix to map user behavior for each item in its system. The system then draws values from this matrix to plot as data points in a vector space. Various metrics then measure the distance between points as a means of calculating user-user and item-item similarity (Jacob). Therefore, collaborative filtering systems can be either user-based or item-based, and they often grapple with the "cold-start" problem, which arises when there is insufficient data on new users or items to make accurate recommendations.

Content-based filtering, in contrast, focuses on the attributes of items themselves, utilizing item features to provide recommendations that are particularly effective for new entries in the system without historical interaction data. This approach is particularly good at solving the cold start problem for new users or new projects because it relies on the intrinsic characteristics of the project itself, rather than historical user data. It typically leverages project metadata such as genre, director, actor, and other descriptive attributes.

Leveraging the structured knowledge represented in knowledge graphs, the Knowledge Graph-Based Filtering can capture complex relationships and provide a nuanced understanding of user preferences and item characteristics. In recent years, knowledge graphs have been used in recommendation systems to overcome the user-item interaction sparsity problem and the cold start problem faced by CF methods by taking the attributes of items and users and representing them in a single data structure (Dadoum). Last but not least, a meta approach, which is the Hybrid recommendation system, combine multiple strategies to harness their collective advantages. This method is often integrating content-based and collaborative techniques to mitigate the inherent limitations of each.

### 2.2 Graph Neural Network on Recommendation System

In recent years, graph neural networks have been investigated across a broad spectrum of applications that can be grouped in two scenarios: structural scenarios, where the data has explicit relational structure, and non-structure scenarios, where the relational structure is implicit or absent (Zhou et al. 57-81). Structural scenarios rise from industrial applications including recommendation system and knowledge graphs. Analyzing user-item graphs through learning user and item embeddings lies on the cores of traditional recommendation system based on GNN. Many models based on user-item interaction are proposed in the past few years, such as GC-MC (Berg et al.), PinSage (Ying et al. 974-983) and NGCF (Wang et al.). However, these traditional models are insufficient to distill dynamic collaborative signals and attribute-based collaborative signal, which makes them perform poorly when facing dynamic and sparse situation. In order to solve the problems, two state-of-the-art models based on GNNs using different convolution operators have been proposed respectively. Working with spectral approaches, Dynamic Graph Recommendation Network (DGRN) connects various user sequences through a dynamic graph framework to capture user's evolving preferences (Zhang et al.). Working with spatial approaches, Knowledge Graph Attention Network (KGAT) utilizes knowledge graph to break down the independent interaction assumption by linking items with their attributes, which can alleviate sparsity problem

(Wang et al.).

### 2.3 Ensemble Learning

Ensemble learning aims to obtain better generalization performance by combining the predictions from multiple models (Ganaie et al.). Classical ensemble learning are broadly categorized into bagging, stacking and boosting. Bagging creates subgroups of data, known as bags, that are trained by individual machine learning methods such as decision trees (Ngo et al.). Stacking involves training several different types of models on the same dataset and employing an additional model to learn the best way to integrate their predictions (Dey and Mathur). Boosting sequentially trains a series of weak learners and combines their outputs in a way that corrects the errors of the previous models. For example, XGboost, a scalable tree boosting system, involving a novel tree learning algorithm for handling sparse data and a theoretically justified weighted quantile sketch procedure enables handling instance weights in approximate tree learning (Ying et al.).

## 3 Method

### 3.1 Subsection Problem Setup

Below is a problem restatement. Given the task of recommendation and a dataset, three recommendation models should be designed and aimed to answer the research questions formulated in section[ ]. First, whether the improved models outperform the existing solutions. Second, whether the models resolved the issues such as cold start and over-squashing

We approach the problem by developing a model in three different directions. The first model is to employ the classical ensemble learning approach. The second model utilized the framework of Graph-Based Neural Networks(e.g. GCN, GAT, and SAGEConv) by considering both content-based and collaborative features. The third model explored the usage of the state-of-the-art model - graph transformer.

Before the implementation, we conducted data cleaning, including normalization, dedimensionalization, removing the outlier, and adapting the data format according to the requirements of the used library and codebase. Additionally, there is also an analysis of the features of our dataset and statements about our understanding of it.

