Blockchain Recommender Systems using Blockchain Data

***Abstract*—While there is extensive research in designing recommender systems for centralized applications, blockchain recommender systems remain under-explored. To fill this gap, we investigate collaborative filtering (CF), a popular recommendation approach that captures similarities among users in terms of their transaction histories. For example, if two users liked movies a, b and c, and the first user additionally liked movie d, then CF may suggest movie d to the second user. We show how to identify similar smart contracts to enable richer recommendations, akin to recommending movie e in the above example if it features similar actors as d. Using Ethereum transaction data, we test two CF methods: a simple and fast Matrix Factorization algorithm and a state-of-the-art method for recommender systems based on Graph Neural Networks (GNNs). We find that GNN outperforms MF and is within 10-15% of its effectiveness on a larger movie recommendation dataset.**

***Index Terms*—Blockchains, Smart contracts, Decentralized applications, Recommender systems, Personalization**

I. Introduction

Personalization is critical for user acquisition, satisfaction, and retention. Research has shown that an effective enabler of personalization is the use of recommender systems [1]. However, while there has been extensive research in designing recommender systems for centralized applications, blockchain recommender systems remain under-explored [2]. This is the gap we intend to fill in this paper.

Recommender systems can be categorized by the data they require: user interactions with items, user attributes, or item attributes. Based on this categorization, there are three approaches: Collaborative Filtering (CF), Content-Based (CB), and Hybrid [3]. CF methods focus on interactions. For example, if user u_1 interacted with items a, b, and c, and user u_2 interacted with items a and b, then item c could be recommended to user u_2. Conversely, CB methods consider similarities among user and item attributes. For instance, if user u_1 frequently watches video game streams, gaming items might be recommended to them based on content similarity. Finally, Hybrid methods take into account both interactions and the attributes of users and items. For example, suppose user u_1 and user u_2 interacted with items a, b, and c, and user u_3 interacted with items a and b. If the profile of item d is similar to the preferences of user u_3, then item c could be recommended due to the similarity in interactions, while item d could be recommended due to the similarity in attributes.

A blockchain transaction is an interaction between a user address (public key) and a smart contract address. CF is therefore feasible, using the transaction records available on the ledger. On the other hand, CB methods may not apply since the pseudo-anonymous blockchain users are known only by their public keys and lack additional descriptive attributes[1]. Finally, Hybrid approaches can be considered, based on the similarity of items in terms of their attributes or descriptions. However, the challenge is how to identify similar smart contracts, which are different from product descriptions or movie genres found in e-commerce or movie recommender systems.

To address this challenge, we propose two methods to identify similar contracts based on the comments within the code, using clustering and word embeddings. We then combine these methods with two popular recommender approaches and make the following contributions[2]:

- Conceptually, we formulate a decentralized application (dApp) recommendation problem that suggests smart contracts to users. Prior efforts on blockchain recommendations were restricted to digital tokens (NFTs) as items [4]. In contrast, we consider a general framework, treating smart contracts as items to recommend and considering smart contract similarity in addition to user-item interactions when making recommendations. Generalizing beyond cryptocurrency and NFT transfer is critical to accommodate a variety of emerging applications such as decentralized auctions.
- As a technical contribution, we implement two methods to enable hybrid recommender approaches by incorporating additional contract information: (1) clustering similar contracts together and (2) contract embeddings.
- Using a dataset of 400,000 Ethereum transactions, we compare the effectiveness and efficiency of CF and hybrid versions of simple and fast Matrix Factorization (MF) as well as Graph Neural Networks (GNN) which is one the state-of-the-art methods for recommender systems [5]. We also compare effectiveness on our dataset versus a well-known movie recommendation dataset.

II. Related Work

The closest work to ours is by Yu et al., on recommending Non-Fungible Tokens (NFTs) [4]. Our focus is more broad, on a general dApp recommender system based on user interactions with arbitrary smart contracts, as required for emerging applications beyond cryptocurrency and NFT trading.

[1]Linking blockchain addresses to Web profiles (e.g., those available on social networks) could create more detailed user profiles; however, this raises privacy concerns and users may not disclose their blockchain addresses on these platforms anyway.

[2]Our source code is available at https://github.com/anonymoust016/blockchain_recommender_system.

