VIETNAM NATIONAL UNIVERSITY – HOCHIMINH CITY

THE INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SICENCE AND ENGINEERING



**GNN IN RECOMMENDER SYSTEM**

By

**Trinh Quang Anh**

A thesis submitted to the School of Computer Science and Engineering in partial fulfillment of the requirements for the degree of   
Bachelor of Science in Computer Science

Ho Chi Minh city, Viet Nam  
2025

**GNN IN RECOMMENDER SYSTEM**

APPROVED BY:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
  
\*(*Type Committee names beneath lines*)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
(*Typed Committee name here*)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
(*Typed Committee name here*)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
(*Typed Committee name here*)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
(*Typed Committee name here*)

THESIS COMMITTEE   
(Whichever applies)

ACKNOWLEDGMENTS

It is with deep gratitude and appreciation that I acknowledge the professional guidance of Dr. Ha Viet Uyen Sinh. His constant encouragement and support helped me to achieve my goal.

My gratitude goes to the other members of the Thesis Course under the guide of Mr. Synh, Mr. Nguyen Thai Binh and Mr. Le Thanh Nam. Their technical help and good humor made these years a great learning experience. I am grateful to the faculty of the School of Computer Science and Engineering of the International University, especially Dr. Le Duy Tan, for his support from the beginning of my studies. Gratitude is also expressed to the members of my family, who alway support and beleive in me.

TABLE OF CONTENTS

[ACKNOWLEDGMENTS iii](#_Toc217045170)

[TABLE OF CONTENTS iv](#_Toc217045171)

[LIST OF ALGORITHMS ix](#_Toc217045172)

[LIST OF TABLES ix](#_Toc217045173)

[LIST OF FIGURES x](#_Toc217045174)

[ABBREVIATION xi](#_Toc217045175)

[ABSTRACT xii](#_Toc217045176)

[CHAPTER I INTRODUCTION 1](#_Toc217045177)

[1.1. Background 1](#_Toc217045178)

[1.2. Current knowledge and advances in recommendation model 2](#_Toc217045179)

[1.3. Problems gaps 2](#_Toc217045180)

[1.4. Scope and objectives 3](#_Toc217045181)

[1.5. Structure of thesis 4](#_Toc217045182)

[CHAPTER II LITERATURE REVIEW 6](#_Toc217045183)

[2.1. Evolution of Recommendation Systems 6](#_Toc217045184)

[2.1.1. Memmory base methods 7](#_Toc217045185)

[2.1.2. Latent factor models 8](#_Toc217045186)

[2.1.3. Neural Collaborative Filtering 9](#_Toc217045187)

[2.2. Graph-based recommendation models 9](#_Toc217045188)

[2.2.1. Graph base of recommendation data 9](#_Toc217045189)

[2.2.2. Graph Convolutional Networks (GCN) 9](#_Toc217045190)

[2.2.3. Neural Graph Collaborative Filtering (NGCF) 10](#_Toc217045191)

[2.2.4. Light Graph Convolution Netwok (LightGCN) 10](#_Toc217045192)

[2.3. Limitation of existing GNN-based model 11](#_Toc217045193)

[2.3.1. Treating all observed interaction as positive signals 11](#_Toc217045194)

[2.3.2. Noise increases as the graph spreads information. 12](#_Toc217045195)

[2.3.3. Limitations of negative samping in implicit feedback 12](#_Toc217045196)

[2.3.4. Limitation of heterogeneous feedback and side information 13](#_Toc217045197)

[2.3.5. Implications for LightGCN 13](#_Toc217045198)

[2.3.6. Motivation for a new framework 14](#_Toc217045199)

[CHAPTER III THEORETICAL FOUNDATION 15](#_Toc217045200)

[3.1. Graph-based Recommendation Overview 15](#_Toc217045201)

[3.2. Graph Neural Networks (GNNs) 15](#_Toc217045202)

[3.3. Graph Convolution Network (GCN) 18](#_Toc217045203)

[3.3.1. Overview 18](#_Toc217045204)

[3.3.2. Architecture 18](#_Toc217045205)

[3.3.3. Pseudocode 19](#_Toc217045206)

[3.4. Neural Graph Collaborative Filtering (NGCF) 19](#_Toc217045207)

[3.4.1. Architecture 20](#_Toc217045208)

[3.4.2. Pseudocode 23](#_Toc217045209)

[3.5. Light Graph Convolutional Networks (LightGCN) 24](#_Toc217045210)

[3.5.1. Architecture 24](#_Toc217045211)

[3.5.2. Pseudocode 27](#_Toc217045212)

[3.6. Limitations of LightGCN 28](#_Toc217045213)

[3.7. Motivation for Denoising with Random Forest 30](#_Toc217045214)

[CHAPTER IV METHODOLOGY 32](#_Toc217045215)

[4.1. Research design 32](#_Toc217045216)

[4.2. Tenrec dataset 35](#_Toc217045217)

[4.2.1. Dataset description 35](#_Toc217045218)

[4.2.2. Data schema 36](#_Toc217045219)

[4.2.3. Data distribution 37](#_Toc217045220)

[4.2.4. Data analysis 38](#_Toc217045221)

[4.3. Data Preprocessing and Semi-Supervised Denoising 39](#_Toc217045222)

[4.3.1. Three-Tier Feedback Stratification 40](#_Toc217045223)

[4.3.2. Semi-Supervised Pseudo-Labeling 41](#_Toc217045224)

[4.4. Graph-Based Representation Learning with LightGCN 42](#_Toc217045225)

[4.4.1. Graph Construction 42](#_Toc217045226)

[4.4.2. Hybrid Negative Sampling Strategy 43](#_Toc217045227)

[4.4.3. LightGCN Architecture 43](#_Toc217045228)

[4.4.4. Optimization Objective 44](#_Toc217045229)

[4.5. Ensemble Learning for Re-Ranking 44](#_Toc217045230)

[4.5.1. Feature Engineering 44](#_Toc217045231)

[4.5.2. Random Forest Re-Ranker 44](#_Toc217045232)

[4.6. Evaluation metrics 45](#_Toc217045233)

[4.6.1. Precision@K 45](#_Toc217045234)

[4.6.2. Recall@K 46](#_Toc217045235)

[4.6.3. NDCG@K (Normalized Discounted Cumulative Gain at K) 46](#_Toc217045236)

[4.7. Demo website 47](#_Toc217045237)

[CHAPTER V EXPERIMENTS AND EVALUATION 49](#_Toc217045238)

[5.1. Backbone model 49](#_Toc217045239)

[5.1.1. Impact of GNN layers 49](#_Toc217045240)

[5.1.2. Backbone comparisons 50](#_Toc217045241)

[5.1.3. Backbone Selection summary 53](#_Toc217045242)

[5.2. Hybrid model with data denoising and true negative sampling 53](#_Toc217045243)

[5.3. Ablation study 56](#_Toc217045244)

[5.3.1. Baseline and Reranking 57](#_Toc217045245)

[5.3.2. Impact of denoising under click-only evaluation 57](#_Toc217045246)

[5.3.3. True negative sampling 58](#_Toc217045247)

[5.3.4. Full framework 58](#_Toc217045248)

[5.3.5. Key insight from study 58](#_Toc217045249)

[CHAPTER VI CONCLUSION AND FUTURE WORK 60](#_Toc217045250)

[6.1. Conclusion 60](#_Toc217045251)

[6.2. Future work 61](#_Toc217045252)

[LIST OF REFERENCES 62](#_Toc217045253)

[APENNDIX 65](#_Toc217045254)

LIST OF ALGORITHMS

[Algorithm 1: Graph Convolution Network (GCN). 17](#_Toc217046788)

[Algorithm 2: Neural Graph Collaborative Filtering (NGCF). 17](#_Toc217046789)

[Algorithm 3: Light Graph Convolution Network (LightGCN). 21](#_Toc217046790)

[Algorithm 4: Random Forest Interaction denoising. 24](#_Toc217046791)

[Algorithm 5: Random Forest reranking. 25](#_Toc217046792)

LIST OF TABLES

[Table 1: GNNs high order (layers). 49](#_Toc217045030)

[Table 2:Model comparisons. 50](#_Toc217045031)

[Table 3: Comparison with framework. 54](#_Toc217045032)

[Table 4: Ablation experiments. 56](#_Toc217045033)

LIST OF FIGURES

[Figure 1: example of GNN 17](#_Toc216875297)

[Figure 2: An illustration of LightGCN model architecture. 17](#_Toc216875298)

[Figure 3: Framework pipeline. 19](#_Toc216875299)

[Figure 4: Data sources of Tenrec (Source: (2)). 20](#_Toc216875300)

[Figure 5: Item distribution. (a) and (b) are item popularity plotted in raw and log-log scales; (c) is the item session length distribution. 22](#_Toc216875301)

[Figure 6: Pre-processing pipeline. 24](#_Toc216875302)

[Figure 7: Graph Neural Network pipeline. 27](#_Toc216875303)

[Figure 8: Demo website 32](#_Toc216875304)

[Figure 9: Training loss and validation metric of MF, with training loss and valid NDCG in in the left and all validation metrics in the right. 36](#_Toc216875305)

[Figure 10: Training loss and evaluation metrics of LightGCN, with training loss and valid NDCG in the left and validation metrics on the right. 37](#_Toc216875306)

ABBREVIATION

|  |  |
| --- | --- |
| **Abbreviation** | **Explanation** |
| RS | Recommender System |
| CF | Collaborative Filtering |
| BPR | Bayesian Personalized Ranking |
| CBR | Case-Based Reasoning |
| k-NN | k-Nearest Neighbor |
| MF | Matrix Factorization |
| NCF | Neural Collaborative Filtering |
| GNN(s) | Graph Neural Network |
| GCN | Graph Convolution Network |
| NGCF | Neural Graph Collaborative Filtering |
| LightGCN | Light Graph Convolution Network |
| RF | Random Forest |

ABSTRACT

Recommender systems assist people in finding useful items from large online platforms. Collab Filtering using Graph Neural Network (GNN), especially LightGCN, has demonstrated remarkable performance in capturing intricate interaction patterns between users and items. However, the usual setting of implicit recommender systems in GNN-based recommenders considers all known interactions to be positive and all unknown ones to be negative. This is quite impractical in real-world applications, especially on short video and feed-based websites where user interactions are noisy, diverse, and not always clear.