After implementing our models, we formulated our evaluation metrics such as RMSE, Recall@k, and F1 score to compare and evaluate the performances of the models

We also have done additional evaluations, such as a sensitivity analysis on the hyperparameters of three of the models.

### 3.2 DGSR++

#### 3.2.1 Brief Overview

We have proposed DGSR++, a hybrid model that combines both dynamic-collaborative user-item interaction data with static content-based features of items.

DGSR++ has a base model called Dynamic Graph Neural Network for Sequential Recommendation(DGSR). It is a pure collaboration-based model that connects different user sequences through a dynamic graph structure, exploring the interactive behavior of users and items with time and order information[1]. We have conducted a full study on this base model, including detecting its performance bottleneck, evaluating its hyperparameters, and inspecting the potential issues of the implementation of the code base. There are serval issues we have spotted and fixed, which are described in the fellow section called implementation detail. Our contributions to this model are that we have improved its compatibility with different types of datasets and solved serval performance bottlenecks, allowing it to perform better in general cases.

Given these improvements on the base model DGSR, a newly designed content-based model is added as additional layers to elaborate the embedding of items by making it aware of the lexical and categorical meaning of the entities(e.g. tags and other metadata) that are related to them. Specifically, DGSR++ incorporates tag-based embeddings that capture the semantic relationships between items through their associated tags. This allows the model to understand the latent connections between items that may not be apparent from collaborative data alone, improving recommendations in scenarios where user-item interaction data is sparse or unavailable. It has been shown that this approach can be used to mitigate the cold start issue by our result.

### 3.2.2 Implementation details of the base model

This section describes the drawbacks and challenges we encountered when researching the base model of DGSR.

First, based on the given subgraph sampling algorithm, which is given that $\mathcal{G} = \{(u, i, t, o_u^i, o_i^u) | u \in \mathcal{U}, i \in \mathcal{V}\}$, we need to sample a subgraph denoted as $\mathcal{G}_u^m(t_k)$, where $m$ is the hyper-parameter used to control the size of the sub-graph, $u$ is the anchor node used as the center of the sampling, $t_k$ is the kth timestamp in the time series and the most recent $n$ first-order neighbors will be selected. Given the neighborhood of $u$ is denoted as $\mathcal{N}_u$ , each item selected is $i \in \mathcal{N}_u$.

However, the implementation's approach to storing sampled subgraphs is suboptimal in terms of efficiency. The method creates multiple directories, each containing several files. While caching files in storage can eliminate the need for generating subgraphs and potentially accelerate execution in some instances, this approach fails to consider I/O efficiency adequately. For systems utilizing HDDs or older storage devices, file fragmentation can lead to inefficient disk head movement, resulting in poor performance.

Furthermore, considering that deep learning models are often trained on remote platforms like Google Colab, we discovered that DGSR's data loading approach was unable to read data in parallel, despite the use of multiple workers. This is attributed to cloud storage load balancing mechanisms that limit frequent I/O operations, causing our model to spend an inordinate amount of time on data loading alone.

To address these issues, we implemented an alternative method to store subgraphs in memory. Although this approach results in higher memory usage, it offers improved efficiency. Users have the option to choose between the original file-based storage method and our memory-based alternative, depending on their specific requirements and system capabilities.

### 3.2.3 Proposed MPKG-lite

This section describes our proposed submodel for generating item embeddings, which incorporates content-based features to address the cold-start issue inherent in collaborative models like DGSR.

Our model draws inspiration from KGAT, which we introduced earlier. By adopting the concept of capturing high-order relationships between different entities (e.g., tags and items), we've developed a modified implementation. We call this embedding generation model Message Passing Knowledge Graph-lite (MPKG-lite).

We've implemented an option to control the model's complexity by adjusting the types of knowledge graph entities and relations. This design choice stems from two factors. First, our dataset contains relatively few relationships, making a full forward MPNN pass on a knowledge graph computationally expensive. Second, deep message-passing in densely connected graphs can lead to overfitting and gradient explosion. Moreover, KGAT essentially comprises two models: one for generating embeddings from a knowledge graph, and another for training on these embeddings to make predictions. We aim to replace the second model with DGSR, as it shows the potential to outperform traditional approaches.