More broadly, previous work on blockchain transaction analytics can be categorized into identity inference, anomaly detection, and price prediction [6]. Identity inference refers to determining the real-world identities or attributes associated with blockchain addresses, mainly using classification techniques [7]–[10]. Anomaly detection refers to identifying unusual or suspicious activity such as sybil attacks, in which a single entity creates multiple fake identities to gain control of a network. Here, supervised learning often falls short due to the class imbalance between legitimate and illicit transactions, leading to the use of rule-based or unsupervised learning methods such as clustering [6]. Finally, price prediction refers to forecasting future cryptocurrency or NFT prices, using techniques from machine learning and time series analysis.

III. Data and Methods

We focus on Ethereum, the largest smart contract platform with nearly 1 million transactions per day. To ensure that our experiments finish in a reasonable time, we sampled 400,000 Ethereum transactions as follows[3]. We selected a random day, December 18, 2022, and, out of the 917,000 public keys who ran at least one smart contract that day, we selected 42,000 at random. For these 42,000 users, we downloaded their most recent transactions using the Etherscan API[4], up to the API limit of 50. The resulting 400,000 transactions are with 31,527 unique smart contracts, with an average of ten contracts per user. Each transaction includes the from (user) and to (contract) addresses, and we also used Etherscan to download the corresponding contract code. We use the solidity-parser[5] library to parse the code and extract comments.

Fig. 1 illustrates the space of recommendation methods we consider, according to data needs on the vertical axis (CF vs. Hybrid) and model complexity on the horizontal axis (Matrix Factorization (MF) vs. Graph Neural Network (GNN)).

The vertical axis distinguishes between CF approaches that only consider user-item interactions and Hybrid approaches that additionally consider item (smart contract) attributes. We found that 74% of Ethereum contracts in our dataset (23,310 out of 31,257) have a comment at the beginning of the contract class. For instance, Fig. 2 shows comments from a staking pool contract and a lending pool contract that describe their purpose. We therefore propose two hybrid approaches based on the comments in the code: (1) clustering similar contracts together, and (2) an embedding approach that captures comment similarity.

For clustering, we selected Latent Dirichlet Allocation (LDA), a popular topic modelling method that groups similar documents (in our case, contracts with similar comments) based on word usage. We applied the "Elbow method" to select a good number of clusters, which turned out to be 15. In the elbow method, we plot model performance (a *coherence* measure that quantifies the similarity of contracts assigned to the same cluster) as a function of the number of clusters. The elbow in this curve corresponds to a point where adding more clusters does not make a substantial difference.

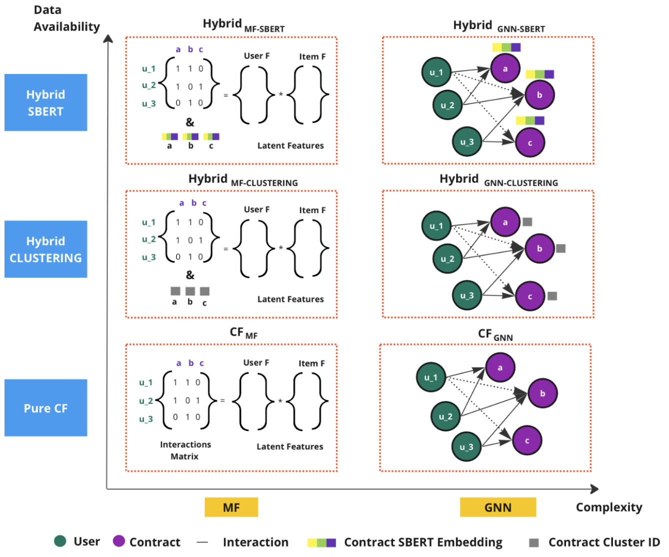
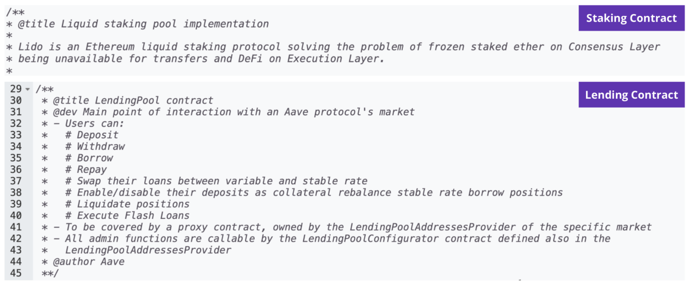