In this thesis, noisy implicit feedback and suboptimal negative sampling are analyzed using a large-scale real-world dataset named Tenrec, provided by Tencent. This dataset demonstrates that clicks and view actions do not necessarily reflect actual user preference, although likes, shares, and follow actions are more accurate expressions of user interest than clicks and views.

In order to address all these problems, I introduce the Hybrid Cascaded Recommendation Framework. In this approach, first, there is a Random Forest-based filter and relabeler of ambiguous interactions based on various behaviors. Then, LightGCN is learned on the cleaned user-item graph using a heterogeneous method of negative sampling. Finally, there is again a Random Forest-based reranker using learned representations and side information.

Evaluation experiments conducted on the Tenrec dataset demonstrate the superiority of the proposed framework in ranking and better identification of the ground-truth preferences over traditional LightGCN. The significance of data quality and preference-aware representation in GNN-based recommenders is emphasized by this study.

The main contributions of this thesis are summarized as follows:

* Overview documents on the GNN-based model and my framework particular on recommender system with there architecture and limited.
* I analyze the limitation of treating all clicks as positive signals in GNN-based recommendation, and propose a behavior-aware denoising strategy to better approximate true **user preference** from noisy implicit feedback
* I develop a **cascaded framework pipeline** that combines interaction denoising (Random Forest), representation learning (LightGCN on a refined graph), and ensemble-based reranking to improve ranking quality in noisy short-video settings.
* I built a intuitive demo website that showcases the recommendation pipeline and allows users to interactively view top-N recommendations, making the proposed method easier to understand and reproduce.

# INTRODUCTION

## Background

The rapid expansion of online data from e-commerce sites, social networking sites, and online streaming services, leading to information overflow for users. Recommender system have become a tool to address this challenge by providing personalized recommendations.

The traditional approaches, such as collaborative filtering or content-based filtering, are based on interaction or content. They perform well but are unable to grasp the complex relationships. This is because the actions taken by a user are diverse, unlike the traditional assumption of only one type.

GNNs are currently a competitive solution for recommenders when modeling users and items as nodes of a graph. They leverage neighborhood information to capture higher-order signals efficiently. LightGCN is one of the most competitive baselines due to its simplicity, efficiency, and effectiveness.

However, on real-world data such as in the Tenrec setting, issues become more complicated. It is not only a challenge in terms of complexity in models but also in figuring out how preferences are learned among so much noisy data.

## Current knowledge and advances in recommendation model

Traditional approaches such as Matrix Factorization and Neural Collaborative Filtering infer from historical behavior but might neglect the relationship between ratings. To remedy this issue, graph-based approaches rely on graphs. Some models such as GCN, GAT, or NGCF propagate information over several steps. Others like LightGCN improve this by concentrating on neighborhood aggregation, making it easier to handle training stability and efficiency.

However, the most GNN-based recommenders, including LightGCN, rely on the following assumption: observed interactions are assumed positive, and unobserved ones are assumed to be negative using random sampling. This is an unrealistic assumption since users view items that they do not want to interact with, yet these are actual negatives and not unobserved.

Framework will be proposed within this thesis, integrating semi-supervised interaction denoising, learning with LightGCN, and reranking through ensembling. The key is that using strong feedback explicitly and distinguishing true preference from exposure, we can get more useful representations and better recommendations in large and noisy spaces.

## Problems gaps

There is a gap between state-of-the-art models using GNN and the real-world recommendation requirement that can be seen using the Tenrec dataset. Tenrec is a large-scale real-world dataset for feed recommendations. It contains a variety of interactions including clicks, likes, shares, and follows, and also contains exposure-based negative samples. Main difficulties identified in Tenrec:

* + **Behavioral ambiguity**: Clicks and brief view times make many behaviors unclear. They do not necessarily indicate the users' desires.
  + **Data imbalance**: The number of explicit positive instances such as likes and shares is much lower compared to the number of clicks.
  + **True negatives exist**: There are objects that users ignore, providing useful negative feedback that is missed by random negative sampling.
  + **Rich Side Information**: We have demographics about users, as well as item information, but CF doesn’t make good use of it.

The standard LightGCN considers all types of feedback equally, and it does not model true negatives or incorporate side information. Therefore, representations learned by standard LightGCN may not align with actual user preferences.

## Scope and objectives

The purpose of this thesis is to bridge the gap between graph neural network-based collaborative filtering and the demands in practical applications involving noisier implicit feedback. It seeks to address:

* + - What kind of methods or preprocessing might be needed in order to filter noisy implicit feedback data?
    - How is representation learning influenced by using true negative feedback in GNN-based recommenders?
    - Can a hybrid model that integrates graph learning with ensemble-based reranking perform better than the standard LightGCN?

To address these questions, this thesis proposes the Hybrid Cascaded Recommendation Framework. This framework integrates semi-supervised interaction noise removal, representation learning with LightGCN, and reranking with ensemble methods. The approach here is that by explicitly capturing the characteristics of strong feedback and the ability to discern preference from exposure, more precise depictions and suggestions can be produced.

## Structure of thesis

This report is structured to provide a comprehensive analysis of the recommendation system evaluation, organized into several key sections:

* The Abstract shows the overview and captures some main idea of the thesis
* The Introduction sets the stage by outlining the research objectives and the significance of the study.
* Literature Review surveys existing work on recommendation models and negative sampling techniques, establishing theoretical foundation.
* Theoretical Foundation details the underlying concepts and models, including traditional models and GNNs models.
* Methodology describes experimental design, including dataset selection, model configurations, and evaluation metrics.
* Experiments section presents empirical results from the model selection phase on a small dataset, comparing performance across hybrid cascade framework and sampling strategies, followed by insights and interpretations.
* Conclusion and Future Work summarize the findings and propose directions for extending the research, such as testing larger datasets and refining hybrid models.

# LITERATURE REVIEW

This chapter traces the evolution of recommender systems, from foundational concepts like Case-Based Reasoning and KNN to the development of latent factor models (MF) and early neural approaches in section 2.1. then in the section 2.2 is the trend of graph appear with Graph-based. Finally section 2.3 show the gap of existing GNN-based.

## Evolution of Recommendation Systems

Recommender systems have evolved across multiple generations due to the increasing size, sparsity of data, and demand for improved personalized outcomes. The traditional approach was memory-based collaborative filtering, which relied on past joint user-item interactions. With the increase in the size and complexity of data, model-based methods were incorporated to improve scalability and generality. Recently, the use of graph-based neural networks has proven effective for learning sophisticated collaborative patterns.

In this section, the evolution from traditional memory approaches to current approaches using graphs in recommendation systems is reviewed, and their strengths and weaknesses are highlighted.

### Memmory base methods

Early recommender systems, typically illustrated in Machine Learning books, essentially focused upon memory-based collaborative filtering techniques such as Case-Based Reasoning (CBR) or k-Nearest Neighbors (kNN). They functioned upon the historical interactions associated with users and items and provided suggestions using those patterns of similarity without requiring any learning.

Case-Based Reasoning (CBR) views recommendations or suggestion-making as the process of searching for and learning from previous cases to adapt their solutions to the recommending situation. In this approach, a case consists of the profile of the user, the item chosen for the interaction, and the effect or the result of the interaction. CBR is considered very easy to understand and flexible since the explanations for the recommendations can easily cite previous instances.

The k-Nearest Neighbors (kNN) class-based recommendation algorithm recommends based on similarity measures between users or between items. In user-based kNN, recommendations to a user are inferred based on similar users’ behavior; in item-based kNN, recommendations to a user include items that resemble previously favored items purchased by the same user. Examples of similarity measures include cosine similarity or Pearson Correlation Coefficients techniques. These techniques work quite easily, which contributed to them seeing widespread adoption within early commercial recommendation systems.

However, CBR and kNN face several critical drawbacks. First, both of them have to calculate similarity on large interaction matrices. This can potentially make them slower for large numbers of users and items. Additionally, both CBR and kNN face sparsity in data since in practice, most of the users interact with just a few items. However, memory-based approaches lack generalization because their predictions depend on interaction data and lack flexibility when it comes to gaining insight into user preferences.

Issues arose, leading to the transition from traditional memory-based techniques to model-based techniques, including matrix factorization and neural collaborative filtering, aiming to learn an efficient low-dimensional latent factor representation of users and items.

