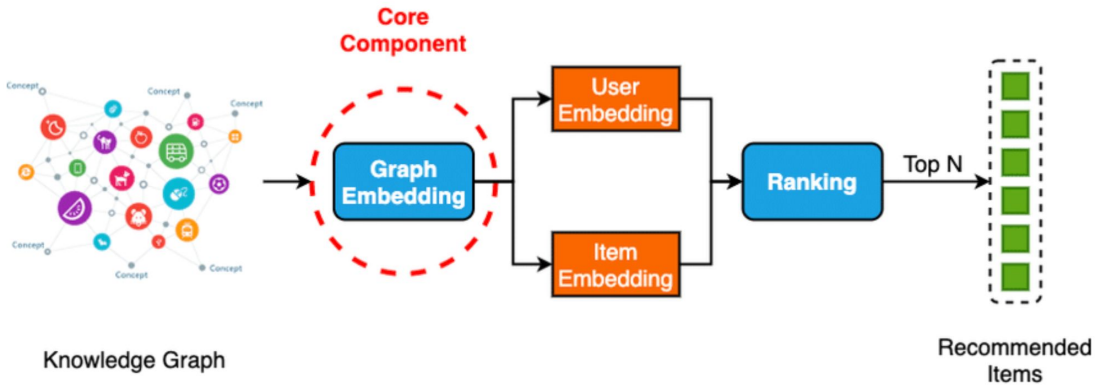
Group 2: Learning-Based Hashing with GNN for Recommender System

Jiaying Li, Yuhan Shao, Yu Hou, Wenhe Zhang

Recommendation System with GNN

1. The graph embedding module of the GNN recommender first learns to produce an embedding for every graph node including the user and item nodes.

2. For a given user, the ranking module computes a match score between this user and each of the candidate items according to their corresponding embeddings.



Motivation

1. Why we use **LightGCN**:
   - LightGCN simplify the design of GCN to make it more concise and efficient for recommendation.
2. Why we use the **Hash(Encoding) function** with LightGCN:
   - Despite normal graph representation learning is effective in learning embedding vectors for objects in the user-item interaction network, the computational costs to infer users' preferences in continuous embedding space are tremendous.
3. Why we use a novel **Decoding function**:
   - The existing hashing scheme (i.e., linear mapping followed by binarization as the encoder and Hamming distance as the decoder) is not effective enough to preserve the exact IP values.

Methods: HashLGN

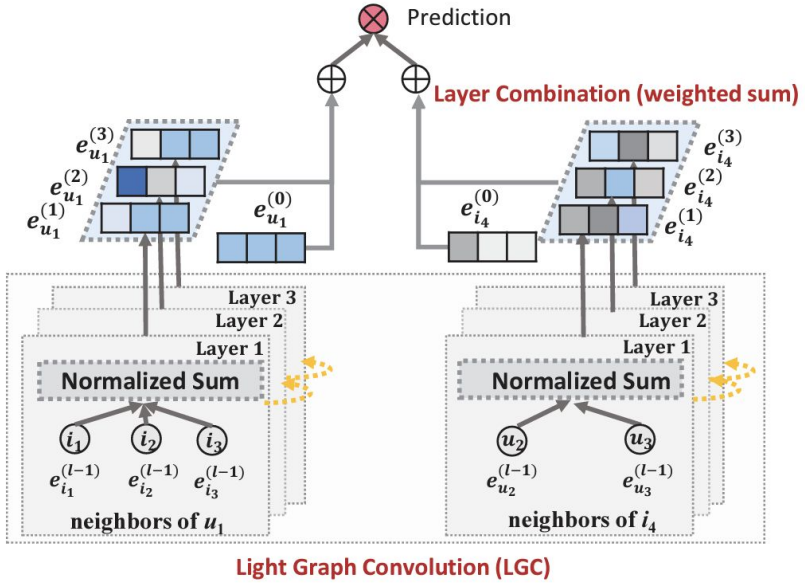
- Graph Encoder: LightGCN
- Learning-based Hashing
- Combined Model: HashLGN

Graph Encoder: LightGCN

- **Light Graph Convolution (LGC)**: Use the simple weighted sum aggregator (neighbor propagation) and abandon the use of feature transformation and nonlinear activation

$$\mathbf{e}_u^{(k+1)} = \mathrm{AGG}(\mathbf{e}_u^{(k)}, \{\mathbf{e}_i^{(k)} : i \in \mathcal{N}_u\}).$$

- **Layer Combination and Model Prediction**: the only trainable model parameters are the embeddings e_u^0 for all users and e_i^0 for all items at the 0-th layer



In 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, we sum over the embeddings at each layer to obtain the final representations

The Hash and Decoding Function

- **Problem Set-Up**: Reuse the idea of learnable hash and decoding function by defining:
  - **Hash Function**: $h(\cdot) : \mathcal{R}^m \rightarrow \{\pm 1\}^d$

    Build the hash function based on a multi-layer perceptron (MLP), in our project, our MLP consists of 2 or 3 fully connected layers, and we use the sign function as the activation function to generate binary outputs.