Fig. 1. Space of dApp recommender systems considered



```
/**
* @title Liquid staking pool implementation
*
* Lido is an Ethereum liquid staking protocol solving the problem of frozen staked ether on Consensus Layer
* being unavailable for transfers and DeFi on Execution Layer.
*
```

```
/**
 * @title LendingPool contract
 * @dev Main point of interaction with an Aave protocol's market
 * - Users can:
 *   # Deposit
 *   # Withdraw
 *   # Borrow
 *   # Repay
 *   # Swap their loans between variable and stable rate
 *   # Enable/disable their deposits as collateral rebalance stable rate borrow positions
 *   # Liquidate positions
 *   # Execute Flash Loans
 * - To be covered by a proxy contract, owned by the LendingPoolAddressesProvider of the specific market
 * - All admin functions are callable by the LendingPoolConfigurator contract defined also in the
 *   LendingPoolAddressesProvider
 * @author Aave
 **/
```

Fig. 2. Lending and staking contract comments.

For contract (comment) embeddings, we use Sentence-BERT (SBERT) [11], which projects sentences into a 768-dimensional numeric space, such that sentences with similar meanings are close together in the projected space. Again, this is a common approach in text mining to convert text to a numeric representation that preserves semantic similarity.

Next, the horizontal axis represents algorithmic complexity. We start with Singular Value Decomposition (SVD), which is a simple and fast MF approach, as a popular example of CF [12] (Fig. 1, bottom left, labelled CF_{MF}). We use the Python implementation of SVD from the LightFM [13] library. MF takes in an interaction matrix with user IDs as rows and item IDs as columns. In our context, the (i, j)th entry of the interaction matrix is one if the ith user interacted with the jth smart contract and zero otherwise. MF decomposes the interaction matrix into a product of simpler user and item matrices that capture latent factors of users and items, revealing "hidden" features that can predict future interactions.

Transitioning to Hybrid MF, adding the cluster-ID assigned by LDA to each contract leads to $\text{Hybrid}_{\text{MF-CLUSTERING}}$, and adding the SBERT embedding to each contract leads to $\text{Hybrid}_{\text{MF-SBERT}}$, illustrated on the middle and top left of Fig. 1.

For neural methods, we selected Graph Neural Networks (GNNs) [14], which is one of the state-of-the-art recom-

[3]More data should generally lead to better recommendations, making sampling appropriate for our proof-of-concept analysis.
[4]https://etherscan.io
[5]https://www.github.com/consensys/python-solidity-parser/

mendation methods [5]. GNNs can capture relationships that extend beyond immediate neighbours. For example, if user u_1 interacts with item a, user u_2 interacts with items a and b, and user u_3 interacts with items b and c, GNN methods can enable the higher-order recommendation of item c to user u_1. These models achieve this by predicting whether a link should exist between the user and contract nodes. Pure CF GNN only considers user-contract interactions, while our Hybrid GNNs incorporate cluster IDs (Hybrid$_{\text{GNN-CLUSTERING}}$) or SBERT embeddings (Hybrid$_{\text{GNN-SBERT}}$) alongside the interaction data. Fig. 1 illustrates the GNN methods on the right-hand side, which we implemented using the Deep GraphSage framework [15], in PyTorch 1.13[6].

We end this section by pointing out a baseline method for comparison, which we call POP: always recommend the most popular smart contract(s).

IV. Experiments

We now present recommendation performance (Section IV-A), recommendation efficiency (Section IV-B), and a comparison of recommendation performance on our Ethereum dataset and a movie recommendation dataset (Section IV-C). The experiments were conducted on a Linux server operating Ubuntu 21.10 and Python 3.9, equipped with 24 CPU cores and an NVIDIA 2080Ti GPU that was used to train the GNNs.

We consider the POP baseline, two pure collaborative filtering approaches (Matrix Factorization and Graph Neural Networks), and two hybrid variants of each of the above two approaches (using clustering and SBERT embeddings). This gives 7 methods in total: 1) POP, 2) CF$_{\text{MF}}$, 3) Hybrid$_{\text{MF-CLUSTERING}}$, 4) Hybrid$_{\text{MF-SBERT}}$, 5) CF$_{\text{GNN}}$, 6) Hybrid$_{\text{GNN-CLUSTERING}}$, and 7) Hybrid$_{\text{GNN-SBERT}}$. Each one outputs a *ranked* list of items, and we use the following standard metrics from the recommender systems literature to evaluate the relevance of the k most highly ranked recommendations:

Hit at k (HIT@k) is the fraction of users with at least one relevant recommendation within the top-k recommendations:

$$\text{HIT@}k = \frac{1}{|U|}\sum_{u=1}^{|U|} \mathbb{1}\left(\bigcup_{i=1}^{k}\{\text{rel}_{ui} > 0\}\right)$$

where U is the set of user addresses, $\mathbb{1}(\cdot)$ is the indicator function which equals 1 if user u has at least one relevant contract in the top-k recommended ones and 0 otherwise, and rel_{ui} indicates the relevance of contract i to user u.