### Latent factor models

To address the scalability issues and sparsity of memory-based approaches, Latent Factor Models, particularly the Matrix Factorization (MF) technique, were introduced. MF involves mapping users and items to a joint low-dimensional Latent Space and learning preferences by performing the dot product of the corresponding Latent Vectors.

Bayesian Personalized Ranking (BPR) is a variant of MF that focuses on implicit feedback and incorporates a ranking task. In BPR, it is assumed that the interactions indicate a preference of the user between observed and missing items, and such an assumption is commonly adopted in recommendation learning.

Bayesian Personalized Ranking Though the latent factor model performs well, it uses linear interactions and handles relationships between users and items independently, neglecting collaborative higher-order relationships. This situation triggers the need for more complex or expressive models, exploring higher-order interaction relationships.

### Neural Collaborative Filtering

Neural Collaborative Filtering (NCF) extends matrix factorization by replacing the inner product with a neural network, enabling the modeling of nonlinear user–item interactions. By learning flexible interaction functions, NCF can capture more complex preference patterns than linear MF.

However, NCF still models interactions in isolation and does not explicitly encode the relational structure among users and items. As a result, it remains limited in exploiting collaborative signals beyond direct user–item pairs.

## Graph-based recommendation models

### Graph base of recommendation data

To address the limitations of pairwise interaction models, recent research has formulated recommendation as a graph learning problem. In this paradigm, users and items are represented as nodes in a bipartite graph, and edges correspond to observed interactions.

Graph Neural Networks (GNNs) enable information propagation across multiple hops, allowing models to capture **high-order connectivity** such as user–item–user and item–user–item relationships (6). This capability makes GNNs particularly suitable for collaborative filtering tasks.

### Graph Convolutional Networks (GCN)

Early attempt and the popular one, Graph Convolutional Networks recommend item by aggregating information from neighboring nodes. In recommendation tasks, GCNs enable users to receive signals from items they interacted with and from other users indirectly connected through shared items.

Despite their success, GCNs were design for node classification tasks with rich node (14). Sparsity, a real world problems, is an problem GCN need to address.

### Neural Graph Collaborative Filtering (NGCF)

NGCF is the pioneering using architecture from GCN to extends graph-based CF by explicitly modeling feature interactions during message passing. It integrates non-linear transformations and feature concatenation to capture richer semantics (13).

While NGCF effectively captures high-order collaborative signals, its architectural complexity significantly increases computational cost and the risk of overfitting, particularly in sparse implicit-feedback datasets.

### Light Graph Convolution Netwok (LightGCN)

Understand the limitation of NGCF, LightGCN was proposed to simplify graph-based collaborative filtering by removing unnecessary components such as feature transformations and non-linear activations (4). It focuses solely on neighborhood aggregation, arguing that these simplified operations are sufficient for recommendation tasks.

Ablation empirical studies have shown that LightGCN achieves competitive or superior performance compared to more complex GNN-based models while maintaining high efficiency. As a result, LightGCN has become a widely adopted backbone for graph-based recommender systems. As this thesis, I also select LightGCN as a backbone model of the hybrid casaded recommendation framework due to it stil has several limitations need to improve in Section 2.3.

## Limitation of existing GNN-based model

Graph Neural Network (GNN)-based or LightGCN model particularly have achieved strong performance by modeling user–item interactions as graphs and leveraging multi-hop message passing to capture high-order collaborative signals. Models such as NGCF and LightGCN have become popular backbones due to their effectiveness in sparse recommendation scenarios. However, despite these advantages, GNN-based recommenders suffer from several fundamental limitations when applied to real-world implicit-feedback data.

### Treating all observed interaction as positive signals

In GNN-based recommenders, one crucial assumption is that each observed user-item interaction expresses positive preference. However, this holds untrue in the unfair world data manifested by implicit feedback. Clicking or short views are noisy and ambiguous since the user may click out of curiosity, due to exposure bias, misleading thumbnail images, or simply by mistake, and not necessarily for the actual item they prefer.

However, such systems consider all of them positive edges in the user-item graph. Hence, weak signals such as a click and strong ones such as a like, share, and follow receive equal consideration. Mixing the above may generate some noise during training, which results in learning a representation that is not really a reflection of user preference.

### Noise increases as the graph spreads information.

A CF model examines the interactions individually, while GNN diffuses the information over the interaction graph. It aids in identifying the higher-order patterns, but at the same time, the noise gets amplified when the interaction graph consists of noisy interactions.

Many of these are simple clicks with no usage following, and these ambiguous relationships can be modeled by assuming that if these are treated as positive edges, the effect will propagate to the neighboring users and items in the graph convolution. This might cause the system to develop popularity or exposure rather than preference-related behaviors because the model might tend to recommend more of those items that the users liked or saw but not the ones they actually liked or cared for.

### Limitations of negative samping in implicit feedback

A major flaw is present in the mechanism for negative feedback. This is because most of the existing GNN-based recommendation systems use the approach of random negative sampling, meaning they believe unseen pairs to be negative. This is not the case since the user may not favor the product if they have not seen it.

As a result is that there are false negatives, which are things that a given user would like if exposed to, and this provides unclear and noisy feedback. Some systems do record interaction information, and this could be a better negative example, though most GNN models do not rely on this and provide equal weights to everything that is not seen.

### Limitation of heterogeneous feedback and side information

Modern data contains strong signals and rich metadata, such as multiple interaction types (clicks, likes, shares, follows), user information, and item features. However, the common GNN-based recommenders make use of it in a shallow manner, often reducing it to a single binary graph.

Because there isn’t a good way to assess the strength of these messages, even when they’re weak, they can be more dominant in learning than strong ones. This will lead to learning representations that won’t be able to distinguish strong preference from weak interest.

### Implications for LightGCN

LightGCN reduces complex procedures within a graph structure-based CF algorithm by eliminating transforms, nonlinear parts, and considering only neighborhood aggregation. This promotes efficiency but also retains all the limitations mentioned above, as LightGCN lacks capabilities to filter out noisy interactions, distinguish strengths of feedback, and take reliable negative samples.

Consequently, in the case of noisy implicit feedback such as short video logs for which LightGCN is used, the model may learn embeddings that combine exposure with preference, leading to negative repercussions on ranking.

### Motivation for a new framework

These boundaries above illustrate that there is a discrepancy between what most GNN-based models recommend and the actual nature of large scale implicit feedback data. Instead of modifying how a GNN model is structured, it is asserted that it is actually how interaction data is processed and used during training that poses the greatest challenge. By decoupling inference of preferences, learning of representation, and ranking, the proposed approach remedies the shortcomings of GNN-based recommenders in noisy environments. This component-wise design transforms the objective from exposure modeling to intent modeling, thus being very effective in short video recommendation tasks such as Tenrec.

# THEORETICAL FOUNDATION

Based on the insights derived from the literature review, this chapter presents the proposed architecture of the GNN-base models through the base architecture GCN and its development model NGCF and LightGCN with the appearance of Random Forest to improve the performance.

The remainder of this chapter is organized as follows. Section 3.1 formally overview about GNN. Section 3.2 provides an overview of the proposed base architecture GNN. Sections 3.3 and 3.4 describe the developed of NGCF and LightGCN, respectively. Section 3.5 dicuss why Random Forrest could be used.

## Graph Neural Networks (GNNs)

Recommender systems have increasingly turned to **Graph Neural Networks (GNNs)** as a way to model complex relationships in user-item data. In a GNN-based recommender, we represent the interaction data as a **bipartite graph**: users and items are nodes, and each interaction (such as a click or rating) is an edge connecting a user to an item. This graph-based view enables the model to capture **high-order connectivity** – beyond direct user-item pairs – by propagating information through the graph structure. Essentially, GNNs treat a user’s neighborhood (the items they interacted with) and an item’s neighborhood (the users who interacted with it) as sources of signal for learning embeddings.

**Message passing principle:** GNN learn node embeddings through an iterative *message-passi*ng or neighbor *aggergation* process. In each GNN layer, every node receives “messages” from its immediate neighbors and updates its own representation based on an aggregation of those messages.

A diagram of a new method

AI-generated content may be incorrect.

Figure 1: example of GNN

For example, in one layer, a user node will aggregate information from all the item nodes it’s connected to, updating the user’s embedding to reflect those interactions. This typically involves: (1) each neighbor transforming its current embedding into a message, (2) the target node aggregating all incoming neighbor messages (embedding propagation in Figure 1), and (3) the target node updating its embedding with the aggregated message (Embedding propagation update with the node follow the arrow). By stacking multiple GNN layers, a node can gather information from neighbors-of-neighbors (two hops away), three hops away, and so on. In a recommendation context, this means:

* **1-hop neighbors:** A user directly connected to the items they interacted with (direct preferences).
* **2-hop neighbors:** Those items connect to other users, so the user indirectly receives information about *other users* with similar item tastes.
* **3-hop neighbors:** The original user can even get signals from items that those *other users* liked, capturing a form of “friends of friends” preference.

Each additional layer allows the model to incorporate broader collaborative patterns (e.g., *“users who liked item X also liked Y”* through 2-hop connections). This propagation of information helps combat data sparsity by letting a user’s representation be influenced by related users/items even if direct interactions are few. As a result, GNN-based models can learn rich embeddings that encode the network of relationships in the data.

