INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION

journal homepage : www.joiv.org/index.php/joiv

Continuous Training of Recommendation System for Airbnb Listings Using Graph Learning

Yun-Hong Chan [a], Kok-Why Ng [a,*], Su-Cheng Haw [a], Naveen Palanichamy [a]

[a] *Faculty of Computing and Informatics, Multimedia University, Persiaran Multimedia, 63100 Cyberjaya, Selangor, Malaysia.*

*Corresponding author: *kwng@mmu.edu.my*

Abstract—**Recommender systems are getting increasingly important nowadays as they can boost user engagement and benefit businesses. However, there remain some unsolved problems. This paper will address two key performance issues. First, the limited ability to identify and leverage intrinsic relationships between data points. Second, the inability to adapt to new data. The first issue is proposed to be addressed through a Graph Neural Network (GNN) to curate better recommendations. GNN will be trained with Airbnb's review data to utilize its outstanding expressive power to represent complex user-listing interactions at scale, followed by generating embeddings to compute the relevant recommendations to the users. With the generated embeddings, the recommender system will compute a recommendation list to every user based on the embedding similarity between the user and listings or the user's first-ever reviewed listing and listings. The second issue is proposed to be resolved by incorporating Continuous Training. The proposed recommender system employs GraphSAGE with a customized Rating-Weighted Triplet Ranking Loss function, which outperformed unsupervised GraphSAGE. Offline simulation validated the recommender system's ability to learn from the latest data and improve over time. Overall, the proposed user-to-item (U2I) recommendation rating-weighted GraphSAGE substantially increased by 99.88% in hit-rate@5 and 98.15% in coverage. This offers an effective solution for enhancing the recommender system for Airbnb listings. This research validates the efficacy of GNN-based recommendations in capturing user-item relationships to aid in predicting relevant recommendations, thus significantly driving up the adoption of GNN-based recommender systems.**

Keywords— **Recommender system; graph neural network; deep learning; continuous training.**

Manuscript received 3 Nov. 2023; revised 29 Dec. 2023; accepted 21 Feb. 2024. Date of publication 31 Mar. 2024.

I. Introduction

Recently, recommender systems have become crucial in various real-world applications [1]–[8], such as ad ranking and movie recommendations. They significantly impact user engagement by providing personalized experiences and facilitating serendipitous content discovery. To enhance user retention, it is essential to develop effective recommender systems that curate personalized experiences and provide timely recommendations [9].

There are various patterns for personalization in the world of research papers and industry papers regarding recommender systems, and one of them is graph learning, which leverages graph-structured data to capture relationships between users and items [10]. Deep learning techniques have further advanced the representation learning on graph-structured data. By mapping each node (user or item) to a vector representation, graph learning identifies structurally similar nodes and enriches our understanding of users' interests. In the real world, there is a lot of data that can be stored in graph-structured data as the entities are linked with certain relationships, such as the relationship created between the user and Airbnb Listing. Graph representation learning utilized the topological structure, graph structure and rich nodes' features information to convert graph nodes into embeddings that can be used for downstream machine learning tasks [11]. Our proposed recommender system employs GraphSAGE with a customized loss function named Rating-Weighted Triplet Ranking Loss. The proposed GNN is aimed to capture the complex relationship within user-item interactions while considering various weightages of user preferences towards different items. Integrating the graph structure information and relationships between data points into a machine-learning task can have a significant impact. For instance, a node's local neighborhood tends to have a more substantial influence on the target node than distant nodes. By leveraging the structural information of a node's local neighborhood, the recommender system can offer users more relevant and captivating items. At Pinterest, PinSage is employed and trained using a massive dataset consisting of 7.5 billion examples. The graph utilized for training comprises 3 billion nodes, representing pins and boards, along with 18 billion edges. Extensive evaluation through offline metrics, user studies, and A/B tests has revealed that PinSage outperforms comparable deep learning and graph-based alternatives, delivering superior quality recommendations [12].