Mean Average Precision at k (MAP@k) assesses the average precision of the model across all users, based on the top-k recommendations produced for each user:

$$\text{MAP@}k = \frac{1}{|U|}\sum_{u=1}^{|U|}\frac{1}{\min(m,K)}\sum_{k=1}^{k}\text{Precision@}k \cdot \Delta\text{rel}_{uk}$$

where U is the set of user addresses, m is the total number of relevant contracts for user u, and Δrel_{uk} equals 1 if recommended contract at rank k is relevant and 0 otherwise.

[6]https://pytorch.org/docs/stable/torch.html

Precision@k is the precision at cutoff k in the ranked list of recommendations, defined as the proportion of relevant items found in the top k recommendations.

Normalized Discounted Cumulative Gain at k (NDCG@k) accounts for the position of relevant items in the ranked list of recommendations, giving higher importance to hits at the top of the list. For a list of k ranked items, NDCG is defined as:

$$\text{NDCG@}k = \frac{\text{DCG}_k}{\text{IDCG}_k}$$

where $\text{DCG@}k = \sum_{i=1}^{k}\frac{\text{rel}_i}{\log_2(i+1)}$, and IDCG@$k$ is the DCG value of the ideal best-possible ranking. rel_i is the relevance score of each item in the ranked list. If the recommended item exists in the ground truth (i.e., the list of actual items the user interacted with), $\text{rel}_i = 1$, and zero otherwise.

We compute the above metrics using 5-fold cross-validation. We also applied *negative sampling*, which is a common method to mitigate class imbalance due to interaction matrix sparsity (users typically interact with a small fraction of the available items) [16]–[18]. Here, for each user, we keep all the contracts they interacted with and randomly sample 100 contracts they did not interact with.

A. Recommendation Effectiveness

Table I summarizes the results at different rank thresholds, from one to 20, with bold numbers corresponding to the best-performing model. Hybrid$_{\text{GNN-SBERT}}$ outperforms all other models across the board. Interestingly, incorporating SBERT embeddings capturing smart contract similarity is effective for GNN but not for MF. This could be because GNNs are already designed to learn embeddings to represent the graph structure, and SBERT can enrich these embeddings with additional contract information. On the other hand, neither MF nor GNN seem to benefit from the addition of contract cluster IDs. Finally, POP only performs well for $k = 1$, indicating that the most popular contract overall is relevant to many users. However, as k increases, user preferences start to matter, and the use of interactions (CF methods) and item attributes (Hybrid methods) leads to better performance.

B. Recommendation Efficiency

We now turn from effectiveness to efficiency, in terms of inference running time (latency) and the memory footprint. We focus on inference efficiency to assess the models' ability to serve recommendations in nearly real-time. We do not measure training efficiency since this can be done offline and off-chain. We omit the trivial POP recommender from this comparison (which simply memorizes the k most popular contracts).

Table II reports the latency (CPU running times to compute all recommendations during 5-fold cross validation) and memory usage. We see that the effectiveness of GNN-based methods comes at the expense of latency and memory usage. However, dividing the total latency by the number of recommendations gives roughly 0.25 milliseconds per recommendation for GNN-based methods (and roughly 0.05 milliseconds for the MF methods), which is acceptable in practice [19].

TABLE I
EFFECTIVENESS OF POP, CF$_{MF}$, HYBRID$_{MF\text{-}CLUSTERING}$, HYBRID$_{MF\text{-}SBERT}$, CF$_{GNN}$, HYBRID$_{GNN\text{-}CLUSTERING}$, AND HYBRID$_{GNN\text{-}SBERT}$ ACROSS MULTIPLE RANK THRESHOLDS