Specifically, The integration of DGSR allows us to use the rich item embeddings generated by MPKG-lite and fine-tune them through sequential interactions. This combined approach leverages both content-based and collaborative filtering techniques, enabling better performance in situations where user interactions are sparse or evolving.

In addition, during our work on integrating the output item embedding of the MPKG-lite and the initial item embedding generated by DGSR, we found out that the issue of over-squashing has minimal impact on bipartite graphs like that employed in DGSR but is a key issue of MPKG-lite. Specifically, this is because in a bipartite graph, nodes can only connect to nodes in the opposite set, which may reduce the number of long-range dependencies compared to more general graph types. There is no need to establish an edge new connection from node to node for the sake of rewiring because it would only lead to extra noise in the training graph. However, for a knowledge graph that contains different types of edges and nodes, employing rewiring algorithms could optimize the performance of the algorithm because it can provide an optimized way to capture high-order relationships.

### 3.2.4 Implementation details

We begin by initializing item embeddings using a pre-trained Word2Vec model. The initial embedding matrix, $\mathbf{E}_{\text{word2vec}} \in \mathbb{R}^{|\text{vocab}| \times d}$, encodes content-based features. Here, $|\text{vocab}|$ represents the vocabulary size, and $d$ is the embedding di-

mension. To address the high dimensionality of these embeddings, we employ Sparse Random Projection (SRP). This technique reduces the embeddings' dimensionality while retaining the most significant features.

The projection can be formulated as:

$$\mathbf{E}_{\text{reduced}} = \mathbf{T}_{\text{SVD}} \cdot \mathbf{E}_{\text{word2vec}} \in \mathbb{R}^{|\text{vocab}| \times d'}$$

where $d'$ is the reduced embedding dimension, and $\mathbf{T}_{\text{SVD}}$ is the random projection matrix generated using SRP. This process ensures that we retain the most important aspects of the content-based features while significantly reducing computational complexity.

MPKG-lite constructs a lightweight knowledge graph using the reduced embeddings. Nodes represent items and tags, while edges represent the relationships between them. The knowledge graph is defined as $G = (V, E)$, where $V$ are the nodes (items and tags), and $E$ are the edges (relations between items and tags).

We employ a Message Passing Neural Network (MPNN) for the knowledge graph. The embeddings for each node are updated by aggregating information from its neighbors. The update rule for each item node $i$ at layer $k$ is:

$$\mathbf{h}_i^{(k+1)} = \sigma\left(\mathbf{W}^{(k)} \cdot \left(\mathbf{h}_i^{(k)} + \sum_{j \in \mathcal{N}(i)} \frac{1}{|\mathcal{N}(i)|} \mathbf{h}_j^{(k)}\right)\right)$$

Here, $\mathbf{h}_i^{(k)}$ is the embedding of item $i$ at layer $k$, $\mathbf{W}^{(k)}$ is the weight matrix, and $\mathcal{N}(i)$ denotes the set of neighbors (tags) for item $i$. This allows each item embedding to be enriched with higher-order relationships captured in the knowledge graph.

Next, given our dataset's sparsity, we control the knowledge graph's complexity by selectively pruning weak or irrelevant relationships. This approach reduces overfitting and maintains a lightweight graph, particularly important in cases where deep message passing might lead to gradient explosion. We achieve this by dynamically adjusting the graph during training, allowing only significant relationships to contribute to the embedding generation process.

After completing the message-passing process, we feed the final embeddings into DGSR (Dynamic Graph Sequential Recommendation) to model user-item interactions over time. Unlike KGAT's second model, DGSR adapts to user preferences in real-time as interactions evolve. The final embeddings from MPKG-lite serve as DGSR's initialization, providing rich content-based features that enhance its collaborative filtering capabilities, particularly in cold-start scenarios.