In summary, GNNs bring an expressive, graph-based approach to representation learning in recommender systems. By iteratively aggregating information from connected nodes, they create embeddings that encode multi-hop collaborative signals. This enables recommendations that are more accurate and personalized, as the model can discover latent connections (for example, inferring that a user might like a movie because other users with similar watch histories liked that movie). In the next section, we focus on a specific GNN architecture – GCN – which is known as the most popular applied architecture for graph learning.

## Graph Convolution Network (GCN)

### Overview

Graph Convolutional Network (GCN) stand out as the most widely applied architecture for graph learning. GCNs work in a similar with CNNs, it apply convolutions directly on the graph, aggregating information from a node's local neighborhood.

In a GCN layer, the aggregation step is typically a normalized sum or average of the neighbor embeddings. The update then applies a linear transformation (a shared weight matrix W) and a non-linear activation function (e.g., ReLU) to the aggregated vector. This design is highly scalable and effective for learning node embeddings that capture both feature and structural information. GCNs are particularly well-suited for recommendation tasks and form the architectural basis for more specialized models like NGCF and LightGCN.

### Architecture

**Layer-wise propagation rule**, each GCN layer follow:

Where *H(0) = X*, weight W*(l),* and activation (e.g., RELU) (6).

**The two layers GCN** employed in node classification tasks is given by:

Where SoftMax outputs label probabilities per node (6).

### Pseudocode

A close-up of a white sheet

AI-generated content may be incorrect.

Algorithm 1: Graph Convolution Network (GCN).

The Graph Convolutional Network (GCN) algorithm generates recommendations by learning embeddings from a user-item bipartite graph G, where nodes are users and items, and edges represent interactions like ratings. As shown in the algorithm, initial node embeddings H0 are updated over L layers via graph convolution, which aggregates neighbor information (e.g., items a user rated) using the normalized adjacency matrix Ā, followed by a transformation W(l) and activation σ. This captures multi-hop relationships. The final embeddings produce interaction scores Ȓ through the dot product of user and item embeddings, predicting preference likelihoods. High-scoring items are recommended, leveraging collaborative filtering patterns.

Kipf & Welling’s GCN offers a theoretically motivated and practical framework for semi-supervised learning on graphs (6). Despite limitations, it provides a strong foundation for further advances like NGCF, which models be analyzed in the next section.

## Neural Graph Collaborative Filtering (NGCF)

Neural Graph Collaborative Filtering was a pioneering framework that adapted the GCN architecture specifically for the recommendation task. Its central argument is that traditional models like MF fail because the collaborative signal, which is latent in the user-item graph structure, is not explicitly encoded in the embedding process.

NGCF solves this by "exploiting the user-item graph structure by propagating embeddings on it". It defines a specialized embedding propagation layer that refines user and item embeddings. This layer consists of three key operations:

1. **Embedding Propagation:** Creates messages by considering not only the neighbor's embedding but also the interaction between the source and neighbor node.
2. **Neighborhood Aggregation:** Gathers messages from all neighbors in the bipartite graph.
3. **Update:** Combines the aggregated message with the node's self-embedding from the previous layer, and passes the result through a Linear transformation (with weight matrices) and a *ReLU* non-linear activation function.

By stacking *L* of these layers, NGCF explicitly models high-order connectivity, capturing the collaborative filtering signal at *L*-hops of distance. The final embedding for a user (or item) is created by concatenating the embeddings learned at all *L* layers. While powerful, this design is also complex, involving multiple large weight matrices for feature transformation and non-linear activations at every layer.

### Architecture

The proposed NGCF model architecture consist of three components in the framework (Figure 2): (1) an embedding layer that offers and initialization of user embeddings and item embeddings; (2) multiple embedding propagation layers that refine the embeddings by injecting high order connectivity relations; and (3) the prediction layer that aggregates the refined embeddings from different propagation layers and outputs the affinity score of a user-item pair

A diagram of a block diagram

AI-generated content may be incorrect.

Figure 2: An illustration of NGCF model architecture (the arrowed lines present the flow of information). The representations of user u1 (left) and item i4 (right) are defined with multiple embedding propagation layers, whose outputs are concatenated to make the.

**Embedding Layer:**

NGCF begins by initializing embedding vectors for both users and items. These embeddings are stored in a lookup table and act as the starting point for further refinement. we describe a user u (an item i) with an embedding vector eu ∈ Rd (ei ∈ Rd),, where d denotes the embedding size. This can be seen as building a parameter matrix as an embedding look-up table:

A black text with a white background

AI-generated content may be incorrect.

**Embedding propagation layers:** NGCF introduces one or more propagation layers that per form message passing on the user–item graph. The two main operations in each layer are:

* Message Construction:

For each user–item edge, a message is formed using a combination of the item’s embedding and the element-wise product between the user and item embeddings. This bi-interaction term allows the model to quantify the mutual influence between connected nodes. Formally, one example formulation is:

A black text on a white background

AI-generated content may be incorrect.

Where *eu* and *ei* are the current embeddings, *W1* and *W2* are trainable weight matrices, and ⊙ denotes the element-wise product.

* Message Aggregation:

Each node aggregates the messages from all its neighbors (plus its own “self-connection”) and applies an activation function (typically LeakyReLU) to produce updated embeddings. The aggregation is performed recursively across layers. We define as:

A black text on a white background

AI-generated content may be incorrect.

**Final prediction layer:** After propagating through *L* layers, NGCF concatenates embeddings from each layer (including the initial embeddings) to obtain a final, enriched representation. The predicted preference score between a user u and an item i is computed via an inner product:



### Pseudocode

A white paper with black text

AI-generated content may be incorrect.

Algorithm 2: Neural Graph Collaborative Filtering (NGCF).

The Neural Graph Collaborative Filtering (NGCF) algorithm refines collaborative filtering using a user-item bipartite graph *G*. Starting with initial embeddings *H(0)*, it updates embed dings over L layers via graph convolution, aggregating neighbor embeddings with the normalized adjacency matrix *Ā* and retaining self-connections via *H(l−1)W(l)*. This captures high-order connectivity (e.g., shared preferences). Final embeddings produce interaction scores *Ȓ* through dot products, recommending items with strong collaborative signals.

## Light Graph Convolutional Networks (LightGCN)

The LightGCN model is a streamlined graph convolution model tailored for collaborative filtering. From the ablation studies on NGCF, the authors surprising discovery the two most “standard” components of a GCN – the feature transformation matrices (*W*) and the non-linear activation functions (*RELU)* – contribute little to no performance in the collaborative filtering task. In fact, they often add unnecessary complexity, increase training difficulty, and degrade performance. (4)

### Architecture

A diagram of a light graph

AI-generated content may be incorrect.

Figure 2: An illustration of LightGCN model architecture.

From the Figure 3 LGC only the normalized sum of neighbor embeddings is performed towards next layer; other operations like self-connection, feature transformation, and nonlinear activation are all removed, which largely simplifies GCNs. In layer combination, it sum over the embeddings at each layer to obtain the final representations.

#### Light Graph Convolution (LGC)

In LightGCN, we adopt the simple weighted sum aggregator and abandon the use of feature transformation and nonlinear activation. The graph convolution operation in LightGCN is defined as:

A mathematical equation with black text

AI-generated content may be incorrect.

It is worth nothing that in LightGCN, we aggregate only the connected neighbors and do not integrate the target node itself

#### Layer combination and model prediction

In LightGCN, the only trainable model parameters are the embeddings at the 0-th layer, i.e., for all users and for all items. When they are given, the embeddings at higher layers can be computed via LGC defined in equation above. After K layers LGC, we further combine the embeddings obtained at each layer to form the final representation of a user (an item):

According to paper (4), setting *αk* uniformly as leads to good performance in general, and they do not design a special component to optimize *αk*. The reasons that paper () performs layer combination to get final representations are three-fold:

* With the increasing number of layers, the embeddings will be oversmoothed. Thus, simply using the last layer is problematic.
* The embeddings at different layers capture different semantics. • Combining embeddings at different layers with weighted sum captures the effect of graph con volution with self-connections, an important trick in GCNs

The model prediction is defined as the inner product of user and item as same as the equation prediction in NGCF model, which is used as the ranking score for recommendation generation. Matrix form Matrix form is provided to facilitate implementation and discussion with existing models. Let the user-item interaction matrix be R ∈ RM×N where M and N denote the number of users and items, respectively, and each entry Rui is 1 if u has interacted with item I otherwise 0. We then obtain the adjacency matrix of the user-item graph as:

Let the 0-th layer embedding matrix be , where T is the embedding size. Then we can obtain the matrix equivalent form of LGC as:



Lastly, we get the final embedding matrix used for model prediction as:



Where is the symmetrically normalized matrix.

### Pseudocode

**A screenshot of a computer program

AI-generated content may be incorrect.**

Algorithm 3: Light Graph Convolution Network (LightGCN).

The Light Graph Convolution Network algorithm simplifies GCN for efficient collaborative filtering. Using a user-item bipartite graph G and initial embeddings H(0), it performs simplified graph convolution over *L* layers, updating embeddings by averaging neighbors’ embeddings via Ā. The average embeddings are computed as the means of all layer outputs, including H(0). Interaction scores Ȓ are derived from the dot product of final user and item embeddings, recommending high-scoring items with minimal computational overhead.