	k	POP	CF$_{MF}$	Hybrid$_{MF\text{-}CLUSTERING}$	Hybrid$_{MF\text{-}SBERT}$	CF$_{GNN}$	Hybrid$_{GNN\text{-}CLUSTERING}$	Hybrid$_{GNN\text{-}SBERT}$
Hit@k	1	0.525	0.629	0.594	0.590	0.626	0.621	**0.652**
	5	0.532	0.778	0.739	0.775	0.823	0.821	**0.845**
	10	0.532	0.826	0.788	0.826	0.881	0.880	**0.905**
	15	0.532	0.854	0.820	0.856	0.912	0.908	**0.931**
	20	0.532	0.874	0.840	0.875	0.934	0.926	**0.949**
MAP@k	1	0.525	0.629	0.594	0.590	0.626	0.621	**0.652**
	5	0.528	0.670	0.638	0.647	0.680	0.676	**0.703**
	10	0.528	0.651	0.622	0.629	0.658	0.655	**0.679**
	15	0.528	0.636	0.606	0.614	0.643	0.638	**0.660**
	20	0.528	0.624	0.594	0.601	0.629	0.625	**0.646**
NDCG@k	1	0.525	0.629	0.594	0.590	0.626	0.621	**0.652**
	5	0.377	0.552	0.505	0.533	0.581	0.578	**0.607**
	10	0.373	0.573	0.522	0.556	0.613	0.611	**0.643**
	15	0.373	0.589	0.539	0.573	0.635	0.633	**0.666**
	20	0.373	0.601	0.550	0.585	0.650	0.648	**0.681**

TABLE II
LATENCY AND MEMORY USAGE OF CF$_{MF}$, HYBRID$_{MF\text{-}CLUSTERING}$, HYBRID$_{MF\text{-}SBERT}$, CF$_{GNN}$, HYBRID$_{GNN\text{-}CLUSTERING}$, AND HYBRID$_{GNN\text{-}SBERT}$

	Latency (s)	**Memory (MB)**
CF$_{MF}$	**6.7**	**608**
Hybrid$_{MF\text{-}CLUSTERING}$	6.8	611
Hybrid$_{MF\text{-}SBERT}$	6.7	1216
CF$_{GNN}$	27.7	2518
Hybrid$_{GNN\text{-}CLUSTERING}$	28.1	2535
Hybrid$_{GNN\text{-}SBERT}$	30.3	2621

C. Recommendation Effectiveness on Ethereum vs. MovieLens

Finally, we examine how the numbers reported in Table I compare to recommendation performance in "classical" applications. For this, we compare our best model (Hybrid$_{GNN\text{-}SBERT}$) on our Ethereum dataset and MovieLens-100k, a well-known movie review dataset with 100,000 reviews [20]. In MovieLens, movie ratings constitute the interactions and movie genres are the item attributes. Since the average number of interactions per user in MovieLens-100k is 182, compared to 10 in our dataset, we consider two variants of MovieLens data: MovieLens$_{FULL}$, containing all interactions, and MovieLens$_{SAMPLED}$, maintaining the same users-to-items ratio as the Ethereum dataset (approximately 1.3) while randomly retaining 10% of each user's interactions, resulting in an average of 16 interactions per user.

Table III shows the results at various rank thresholds. Hybrid$_{GNN\text{-}SBERT}$ performs 10-15% worse on Ethereum compared to MovieLens$_{FULL}$. This is likely due to more interactions in the MovieLens dataset: the more interactions for a user a recommender method has access to, the better it can provide recommendations. Additionally, Hybrid$_{GNN\text{-}SBERT}$ on Ethereum outperforms MovieLens$_{SAMPLED}$, illustrating the potential of Ethereum data to provide recommendations. That being said, a user study would need to be done to verify that our blockchain recommendations are useful.

TABLE III
EFFECTIVENESS OF HYBRID$_{GNN\text{-}SBERT}$ ON ETHEREUM AND MOVIELENS ACROSS MULTIPLE RANK THRESHOLDS

	k	Ethereum	MovieLens$_{SAMPLED}$	MovieLens$_{FULL}$
HIT@k	1	0.652	0.365	**0.776**
	5	0.845	0.740	**0.967**
	10	0.905	0.839	**0.992**
	15	0.931	0.893	**0.997**
	20	0.949	0.921	**0.998**
MAP@k	1	0.652	0.365	**0.776**
	5	0.703	0.489	**0.831**
	10	0.679	0.462	**0.798**
	15	0.660	0.441	**0.774**
	20	0.646	0.424	**0.759**
NDCG@k	1	0.652	0.365	**0.776**
	5	0.607	0.394	**0.749**
	10	0.643	0.434	**0.748**
	15	0.666	0.466	**0.758**
	20	0.681	0.491	**0.768**

V. CONCLUSIONS AND FUTURE WORK

We investigated the potential of providing recommendations using blockchain data. We considered Collaborative Filtering using Matrix Factorization, Collaborative Filtering using GNNs, two variations of Hybrid Matrix Factorization, and two variations of Hybrid GNNs. We found that: (1) GNN methods outperform MF methods and are within 10-15% of their performance on a movie recommendation dataset, (2) MF methods are more efficient than GNN methods, and (3) incorporating smart contract embeddings to create hybrid methods was effective for GNN but not for MF.