The final transformation for each item embedding after message passing is:

$$\mathbf{h}_i^{\text{final}} = \text{LeakyReLU}(\mathbf{W}_{\text{final}} \cdot \mathbf{h}_i^{(L)})$$

where $\mathbf{h}_i^{(L)}$ is the embedding after $L$ message-passing layers. These final embeddings are then combined with DGSR's sequential features through:

$$\mathbf{h}_i^{\text{combined}} = \tanh(\mathbf{W} \cdot \text{concat}(\mathbf{h}_i^{\text{final}}, \mathbf{h}_i^{\text{DGSR}}))$$

This combination ensures that both content-based and collaborative features are integrated, providing a holistic representation of the item.

To enhance generalization and mitigate overfitting, we implement dropout during the message-passing phase. This technique randomly deactivates a portion of the nodes' embeddings, preventing the model from over-relying on specific graph relationships. We also employ batch normalization to stabilize training and combat gradient explosion, particularly in the MPNN's deeper layers.

#### 3.2.5 Hyperparameter optimization

We have also optimized our DGSR++ model by conducting hyperparameter optimization. Hyperparameter optimization is crucial to improving the performance of machine learning models, as it allows us to fine-tune various parameters that control the model's learning process.

In this study, we focused on optimizing several key hyperparameters that significantly impact model performance. The specific hyperparameters and their corresponding ranges are as follows:

- **Learning Rate**: This controls the step size during model weight updates. We optimized it over a logarithmic scale between $5 \times 10^{-4}$ and $5 \times 10^{-3}$, aiming to strike a balance between fast convergence and avoiding oscillations during training.
- **L2 Regularization**: To prevent overfitting, we adjusted the L2 regularization term within

the range of $2 \times 10^{-4}$ to $5 \times 10^{-3}$. The goal was to find an appropriate value that constrains the model's complexity without compromising performance.

- **Number of GNN Layers**: For the label generation model, the number of GNN layers was optimized between 2 and 5. More layers enable the model to capture more complex relationships between labels, but excessive depth could lead to gradient vanishing or overfitting.

- **Negative Slope for Leaky ReLU**: The slope of the negative side of the Leaky ReLU activation function was tuned within the range of 0.001 to 0.1. This ensures that even with small gradients, the model can effectively backpropagate and learn from negative inputs.

- **Feature Dropout**: To reduce over-reliance on specific input features, we optimized the feature dropout rate between 0.1 and 0.5. This helps mitigate overfitting by randomly dropping features during training.

- **Attention Dropout**: The dropout rate in the attention mechanism was adjusted between 0.1 and 0.5. This is crucial to prevent the attention mechanism from focusing too heavily on specific node features, thereby improving the model's generalization ability.

- **Number of GNN Layers (Model Layer Number)**: We optimized the number of GNN layers within a range of 3 to 5. This ensures the model has sufficient depth to capture the structure of the graph while avoiding over-squashing information due to excessive layers.

For hyperparameter optimization, we employed Bayesian Optimization using the Tree-structured Parzen Estimator (TPE) to explore the hyperparameter space. This method was selected due to its ability to efficiently balance exploration and exploitation, reducing the computational cost compared to traditional grid search, while preserving higher accuracy than randomized search. Additionally, TPE can further balance exploration and exploitation, which is more efficient than a regular Bayesian Optimization implementation. The intuition behind Bayesian optimization is that it builds a surrogate model to estimate the distribution of the loss function, updating the surrogate as new results are obtained to progressively focus on the most promising regions of the hyperparameter space.

The final hyperparameters were selected based on the model's performance on the validation set, minimizing the value of the loss function. By iterative attempts, we ensured that the model achieved near-optimal performance.

This optimization process was conducted over 10 trials, with each trial dynamically adjusting the hyperparameter ranges based on the results from previous trials, gradually converging toward the global optimum. As a result, the model's performance was significantly improved through these optimized hyperparameters.