LightGCN’s design deliberately omits the overhead of feature transform layers, focusing the model capacity on learning good embeddings and propagating them through the graph. Empirically, this has been shown to **capture long-range collaborative signals** effectively: by using multi-hop neighbors, LightGCN can recommend items that are not immediately obvious from one-hop interactions (for example, catering to a user’s interest that is only apparent after considering similar users’ behavior). The simplicity of LightGCN makes it a **strong baseline** and a starting point for many improvements in graph-based recommenders. However, as we discuss next, certain assumptions in how LightGCN (and similar models) are trained can become problematic in noisy, real-world scenarios.

Despite their effectiveness, LightGCN and similar GNN recommenders make simplifying assumptions about the data that do not perfectly hold in reality. They assume *all observed interactions = positive*, *all unobserved = negative*, and they lack a mechanism to treat some “positives” as less positive than others. In noisy environments (like a short-video feed or news feed), these assumptions lead to mis-aligned training signals. The model may struggle to learn truly discriminative embeddings because it’s being fed a mix of genuine preferences and incidental interactions all labeled the same. These limitations motivate the need for a more *nuanced approach* to training graph-based recommender models – one that can filter out noise and incorporate more reliable signals. In the next section, we discuss our approach to address these issues by introducing an explicit *denoising* step using a Random Forest model before feeding data into LightGCN.

## Motivation for Denoising with Random Forest

Given the challenges above, a key question arises: *How can we provide LightGCN (or any GNN model) with better input data that truly reflects user preferences?* My approach is to perform a **denoising step** on the raw interaction data using a **Random Forest classifier** as a filtering mechanism. The idea is to distinguish between **strong vs. weak user feedback** by leveraging the variety of user behaviors available (clicks, likes, shares, follows, etc.), and only treat the high-quality interactions as positives for the graph model. This section explains why this is beneficial and how we implement it.

Random Forest (RF) is an ensemble of decision trees that is known for its robustness and interpretability. It can model non-linear relationships and handle heterogeneous features without heavy tuning. In our case, we have a variety of features that could help predict whether a given interaction (a click) was likely positive or negative. These features can include properties of the interaction or content: e.g., user’s watch time, whether the user hovered or repeatedly watched, the item’s category, and user’s attributes (age, gender), etc. An RF is well-suited to handle this mix of features. As noted in our design, *“a Random Forest classifier is selected due to its robustness, interpretability, and ability to model non-linear relationships without extensive hyperparameter tuning.”*. In contrast to a complex deep model, an RF can be trained relatively quickly and tends not to overfit on noisy data due to the averaging of many trees. Importantly, it provides feature importance metrics, giving us insight into what factors indicate true preference

### Pseudocode

Random Forest is used twice in two different task in cascaded framework with the basic flow: Raw interactions → RF denoise → LightGCN candidate retrieval → RF rerank → final recommendations. In this section, I explain about two tasks of RF.

A screenshot of a computer program

AI-generated content may be incorrect.

Algorithm 4: Random Forest Interaction denoising.

This Random Forest acts like a gatekeeper. It learns patterns that look “real” (from like/share/follow) and then checks which clicks match that pattern. Instead of assuming every click is positive, you keep only the clicks that look more trustworthy. That way LightGCN is not forced to learn from a huge amount of random or exposure-driven clicks.

A screenshot of a computer program

AI-generated content may be incorrect.

Algorithm 5: Random Forest reranking.

LightGCN is good at pulling a decent candidate set fast, but its ranking signal is limited (mostly embedding similarity). The reranking Random Forest learns a richer decision rule: it can use the LightGCN score plus extra clues (item popularity, freshness, user activity level, category match, etc.). In practice, this step often boosts NDCG@K because it fixes the order of the Top-K list.

# METHODOLOGY

This chapter focuses on the proposed **Hybrid Cascaded Recommendation Framework** and is organized as follows. Section 4.1 make an overview about the framework. Section 4.2 introduces the Tenrec dataset and highlights its characteristics and challenges. Section 4.3 details the overall pipeline of our proposed method, including data preprocessing with semi-supervised denoising, graph-based embedding learning using LightGCN, and the ensemble re-ranking stage. I emphasize the novelty of each component compared to baseline approaches. In section 4.4, I defines the evaluation metrics used to assess recommendation performance. Finally, Section 4.5 show the friendly website for demo purpose.

## Research design

This study adopts a **hybrid experimental research design** that combines data-centric preprocessing, graph-based representation learning, and ensemble-based ranking. Instead of proposing a new graph neural architecture, the methodology focuses on **revisiting the training assumptions of GNN-based recommender systems under noisy implicit feedback**.

The overall design follows three principles:

1. **Noise-aware learning**: Explicitly distinguish strong, weak, and ambiguous user feedback instead of treating all interactions equally.
2. **Representation–ranking decoupling**: Separate latent preference learning (LightGCN) from final decision making (tree-based ranking).
3. **Realistic evaluation**: Leverage exposure-based negative signals provided by Tenrec to avoid biased sampling.

Accordingly, the proposed framework consists of three sequential stages:

1. Semi-supervised data preprocessing and denoising
2. Graph-based representation learning using LightGCN
3. Ensemble learning for final re-ranking

Figure 4.1 illustrates the complete pipeline of the proposed framework.

A diagram of a graph

AI-generated content may be incorrect.

Figure 3: Framework pipeline.

## Tenrec dataset

### Dataset description

Tenrec dataset is a multipurpose benchmark with 5 million users and 140 million interactions across four scenarios (clicks, likes, shares, negative feedback) (2)

A diagram of a social media network

AI-generated content may be incorrect.

Figure 4: Data sources of Tenrec (Source: (2)).

Tenrec is a dataset suite developed for multiple recommendation tasks, collected from two different feeds recommendation platforms of Tencent[[1]](#footnote-1), namely, QQ BOW (QB) and QQ KAN (QK). An item in QK/QB can either be a news article or a video. Note that the article and video recommendation models are trained separately with different neural networks and features. Thus, I think that Tenrec is composed of user feedback from four scenarios in total, namely, QK-video, QK-article, QB-video, and QB-article.

In this thesis, the experiments focus on the **QK-video scenario**, which best reflects short-video recommendation environments characterized by rapid content consumption and noisy implicit feedback.

### Data schema

The format of each instance in QK/QB-video is:

{user ID, item ID, click, like, share, follow, video\_category, watching\_times, gender, age}.

Where:

* click, like, share, follow are binary interaction signals
* watch\_times records the number of viewing behaviors for the video
* category denotes the video topic
* gender and age are anonymized user attributes (age is binned into 10-year intervals)

All user and item identifiers are anonymized to preserve privacy. Timestamp information has been removed; however, interactions are provided in chronological order.

### Data distribution

A graph with a curve

AI-generated content may be incorrect.

Figure 5: Item distribution. (a) and (b) are item popularity plotted in raw and log-log scales; (c) is the item session length distribution.

Tenrec exhibits several challenging characteristics:

Figure 7 (a) and (b) show the item popularity of QK-video in the term of the clicking behaviors. Clearly, the item popularity follows a typical long-tail distribution, which shows the dramatic gap between the interactions. Making it difficult for models to prefer to recommend popular items without the least ones. (c) shows the session length distribution, where the number of sessions with length in [0-20] accounts for 53% of all sessions. Indicating low user engagement depth and high uncertainty. Similar distributions can be observed on the other three datasets, which are thus simply omitted. Therefore, Tenrec is an imbalance dataset, with some video accounts for majority of viewers and most users are too lazy to interact with websites leading to sparse data.

### Data analysis

Pre-processing Tenrec for GNN-based recommender systems presents several non-trivial challenges, which must be addressed carefully to ensure model performance and evaluation validity.

**Large-scale data processing**: Tenrec contains approximately 5 million users and 140 million interactions across domains (e.g., QK-video alone records 142 million clicks from 5 million users) [2]. Executing operations such as sorting, filtering, and negative sampling on this scale can exceed memory limits. One recommended solution is to utilize chunk-based CSV readers or fast in-memory Data Frame engines like polars or employ columnar formats such as Apache Parquet for efficient I/O and scalable data manipulation.

**Multi-behavior interaction:** modeling records include multiple positive feedback signals (e.g., click, like, share, follow) as well as \*\*true negative instances\*\* (exposures without any action), instead of implicit negatives [2]. Treating all behaviors uniformly ignores the varied intensities—e.g., a “share” may be semantically stronger than a “click”. To properly capture these differences, it is necessary to encode multi-channel feedback into edge-type aware graphs or assign distinct weights per behavior during training.

**Negative feedback as ground truth:** In addition to explicit interactions, Tenrec comprises \*\*non-interacted but exposed items\*\*, enabling rigorous evaluation of recommender models under binary classification (not just ranking). Effective modeling and sampling of negatives from exposures rather than assuming all unseen items are negative is key to avoiding imbalanced training and biased performance assessments. If bad in sampling, it will make noise in data and fluctuate metrics.

**Metadata encoding for GNN usage**: Aside from IDs and behaviors, Tenrec also includes side information—user age bins, gender, and item categories. To integrate them into GNN models, the metadata must be label-encoded or embedded and potentially combined with ID embeddings. Handling high-cardinality categorical features efficiently is important to avoid overfitting and ensure graph propagation accounts for side-information.

The raw interaction logs in the QB-video and QK-video dataset contain many inconsistent and low-quality signals that can distort the learning dynamics of graph-based recommenders. To address this, the pre-processing stage follows a structured multi-step pipeline designed to isolate meaningful behavioral patterns, relabel ambiguous interactions, and prepare both the positive graph and the negative sampling pool for downstream modeling. The process consists of four major steps, described below.