In this study, we used the GraphSAGE GNN architecture. One direction for future work is to investigate transformer-based GNN architectures such as Graphormer [21], [22]. Furthermore, we will investigate the tokenomics and incentive mechanisms required for the practical implementation of a decentralized recommendation system on a blockchain platform.

References

[1] M. Gorgoglione, U. Panniello, and A. Tuzhilin, "Recommendation strategies in personalization applications," *Information & Management*, 2019.

[2] G. Adomavicius, N. Manouselis, and Y. Kwon, "Multi-criteria recommender systems," in *Recommender Systems Handbook*, 2011.

[3] D. Jannach, M. Zanker, M. Ge, and M. Gröning, "Recommender systems in computer science and information systems - a landscape of research," in *Proceedings of the 13th International Conference on E-Commerce and Web Technologies*, 2012.

[4] G. Yu, Q. Wang, T. Altaf, X. Wang, X. Xu, and S. Chen, "Predicting nft classification with gnn: A recommender system for web3 assets," in *IEEE International Conference on Blockchain and Cryptocurrency*, 2023.

[5] 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," *ACM Transactions on Recommender Systems*, 2023.

[6] X. F. Liu, X.-J. Jiang, S.-H. Liu, and C. K. Tse, "Knowledge discovery in cryptocurrency transactions: A survey," *IEEE Access*, 2021.

[7] H. H. S. Yin, K. Langenheldt, M. Harlev, R. R. Mukkamala, and R. Vatrapu, "Regulating cryptocurrencies: A supervised machine learning approach to de-anonymizing the bitcoin blockchain," *Journal of Management Information Systems*, 2019.

[8] K. Toyoda, P. T. Mathiopoulos, and T. Ohtsuki, "A novel methodology for hyip operators' bitcoin addresses identification," *IEEE Access*, 2019.

[9] M. Jourdan, S. Blandin, L. Wynter, and P. Deshpande, "Characterizing entities in the bitcoin blockchain," in *2018 IEEE international conference on data mining workshops (ICDMW)*. IEEE, 2018, pp. 55–62.

[10] S. Ranshous, C. A. Joslyn, S. Kreyling, K. Nowak, N. F. Samatova, C. L. West, and S. Winters, "Exchange pattern mining in the bitcoin transaction directed hypergraph," in *International Conference on Financial Cryptography and Data Security*, 2017.

[11] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using siamese BERT-networks," in *arXiv*, 2019.

[12] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, 2009.

[13] M. Kula, "Metadata embeddings for user and item cold-start recommendations," in *arXiv*, 2015.

[14] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, "Graph neural networks in recommender systems: A survey," *ACM Computing Surveys*, 2022.

[15] D. El Alaoui, J. Riffi, A. Sabri *et al.*, "Deep graphsage-based recommendation system: jumping knowledge connections with ordinal aggregation network," *Neural Computing and Applications*, 2022.

[16] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 173–182.

[17] S. Rendle, W. Krichene, L. Zhang, and J. Anderson, "Neural collaborative filtering vs. matrix factorization revisited," in *Proceedings of the 14th ACM Conference on Recommender Systems*, 2020, pp. 240–248.

[18] 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*, 2019, pp. 165–174.

[19] M. Singh, "Scalability and sparsity issues in recommender datasets: a survey," *Knowledge and Information Systems*, 2020.

[20] F. M. Harper and J. A. Konstan, "Movielens datasets: History and context," *ACM Transactions on Interactive Intelligent Systems*, 2016.

[21] Y. Shi, S. Zheng, G. Ke, Y. Shen, J. You, J. He, S. Luo, C. Liu, D. He, and T.-Y. Liu, "Benchmarking graphormer on large-scale molecular modeling datasets," *arXiv*, 2022.

[22] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform badly for graph representation?" in *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.