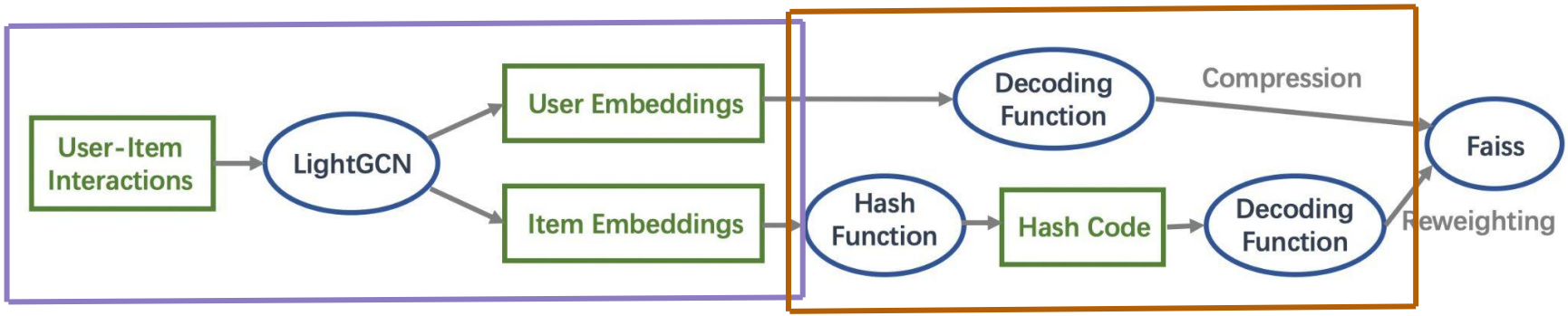
  - **Decoding Function**: $\Omega : \{\pm 1\}^d \times \mathcal{R}^m \rightarrow \mathcal{R}$, where $\Omega(h(o); \tilde{q}) \approx q^T o$

    Goal: estimate the value of $q^T o$ using q and o's hashcode $c = h(o)$.

    - ➢ **Reweighting function:** learnable, transforms discrete code c into a continuous d-dimensional vector. $g(c) = \alpha \odot \max(c, 0) + \beta \odot \min(c, 0)$
    - ➢ **Compression function**: approximate the inner product of q and o based on the compressed vector $f(q)$ for q and the re-weighted vector $g(c)$ for o's hashcode c. $\Omega(c; q) = f(q)^T g(c).$

Combined Model: HashLGN

Combine the power of **LightGCN**, **Hash and Decoding Function**



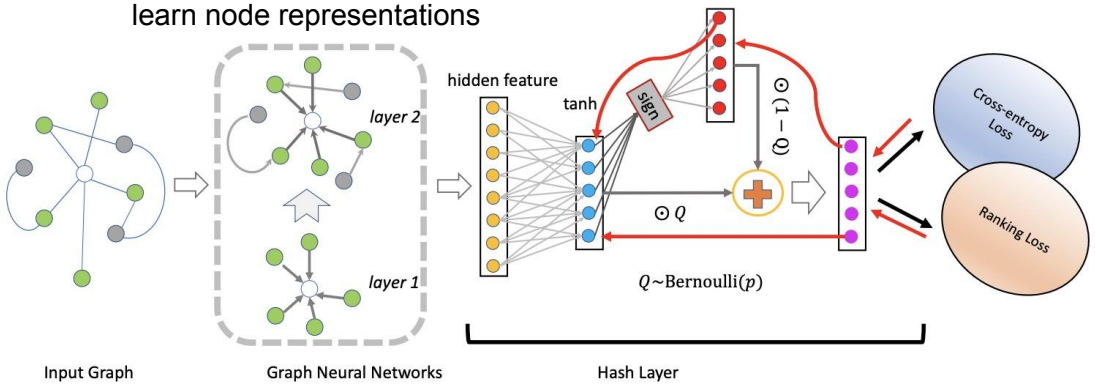
Training Jointly with BPR Loss

- **Bayesian Personalized Ranking (BPR)** loss
  - Pairwise loss: Encourage the prediction of interacted items to be higher than the items that user never interacted with
  - Formula:

$$L_{BPR} = -\sum_{u=1}^{M} \sum_{i \in \mathcal{N}_u} \sum_{j \notin \mathcal{N}_u} \ln \sigma(\hat{y}_{ui} - \hat{y}_{uj}) + \lambda ||\mathbf{E}^{(0)}||^2$$