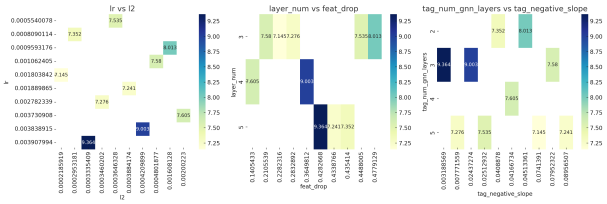


Figure 1: Hyperparameter optimization results

From the heatmaps, we can see that the combination with the lowest loss is as follows:

- **Learning Rate (lr)**: 0.001804
- **L2 Regularization (l2)**: 0.000219
- **Number of MPKG-lite Layers (tag_num_gnn_layers)**: 5
- **Tag Negative Slope**: 0.074139
- **Feature Drop Rate (feat_drop)**: 0.228232
- **Attention Drop Rate (attn_drop)**: 0.240454
- **Number of Layers (layer_num)**: 3
- **Loss**: 7.1452

Therefore, we selected this combination for our later experiments and evaluations.

### 3.3 GTN

The Graph Transformer Network (GTN) is a deep learning model proficient in generating graph structures. Contrary to the frequently used model, Graph Neural Networks (GNNs), which necessitate a fixed and homogeneous graph structure,

GTNs excel at processing heterogeneous graphs. They facilitate the generation of meta-path graphs and the learning of node representations within these graphs. By harnessing the power of GTNs, we can enhance the performance of recommendation systems by uncovering intricate connections between users and items. In this project, we developed a recommendation system based on a GTN model sourced from GitHub. Our dataset consisted of a matrix where each row represents a unique user and each column corresponds to a video game. The elements in the matrix represent user scores for video games, with a score of 0 indicating that a user has not played the game. To train the model, we utilized the datasets provided by the original developer, which included node features, edges connecting nodes, and target labels. After collecting and applying the necessary hyperparameters to the system, we invoked the function designed to generate meta-paths within the model. This step allowed us to discern the relationships between users, specifically in the online datasets we procured. The recommendation system predicted the interactions, scoring in this case, of the users to each game and subsequently outputted personalized game suggestions for each user, based on their interconnections within the graph.

### 3.4 Traditional Method + Ensembled learning

We apologize for not being able to present the analysis of this part bacause our members were unable to finish this part of description on time.

## 4 Experiments

In this section, we perform experiments on three models that we either proposed or worked on. The experiments are run on a specialized dataset to evaluate the performance of our models. We aim to answer the following questions through experiments:

- **RQ1:** Which model out of all the three has the best performance?
- **RQ2:** How far is each of the models affected by the cold-start issue?

### 4.1 Experiments setup

#### 4.1.1 Evaluation matrics

We utilize two commonly used metrics, Recall@K and Normalized Discounted Cumulative Gain (NDCG), to assess all models. Recall@K measures the proportion of true items within the top K recommendations, while NDCG@K is a rank-sensitive metric, where a higher NDCG indicates that target items are ranked closer to the top.

#### 4.1.2 Baselines

To demonstrate the effectiveness, we plan to compare our models with the following models:

- **GRU4Rec+**, an improved RNN-based model that adopts a different loss function and sampling strategy for Top-K recommendation.
- **KGAT**, Knowledge Graph Attention Network, which incorporates high-order relations to refine item embeddings and improve recommendations.
- **FPMC**, a model that combines matrix factorization and first-order Markov Chains to capture users' long-term preferences and item-to-item transitions.

However, due to a limited amount of time and the fact that only Alvin is working on this part of codebase, we were unable to adjust the data required for these standard models. Nonetheless, we can still compare our models.

#### 4.1.3 Datasets

In the context of my research, I adopted a methodical strategy for managing data. Firstly, missing data point is the main issues for me. The treatment of missing values is executed through a combination of imputation methods. For instance, our dataset of movie only include the name of them. Instead, we need more information. Though the web crawler in python. We eventually got the label of those movie. Furthermore, since different forms of dataset are fixed in different program, the dataset needs to be transform. With the help of python pandas, different dataset are available for the program. By analyzing the descriptive statistics of the data and visualizing its patterns, I gained valuable insights into the fundamental phenomena that guide the next steps in data cleaning and preprocessing. This rigorous data extraction procedure is not only essential to maintaining the sanctity of empirical investigation, but also provides fertile ground for subsequent analytical work.