## Data Preprocessing and Semi-Supervised Denoising

A diagram of a work flow

AI-generated content may be incorrect.

Figure 6: Pre-processing pipeline.

This phase focuses on enhancing data quality by classifying user interactions into distinct categories to mitigate noise and ambiguity. Standard GNN-based recommender systems assume that all observed interactions are positive and all unobserved interactions are negative. This assumption is violated in Tenrec, where click and watch behaviors are often ambiguous. To address this issue, we introduce a **three-tier feedback stratification strategy** combined with semi-supervised pseudo-labeling.

### Three-Tier Feedback Stratification

The first step is to categorize each user–video interaction into one of three behavioral tiers. This tiering serves as the foundation for distinguishing reliable preference indicators from noise.

* **Strong Positive Tier:** Interactions that include explicit engagement actions - such as *like*, *share*, or *follow* - are placed in this tier. These represent the most trustworthy signals of user preference and require no further interpretation.
* **Strong Negative Tier:** Interactions where the user watches a video for a *very short duration* (for example, below a small threshold) and does *not* engage in any action are treated as strong negative evidence. These events typically reflect scrolling behavior, indicating clear disinterest.
* **Ambiguous Tier:** All remaining interactions are placed in this tier. These include long views without explicit engagement, short views that may or may not be accidental, and mixed behavioral patterns. Because these signals are inherently uncertain, they are reserved for model-based relabeling.

This tiering step separates high-certainty behavioral signals from low-certainty ones, enabling targeted processing in the next stages.

After the tiering step, a small supervised model is trained solely on the high-confidence tiers. The goal is to use these clean labels to infer preference for ambiguous cases.

* **Training Data**: Only interactions from the Strong Positive (label 1) and Strong Negative (label 0) tiers are used.
* **Input Features**: Metadata and context features such as video category, user demographic fields (age group, gender), and coarse behavioral statistics (e.g., viewing time) are included.
* **Model Choice**: A Random Forest classifier is selected due to its robustness, interpretability, and ability to model non-linear relationships without extensive hyperparameter tuning.

This teacher model learns a behavioral boundary that generalizes beyond explicit engagement signals, allowing it to infer user preference under uncertainty.

### Semi-Supervised Pseudo-Labeling

A **Random Forest classifier** is trained as a teacher model using Tier 1 (label = 1) and Tier 3 (label = 0) samples. The model uses metadata and contextual features such as:

* Video category
* User age and gender
* Watch time statistics

The trained classifier predicts preference probabilities for Tier 2 samples:

* Samples with probability > 0.6 are reclassified as **weak positives**
* Samples with probability < 0.4 are reclassified as **weak negatives**
* Samples in the intermediate range are discarded to avoid introducing noise

This process significantly reduces label ambiguity and constructs cleaner training data.

## Graph-Based Representation Learning with LightGCN

A diagram of a process

AI-generated content may be incorrect.

Figure 7: Graph Neural Network pipeline.

### Graph Construction

Using the refined positive set, a user–item bipartite graph is constructed. Nodes represent users and items, while edges correspond to positive interactions.

The adjacency matrix is normalized to obtain , which is used for message passing.

### Hybrid Negative Sampling Strategy

To train LightGCN effectively, a **hybrid negative sampling strategy** is adopted:

* **50% True Negatives (Hard Negatives)**: Sampled from the Final Negative Dictionary, representing items the user explicitly ignored.
* **50% Random Negatives (Soft Negatives)**: Sampled uniformly from the item pool excluding known positives, ensuring generalization.

Training triplets are constructed accordingly.

### LightGCN Architecture

The main idea of LightGCN is learning embeddings through linear message passing:

* **Initialization**  
  User and item embeddings are randomly initialized.
* **Propagation**  
  Embeddings are updated over layers:
* **Aggregation**  
  Final embeddings are obtained by mean pooling:

### Optimization Objective

The model is trained using **Bayesian Personalized Ranking (BPR) loss**, which encourages positive items to be ranked higher than negative items. Optimization is performed using Adam until convergence.

## Ensemble Learning for Re-Ranking

### Feature Engineering

For each user–item pair, a feature vector is constructed by concatenating:

* LightGCN embeddings and dot-product scores
* User metadata (age, gender)
* Item statistics (popularity, average watch time, category)

All item statistics are computed from the training set only to avoid data leakage.

### Random Forest Re-Ranker

A Random Forest model is trained to predict the probability that a user prefers an item. During the recommendation phase, we can use LightGCN to generate an initial list of candidate items for each user (for example, the top 100 items), then compute the feature vectors for these user–item pairs and finally use the RF to compute a refined score *pu,i* for each candidate. The candidates are then sorted by this score, and the top-K items are recommended to the user. Thepredicted probability *pu,i* essentially acts as the final ensemble score for ranking. In our offline experiments, we simulate this by simply computing RF scores for all user-item pairs in the test set (which is feasible for evaluation) and ranking accordingly. The predicted probability is used to re-rank candidate items and generate the final Top- recommendation list.

## Evaluation metrics

There are several standard recommendation accuracy measures on the held-out test set. These include Precision@K, Recall@K, and Normalized Discounted Cumulative Gain (NDCG@K), and how well the model ranks relevant items for each user. Higher precision/recall and NDCG indicate better top-K ranking performance

### Precision@K

Precision@K measures the proportion of recommended items in the top-K set that are relevant to the user. It is defined as:

A higher Precision@K indicates that a larger percentage of the recommended items are actually relevant. This metric is critical for user experience, as it reflects the likelihood that a user will find value in the suggestions, though it may penalize models that recommend too many items, including some irrelevant ones.

### Recall@K

Recall@K assesses the proportion of all relevant items that are successfully recommended in the top-K list. It is calculated as:

This metric reflects the system’s ability to retrieve all relevant items for a user. A higher Recall@20 shows that the model successfully identifies a larger fraction of relevant items within the top 20. This metric is important for ensuring the model doesn’t miss key items, though it doesn’t account for their order, making it complementary to NDCG@20.

### NDCG@K (Normalized Discounted Cumulative Gain at K)

NDCG@K evaluates the quality of the ranking of recommended items, emphasizing the position of relevant items in the list. It accounts for the position of relevant items by applying a logarithmic discount: the higher the rank, the less the contribution to the score. It is computed as:

Where DCG@K (Discounted Cumulative Gain) is:

And IDCG@K is the ideal DCG value, representing the best possible ranking of relevant items. NDCG@K values range from 0 to 1. A higher NDCG@K indicates better ranking performance, meaning the model effectively prioritizes relevant items. For example, if a user’s top preferences are ranked within the first 20 recommendations, the score reflects this accuracy, making it crucial for user satisfaction in real-world systems.