  - During training, we fed the decoded embeddings of users and items into BPR Loss
    - $\hat{y}_{ui}$: The inner product between user and interacted item embedding
    - $\hat{y}_{uj}$: The inner product between user and uninteracted item embedding
    - $\mathbf{E}^{(0)}$: The embeddings at the 0-th layer of LightGCN

Experiment



Dataset: MovieLens 1M

- include ratings of 6040 users and 1000209 user-item interactions in total
- training set : testing set = 8:2

Three frameworks

- Baseline 1: LightGCN
- Baseline 2: HashGNN
- Our method: HashLGN (LightGCN with Hash and Decoding Function)

Evaluation metrics

Precision: $$\textbf{Precision@}K(u) = \frac{|R^K(u) \cap T(u)|}{K}$$

Recall: $$\textbf{Recall@}K(u) = \frac{|R^K(u) \cap T(u)|}{|T(u)|}$$

F1: $$\textbf{F1@}K(u) = \frac{2 \times \textbf{Precision@}K(u) \times \textbf{Recall@}K(u)}{\textbf{Precision@}K(u) + \textbf{Recall@}K(u)}$$

NDCG: $$\textbf{NDCG@}K = \frac{1}{|\mathcal{U}|} \Sigma_{u \in \mathcal{U}} \frac{\Sigma_{k=1}^{K} \frac{I(R_k^K(u) \in T(u))}{\log(k+1)}}{\Sigma_{k=1}^{K} \frac{1}{\log(k+1)}}$$

Normalized Discounted Cumulative Gain (NDCG)

- is popular method for measuring the quality of a set of search results
- the contributions of the appropriately recommended items are differentiated by NDCG depending on their ranking positions

Average search time: the average search time for getting the search result for all the users (6040)

Result

Maximum inner product search(MIPS): returns the vectors with the top-K largest IP values
Threshold inner product search(TIPS): returns all vectors with IP values larger than a given threshold

Method	Precision		Recall		F1 Score		NDCG		Avg Search Time	
	Top50	Top100	Top50	Top100	Top50	Top100	Top50	Top100	Top50	Top100
HashGNN	**22.25%**	13.96%	28.91%	34.26%	0.2023	0.1634	**0.5930**	**0.6578**	~ 31.5*s*	~ 28*s*
LightGCN	20.87%	**15.22%**	**41.14%**	**55.58%**	**0.2762**	**0.2384**	0.3982	0.4415	~ 14.1734*s*	~ 14.6762*s*
HashLGN	16.76%	12.82%	31.51%	45.85%	0.2181	0.1998	0.2931	0.3384	~ **13.5427s**	~ **13.7535s**

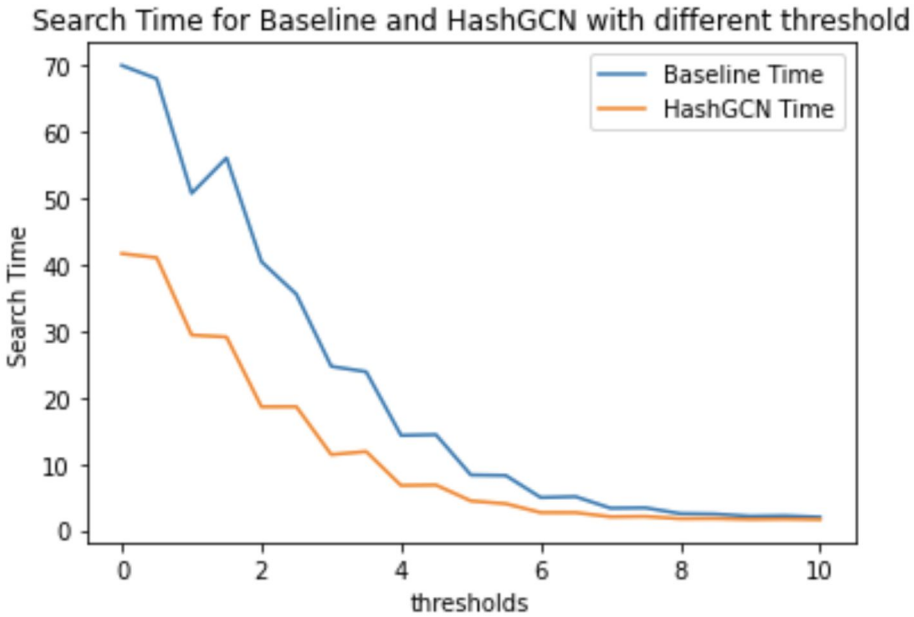
Table 1. Performance comparison of two baselines and the HashLGN model on topK-based search.