### 4.2 Performance Comparison

Based on our experiment, the following results were obtained:

| Metrics | DGSR++ | Ensembled |
|---|---|---|
| Recall@5 | 0.7989 | 0.6036 |
| Recall@10 | 0.8682 | 0.6413 |
| NDCG@5 | 0.6306 | 0.6992 |
| NDCG@10 | 0.7243 | 0.8057 |

Table 1: Performance Comparison of DGSR++ and Ensemble Model

The performance of the models was evaluated using Recall@K and NDCG@K, two widely used metrics in recommendation systems. Recall@K measures the proportion of relevant items found in the top K recommendations, while NDCG@K considers the rank of the relevant items in the recommendation list. The results of our experiments are summarized in Table X.

Firstly, the **DGSR++** model achieved a **Recall@5** of **0.7989** and a **Recall@10** of **0.8682**, demonstrating a strong ability to accurately recommend relevant items within the top 5 and top 10 suggestions. In terms of ranking quality, DGSR++ scored **0.6306** on **NDCG@5** and **0.7243** on **NDCG@10**, indicating that the model not only recalled relevant items but also ranked them relatively high in the recommendation list.

The **Ensemble** model showed lower performance on recall metrics, with **Recall@5** of **0.6036** and **Recall@10** of **0.6413**. These results suggest that the ensemble method is less effective at finding relevant items compared to DGSR++. However, the ensemble model performed better in ranking relevant items, achieving **NDCG@5** of **0.6992** and **NDCG@10** of **0.8057**. This indicates that although the ensemble model retrieved fewer relevant items, it positioned those items higher in the recommendation list.

#### 4.2.1 Study of the performance of different models (RQ1)

Based on **Recall** metrics, DGSR++ outperforms the ensemble model, especially in Recall@10, where DGSR++ achieves **0.8682** compared to **0.6413** for the ensemble model. This highlights DGSR++'s superior ability to retrieve relevant items in general. However, on the **NDCG** metrics, the ensemble model shows a slight advantage, with higher NDCG@5 and NDCG@10 scores, suggesting that it ranks relevant items more effectively.

Thus, the **best-performing model** depends on the task. If the priority is maximizing the number of relevant items retrieved, **DGSR++** is the better choice. However, if ranking the retrieved items is more critical, the **ensemble model** shows some benefits.

#### 4.2.2 Study of how each model handles the cold-start problem (RQ2)

The cold-start problem arises when there is insufficient data about new users or items, making accurate recommendations challenging. DGSR++, by combining dynamic user-item interactions with content-based features, appears to handle this issue more effectively. Its superior Recall scores suggest that it can make better recommendations even with limited interaction data, likely due to its use of tag-based embeddings and content features.

On the other hand, the ensemble model, which relies more on collaborative filtering, struggles with cold-start situations, as evidenced by its lower recall performance. This is expected, as collaborative filtering often requires a substantial amount of interaction data to perform well.

In conclusion, **DGSR++** shows stronger overall performance in retrieving relevant items and addressing the cold-start problem, while the **ensemble model** provides better ranking of the retrieved items. The choice between these models depends on the specific objectives of the recommendation task.

## 5 Conclusion

This paper presents a study focused on improving recommendation systems by tackling key issues like the cold-start problem and sparse data. Our main contribution is DGSR++, a hybrid model that combines user-item interactions with content-based features through tag-based embeddings. This model has shown strong performance, particularly in addressing the cold-start problem, by using both collaborative and content-based methods.

We developed MPKG-lite, a lightweight model that generates item embeddings using knowledge graphs. By capturing relationships between items and their features, it provides an efficient solution for handling sparse data.

Additionally, we explored an updated version of the Graph Transformer Network (GTN), which

improved the ability to model complex relationships between users and items, showing potential for future work in leveraging graph structures for recommendations.