## Demo website

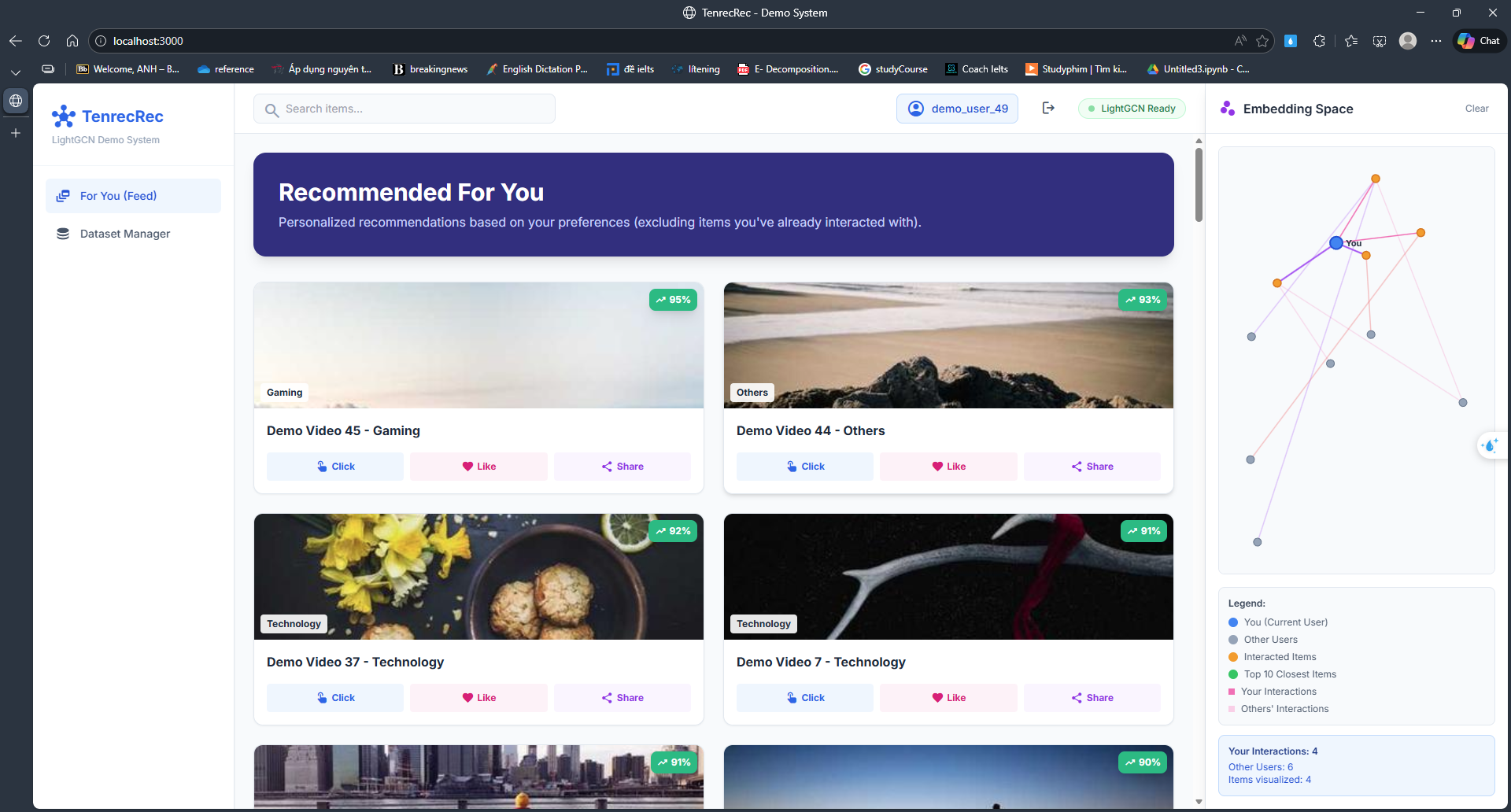


Figure 8: Demo website

The Figure 8 is a website I build with NodeJS and MongoDB to test the framework running on the industrial pipeline. It is a friendly user interaction with two main parts:

* **The recommendation area:**

This area show the list of top item that framework recommend for user, each item has the score compute by the model. Because the privacy of dataset I can not access the source of real item and user, images are for illustration purposes only.

* **The present graph area:**

This area show how the Graph Neural Network working and present data in graph. It have three object: the big blue circle is the user need to recommend, the orange circles are other users in the network and the gray ones are items.

Each time user interacts with item in the recommendation area, the graph will update information and connect user with that item, and so on.

# EXPERIMENTS AND EVALUATION

This chapter present the experiment evaluation the hybrid casaded framwork and discuss about result observe. It invole three main part, Section 5.1 make experiment to find out the backbone model for framework. In section 5.2, It analyzes the impact of interaction denoising and discusses how data refinement affects graph structure and model behavior. Finally Section 5.3 create the ablation study to evaluate the contribution of three important modules.

## Backbone model

### Impact of GNN layers

One of the key motivations behind graph-based recommender systems is their ability to model high-order user–item connectivity through multi-hop message propagation. To empirically examine this effect, we conduct an experiment on the number of GNN layers for NGCF and LightGCN.

Table 1: GNNs high order (layers).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **1 layer** | **2 layers** | **3 layers** | **4 layers** |
| **NGCF** | 0.0423 | 0.0425 | 0.0426 | 0.0425 |
| **LightGCN** | 0.0658 | 0.061768 | 0.0719 | 0.0702 |

However, when the number of layers continues to increase, performance begins to degrade. This phenomenon is commonly referred to as the *over-smoothing problem*, where repeated aggregation causes node representations to become overly similar. In our experiments, NGCF achieves its best performance at three layers, while LightGCN peaks at two to three layers, after which performance declines.

These results suggest that carefully selecting the number of GNN layers is crucial, and that shallow but sufficiently expressive architectures are preferable for recommendation tasks.

### Backbone comparisons

To futher justify the backbone model, I let LightGCN compare with two varient GNNs and two traditional models to have the general perspective. Table 5 report the performance and relative training time.

Table 2:Model comparisons.

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **NDCG** | **Recall** | **Time** |
| MF | 0.0467 | 0.0936 | **x0.75** |
| NCF | 0.0405 | 0.0757 | X3 |
| GCN | 0.0062 | 0.0129 | x5 |
| NGCF | 0.0426 | 0.0871 | x8 |
| LightGCN | **0.0618** | **0.123** | x1 |

A graph of training loss and training loss

AI-generated content may be incorrect.

Figure 9: Training loss and validation metric of MF, with training loss and valid NDCG in in the left and all validation metrics in the right.

**Traditional models:** Without message passing or aggregation steps and 1-hop make them reduce the huge training time compared with the GNNs. However, the NCF using the multiple neural network (MLP) and backpropagation) is more complex than the simplicity and direct sampling strategy of MF. From Figure 7, MF has ability to learn some patterns from the dataset. Overall, both models cannot learn latent and imply data in high order which reduces its performance.

**Light Graph Convolution Network (LightGCN):** Among all compared methods, LightGCN achieves the best trade-off between accuracy and efficiency. While GNN-based models generally outperform traditional matrix factorization approaches, their training costs vary significantly. NGCF, despite its expressive neural architecture, incurs substantial computational overhead due to nonlinear transformations and multiple feature projections.

In contrast, LightGCN removes unnecessary transformations and nonlinearities that are not beneficial for collaborative filtering. This design choice allows it to achieve higher Recall and NDCG scores than MF and NGCF, while maintaining training time comparable to simpler models. These results indicate that LightGCN is particularly well-suited for large-scale and noisy implicit-feedback datasets such as Tenrec.

A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

AI-generated content may be incorrect.

Figure 10: Training loss and evaluation metrics of LightGCN, with training loss and valid NDCG in the left and validation metrics on the right.

Training loss decreases from 250 to nearly 0 indicates the converge in training set (Figure 9, while Validation metrics experienced slightly increase in the first 5 epochs then it remains the same (early stop technique). It indicates that model still learning from implicit data from user (click is the dominent interaction compare with the strong interaction as like, share and follow). Overall, the performance is still low but brings the highest performance compared with the others. Therefore, LightGCN provides the best trade-off between accuracy and computational efficiency, making it a suitable backbone for further enhancement.

### Backbone Selection summary

Based on the above analysis, I select LightGCN as the backbone model for subsequent experiments due to its effectively captures high-order collaborative signals, avoids over-parameterization, and offers a favorable balance between recommendation accuracy and computational efficiency. These properties make LightGCN a strong foundation for further enhancements through data denoising and multi-stage ranking.

## Hybrid model with data denoising and true negative sampling

To further investigate the limitations of the baseline LightGCN and evaluate the benefits of integrating multi-stage data refinement and ensemble-based ranking, I developed a framework with Data Denoising, negtive sampling and reranking. This model combines semi-supervised relabeling, graph-based representation learning, and a non-linear ranking mechanism. The experimental results show that this hybrid architecture consistently outperforms the pure LightGCN baseline both in accuracy and stability of recommendation.

Another important driving force in the design is that it stems from recognizing that the QB-video dataset contains a significant amount of noisy interaction, which is mainly contributed by fast scrolling, accidental taps, and view events that are too short. In other words, if these noisy interactions are viewed as positive edges directly, Light-GCN is forced to spread noisy information on the graph and may produce distorted representations, which would depict mere exposure as preference.

To address this issue, the proposed framework applies interaction denoising as a preprocessing step to create the graph. The user-item interactions are divided into strong positives, strong negatives, and an ambiguous area with the dominant click-only interactions. A light Random Forest teacher model is used to predict the pseudo-labels of the ambiguous interactions after training with reliable samples. The interactions with high confidence are considered for learning the graph, and the others go to the filtered negative set or are removed altogether. Although the size of the interaction graph shrinks after this step, the resultant dataset represents the user intent more accurately.

Table 3: Comparison with framework.

|  |  |  |  |
| --- | --- | --- | --- |
| Model | NDCG | Recall | AUC |
| LightGCN | 0.0618 | 0.123 | **0**.056 |
| Denoise (no reranking) | 0.1042 | 0.3283 | 0.5578 |
| Full framework | **0.2108** | **0.4467** | **0.7114** |

As shown in Table 3, the denoise component by itself achieves a dramatic improvement over the baseline on Recall and AUC. This shows that the noisy interactions removed and the subsequent training of the LightGCN on the cleaned graph have resulted in more discriminative embeddings despite the reduction in the total number of interactions.

With this new reranking step, the Full Hybrid achieves the best possible scores in terms of NDCG and AUC metrics, indicating an improvement in ranking quality and preference. While there is a slight decrease in Recall compared to the variant focused solely on denoising, this is expected. Reranking has been trained on optimizing the rank within the top-N list as opposed to optimizing item coverage; this justifies the drop on Recall while still optimizing ranking metrics.

Notably, the improvement in the value of AUC suggests that the hybrid model is more capable than its competitor of distinguishing between relevant and non-relevant items in the search candidate set. This means that the system is capable of learning a more precise boundary or definition of user preferences even with a reduced set of suggested items. This is more preferable in real-world applications because most users are only exposed to the top-rated items rather than the entire set of search candidates.

Apart from the performance boost, it’s clear that the hybrid framework cuts down on costs when it comes to training too. The denoised graph has significantly fewer nodes, making it easier to reach convergence and optimize. There is less variance in the graphs for each epoch, indicating that it’s beneficial to learn on the compact and semantically sound interaction graph with LightGCN.In summary, the proposed framework does not merely improve ranking metrics but explicitly shifts the learning objective from modeling user exposure to modeling user preference. By filtering ambiguous interactions and prioritizing high-intent behaviors, the model learns embeddings that better reflect what users actually like, rather than what they accidentally interact with.