Method	Precision	Recall	F1 Score	NDCG	Threshold	Avg Search Time
HashGNN	15.90%	29.98%	0.1594	**0.6582**	5	~ 24.51*s*
LightGCN	**21.18%**	45.23%	**0.2866**	0.0913	6	~ 22.73*s*
HashLGN	14.57%	**46.65%**	0.2201	0.1317	4	~ **12.07s**

Table 2. Performance comparison of two baselines and the HashLGN model on threshold-based search.

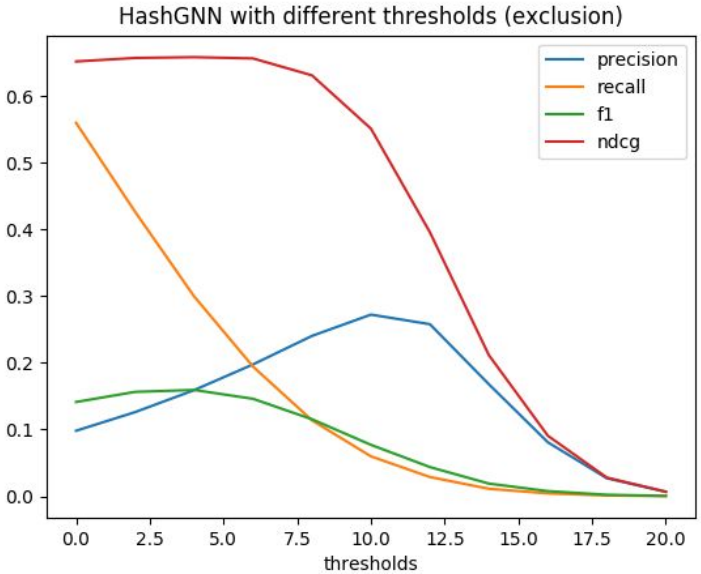
Search time

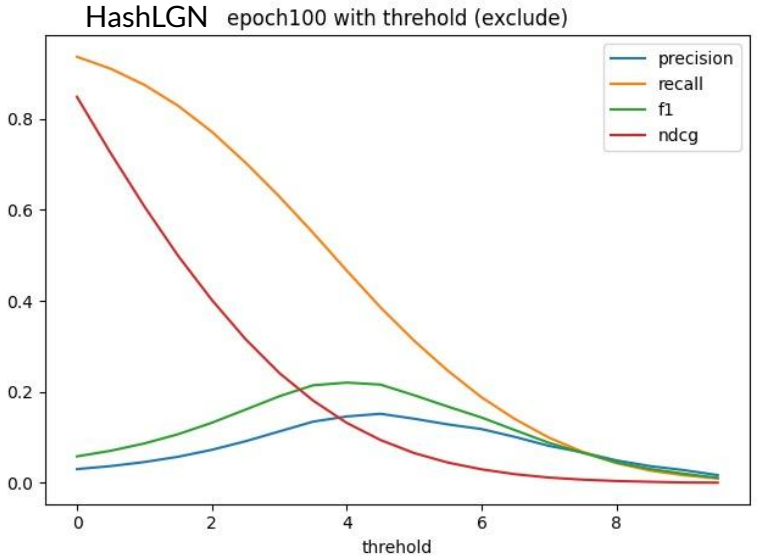
Much higher efficiency!!!



Result

HashLGN has better performance than HashGNN





Conclusions

- In this work, we proposed a new framework called HashLGN which consists of two parts: **a LightGCN encoder** for learning node representations, and **a hash and decoding function** to achieve higher efficiency in inner product search.

- Comprehensive experiments demonstrate that our model can achieve comparable performance with the state-of-art models but with much **higher efficiency**.