Besides, we also looked at ensemble learning, combining multiple models to improve overall performance. While DGSR++ excelled in finding relevant items, the ensemble model performed better in ranking them accurately. We found out that each model has its strengths: DGSR++ and MPKG-lite are particularly good at handling cold-start issues, while ensemble learning improves ranking. However, due to the unfinished parts of our researchers, such as the improved GTN part, we can hardly draw up its conclusion.

## 6 Limitations

Admittedly, there are lots of limitations in our work. The biggest one is that all researchers are high school students, which means that we are not experienced enough to achieve innovative outcomes while debugging efficiently.

Specifically, for the DGSR++ part, we don't have enough resources and time to carry out enough iterations of hyperparameter optimization. Even if we have tried very hard to optimize the model that we are using, the efficiency is not very satisfactory. Specifically, there is a common guideline to conduct at least 10 to 20 evaluations per hyperparameter[[source](https://neptune.ai/blog/how-to-optimize-hyperparameter-search)]. There are 7 essential hyperparameters in our model, which means a thorough Bayesian hyperparameter optimization requires at least 70 iterations to achieve tangible improvement. But it would be too long for our available resources - only 10 iterations took us 11 hours. Therefore, there must be a more optimal permutation of hyperparameter that could overcome the bottleneck of gradient descent that we are facing currently. Additionally, most of our trained parameters only endured around 10 epochs, which is inadequate but takes a considerable amount of time.

Moreover, for the MPKG-lite part, the reason why its name involves "lite" is because the complete and generic way to generate item embedding is to have multiple relationships and entities. The current approach is only a compromise. Even if we leave out an endpoint to implement this feature, we don't have the data and resources to finish this feature within our available time.

## 7 Future works

### 7.1 For GTN part

In the future, we plan to apply a Generative Adversarial Network (GAN) to our system to improve the accuracy of recommendations. Specifically, we will use our existing GTN-based system as the generator and focus on training the discriminator. The discriminator will be trained using the non-zero entries of the user-game matrix as real data points, representing actual user-game interactions, while the predicted scores generated by the GTN for games that users have not played will be treated as fake samples. The recommendations generated by the GTN will then be evaluated by the discriminator, which will classify each recommendation as either "real" or "fake" based on whether it is likely to be relevant to the user.

## References


Amine Dadoun. 2023. Introduction to knowledge graph-based recommender systems. Medium, 23 May 2023. Available at: https://towardsdatascience.com/introduction-to-knowledge-graph-based-recommender-systems-34254efd1960.

Jacob Murel. 2024. What is collaborative filtering? — IBM. IBM, 18 Mar. 2024. Available at: https://www.ibm.com/topics/collaborativefiltering#:~:text=Collaborative%20filtering%20uses%20a%20matrix. Accessed 13 Aug. 2024.

Google. 2022. Recommendation systems - Google Search. Google.com. Available at: https://www.google.com/search?q=Recommendation+systems&oq=Recommendation+systems&gs_lcrp=EgZjaHJvbWUyBggAEEUYOdIBCDY5MmowajE1qAIIsAIB&sourceid=chrome&ie=UTF-8. Accessed 13 Aug. 2024.

Jie Zhou, et al. 2020. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81.

Rianne van den Berg, Thomas N. Kipf, and Max Welling. 2017. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*.

Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on*

*knowledge discovery and data mining*, pages 785–794.

Xiang Wang, et al. 2019. Kgat: Knowledge graph attention network for recommendation. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 950–958.

Mengqi Zhang, et al. 2022. Dynamic graph neural networks for sequential recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 35(5):4741–4753.

Xiang Wang, et al. 2019. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pages 165–174.

Rex Ying, et al. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983.

Ratul Dey and Rajeev Mathur. 2023. Ensemble learning method using stacking with base learner, a comparison. In *Lecture notes in networks and systems*, pages 159–169. doi:10.1007/978-981-99-3878-0_14.

M. A. Ganaie, et al. 2022. Ensemble deep learning: A review. *Engineering Applications of Artificial Intelligence*, 115:105151. doi:10.1016/j.engappai.2022.105151.

Giang Ngo, et al. 2022. Evolutionary bagging for ensemble learning. *Neurocomputing*, 510:1–14. doi:10.1016/j.neucom.2022.08.055.