## Ablation study

In this section, I divide my framework into three main modules as module denoise module sampling, and module reranking. This help me could analyze the contribution of each module on the performance of base line. The main goal of this section is to confirm the following issues:

* Impact of the denoising module (noise filtering using Random Forrest)
* The effectiveness of true negative sampling strategy
* Contribution of the reranking step after learning embedding

Due to limited time and computational constraints, the full ablation experiments are still ongoing. However, based on early trends observed during partial training, it is expected that the denoising module improves signal quality by filtering out weak implicit interactions (e.g., short watch duration without follow/like). Further experiments will be conducted to confirm the contribution of each component

Table 4: Ablation experiments.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Denoise** | **True Negative** | **Sampler** | **Reranking** | M**eaning** | **Recall** | **NDCG** |
| A0 | 0 | 0 | uniform | 0 | LightGCN baseline | 0.317 | 0.107 |
| A1 | 0 | 0 | uniform | 1 | + Reranking | **0.326** | **0.150** |
| A2 | 1 | 0 | uniform | 0 | + Denoise | 0.255 | 0.081 |
| A3 | 1 | 1 | true\_neg | 0 | + True Negative | 0.013 | 0.012 |
| A4 | 1 | 1 | hybrid | 0 | + Denoise and true negative | 0.025 | 0.016 |
| A5 | 1 | 1 | hybrid | 1 | Full model | 0.024 | 0.11 |

### Baseline and Reranking

The LightGCN baseline (A0) achieve a Recall of 0.317 and an NDCG of 0.107 when trained and evaluated using click-based interactions. When the reranking modeule is added (A1), both metrics improve consistently, with 0.326 and 0.150 for NDCG and recall respectively.

This result indicates that reranking contribute positively even when underlying embeddings are learned from noisy data. Because it learns from side information, The Random Forest reranker integrates embeddings with metadata and behavior statistics instead of relies on a linear dot product scoring function of LightGCN. This explains the noticeable gain in NDCG, which emphasizes reranking quality rather than coverage. Therefore, random forest reranking has ability to improve the weakness of LightGCN in the imbalance real world dataset Tenrec.

### Impact of denoising under click-only evaluation

When denoising is applied without reranking (A2), performance drops significantly compared to the baseline. Recall decreases from 0.317 to 0.255, and NDCG drops from 0.107 to 0.081. This degradation becomes more severe when true-negative sampling is introduced (A3 and A4), where both Recall and NDCG fall to near-zero values.

This behavior can be explained by a semantic mismatch between training and evaluation signals. While the Filtering suppose that click just a weak positive in training set, the test set is click only. In other word, the model is optimized for preference recognition, while the evaluation protocol measure click recovery, leading to understand performance.

### True negative sampling

There are the drop performance sharply in setting A3, A4 and A5 highlighting the mismatch as I mention above. Since the clicks account for the majority of interactions in the dataset, they are treated as ground-truth positives during testing. In contrast, the true negative sampler is designed to identify interactions that are likely not preferred by users, even if a click has occurred. As a result, many interactions are counted as positives during evaluation are

This mismatch indicates that the model learning suboptimal on click-based protocol. Therefore, the observed performance decreases do not necessarily suppose that true negative sampling is ineffective, but rather that it is not well aligned with the evaluation object in this setting. Overall it is the way to develop in the future to address this issue.

### Full framework

The experiment of full framework bring the result is suboptimal compare with the base line due to the mismatch in two module denoise and sampling. Al though recall remains low, NDCG improves significantly compare to the denoising experiments (A3). This results indicate that module reranking still strength by using side information and a non linear ranking model, it is able to better the limited set of candidate items in the long tail dataset. Instead of focusing on retrieving many items, it emphasizes more relevant items at the top of the recommendation list.

### Key insight from study

The ablation results suppose several important insights:

* Reranking module consistently improves the quality in click-based evaluation. Especially in terms of NDCG, by integrating non-linear feature interactions that improve the weakness side of LightGCN.
* Denoising and true negative sampling module are suboptimal in the click-based evaluation of the baseline. However it does not ignore the design of cascaded framework, it show the way to develop in the future.

Overall, these result suggest that the proposed framework is better suited for scenarios where the evaluation target align with true user preference rather than raw interaction frequency. While the current

# CONCLUSION AND FUTURE WORK

## Conclusion

This thesis provides a framework for a recommender system. It addresses one of the main issues of GNN models—their inability to distinguish between correct user interest and noisy actions related to exposure.

Evaluation on the Tenrec QB-video dataset indicates the effectiveness of the smaller and cleaner interaction graph in improving ranking performance and distinguishing actual preferences from accidental behaviors. This hybrid approach outperformed the LightGCN-only baseline on the AUC and NDCG metrics, indicating the learned graph embeddings capture the actual preferences of users more accurately than the accidental patterns identified by the graph. This clearly indicates the significance of the quality of the data and the interpretation of the interactions for graph-based recommenders.

These findings confirm that data quality and interaction semantics play a critical role in graph-based recommender systems, and that combining denoising with graph learning and ensemble ranking provides a practical and effective solution for noisy real-world datasets.

## Future work

In the future, I tend to improve the framework in some cases. First, I try to make experiments in QK-video to evaluate the result on larger dataset. Second, developing the framework with new strategy to enhance the performance of framework to capture more preference of user in click-base data. Finally, incorporating temporal and sequential information is a promising direction. Users are usually attracted in short-video platforms, where interests change rapidly. Integrating time-aware graph modeling or sequential recommendation techniques could further enhance preference modeling.

LIST OF REFERENCES

1. S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, “Graph neural networks in recommender systems: A survey,” 2022.
2. G. Yuan, F. Yuan, Y. Li, B. Kong, S. Li, L. Chen, M. Yang, C. Yu, B. Hu, Z. Li, Y. Xu, and X. Qie, “Tenrec: A large-scale multipurpose benchmark dataset for recommender systems,” arXiv preprint arXiv:2210.10629, 2022
3. C. Gao, Y. Zheng, N. Li, Y. Li, Y. Qin, J. Piao, Y. Quan, J. Chang, D. Jin, X. He, and Y. Li, “A survey of graph neural networks for recommender systems: Challenges, methods, and directions,” 2023.
4. X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, “Lightgcn: Simplifying and powering graph convolution network for recommendation,” 2020.
5. P. Veliˇckovi´c, G. Cucurull, A. Casanova, A. Romero, P. Li`o, and Y. Bengio, “Graph attention networks,” 2018.
6. T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” 2017.
7. S. Zhang, L. Yao, A. Sun, and Y. Tay, “Deep learning based recommender system: A survey and new perspectives,” ACM Computing Surveys, vol. 52, p. 1–38, Feb. 2019.
8. H. liu, Wei Wang, Yihan Zhang, Renqian Gu, Yaqi Hao, “Neural matrix factorization recommendation for user preference prediction based on explicit and implicit feedback”, 2022.
9. Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and T. Chua, “Neural collaborative filtering”, 2017.
10. J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, “Graph neural networks: A review of methods and applications,” 2021.
11. D. Bacciu, F. Errica, A. Micheli, and M. Podda, “A gentle introduction to deep learning for graphs,” Neural Networks, vol. 129, p. 203–221, Sept. 2020.
12. Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” Computer, vol. 42, pp. 30–37, Aug. 2009.
13. X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, “Neural graph collaborative filtering,” in Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR ’19, p. 165–174, ACM, July 2019.
14. W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” 2018.
15. X. Wang, X. He, Y. Cao, M. Liu, and T.-S. Chua, “Kgat: Knowledge graph attention network for recommendation,” in Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining, KDD ’19, p. 950–958, ACM, July 2019.
16. Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A comprehensive survey on graph neural networks,” IEEE Transactions on Neural Networks and Learning Systems, vol. 32, p. 4–24, Jan. 2021.
17. J. Chen, J. Zhu, and L. Song, “Stochastic training of graph convolutional networks with variance reduction,” 2018. [17] P. Veliˇckovi´c, G. Cucurull, A. Casanova, A. Romero, P. Li`o, and Y. Bengio, “Graph attention networks,” 2018.

APENNDIX

1. **Requirement environment**

To implement and experiment with GNNs, two widely used libraries are PyTorch and PyTorch Geometric (PyG):

* PyTorch: An open-source deep learning library that provides flexible tools for building and training machine learning models.
* PyTorch Geometric (PyG): An extension library of PyTorch specifically designed for processing graph data and building GNN models.

**Installation steps:**

Install Pytorch

pip install torch

Install PyG dependencies

*pip install pyg\_lib torch\_scatter torch\_sparse torch\_cluster torch\_spline\_conv-f https://data.pyg.org/whl/torch-${VERSION}$+${CUDA}.html*

where $VERSION should be replaced by either 2.4.0 or 2.5.0 and $CUDA should be replaced by either CPU, cu118, cu121, or cu124 depending on your PyTorch installation

Install PyG

PyG is available for Python 3.9 to Python 3.12. From PyG 2.3 onwards, you can install and use PyG without any external library required except for PyTorch. For this, simply run:

*pip install torch\_geometric*

To have more details you can access the link[[2]](#footnote-2)

1. [Tencent](https://www.tencent.com/en-us/)  [↑](#footnote-ref-1)
2. [pyg-team/pytorch\_geometric: Graph Neural Network Library for PyTorch](https://github.com/pyg-team/pytorch_geometric) [↑](#footnote-ref-2)