Continuous Training (CT) automatically retraining and serving models in production. While many consider a machine learning project complete after model deployment, the iterative and cyclic nature of the model lifecycle necessitates ongoing monitoring and retraining. Model performance can degrade over time, mainly due to data changes such as feature drift and concept drift. To address this, the recommender system should incorporate Continuous Training to adapt to evolving user interests, ensuring relevancy even as user preferences change during platform engagement. As users' interests and behaviors might vary over time, resulting in the data faced by the recommender system being very volatile, building a recommender system requires considering this fact. Otherwise, it would not be adaptive enough to learn the latest users' interests and provide irrelevant recommendations to the users.

This paper will first explore and analyze various recommender system techniques and develop an end-to-end Continuous Training pipeline of the recommender system that incorporates graph learning. By integrating these approaches, the recommender system can dynamically adapt to user preferences and ensure the freshness and accuracy of recommendations over time. The following section will review the different graph learning techniques. Section 3 will discuss our proposed method. Section 4 is our experimental results. The last section will conclude our work.

II. Material and Method

This section reviews conducted on four different graph-learning-based techniques for recommender systems such as graph embedding, Graph Convolutional Network (GCN), GraphSAGE, and Graph Attention Network (GAT). Although each graph-learning-based technique has its pros and cons, they all share a common ground: generating embeddings for nodes in the graph.

A. Graph Embedding Techniques for Recommender Systems

Wang et al. [13] proposed constructing an item graph from users' behavior history and then applying the state-of-art graph embedding methods to learn the embedding of each item, dubbed Base Graph Embedding (BGE). The similarities are computed using the dot product of the embedding vector of items. Chen et al. [14] presented collaborative similarity embedding (CSE), a unified representation learning framework. CSE involves a direct similarity embedding module for modeling user-item associations and a neighborhood similarity embedding module for modeling user-user and item-item similarities. They aimed to exploit comprehensive, collaborative relations in a user-item bipartite graph for recommender systems.

B. Graph Convolutional Techniques for Recommender System

Graph Convolutional Network (GCN) learns informative embeddings of users and items by effectively aggregating information from their neighborhoods in graphs by utilizing convolution and pooling operations. Sun et al. [15] proposed Multi-Graph Convolution Collaborative Filtering (Multi-GCCF) that would consider the difference in the natures of the nodes and implement aggregation and transformation functions that are dependent on the nature of the nodes to ensure the relevance of the nature of the nodes with the embedding's construction process in graph convolution network. To achieve this, multiple graphs are utilized in the embedding learning process. It outperformed PinSage, which proved to be a powerful GraphSAGE-based recommender system algorithm by Pinterest for offline evaluation.

A novel framework called DGCN-BinCF (Binarized Collaborative Filtering with Distilling Graph Convolutional Network) is proposed by Wang et al. [16] to mine the hidden interactions between users and items from implicit feedback. This framework incorporates GCN-based Collaborative Filtering to capture high-order feature interaction via cross-operation.

Kang et al. [17] proposed Joint Multi-grained Popularity-aware Graph Convolution Collaborative Filtering (JMP-GCF). This method has specifically catered to capturing the signals related to modeling user preferences within and between different popularity granularities. They presented a separated Bayesian Personalized Ranking (BPR) loss to optimize the model's parameter to accommodate the architecture of capturing multigrain popularity features. A multistage stacked training method is also used to speed up convergence.

C. GraphSAGE Techniques for Recommender Systems

The majority of currently used methods for creating node embeddings are inherently transductive. This becomes problematic when graphs evolve and constantly encounter unseen nodes in the production machine learning systems. The importance of the inductive approach is highlighted as it is generalizable and useful in helping facilitate the construction of embeddings for new data in the same form of features. GraphSAGE is proposed in [11] to resolve it in a way that extends GCNs to the problem of inductive unsupervised learning, followed by a framework that generalizes the GCN strategy to use trainable aggregation functions—with PinSage deployed at Pinterest [12] thoroughly assessed the learned embeddings' quality on a variety of recommendation tasks. Offline metrics, user studies, and A/B tests all significantly boosted recommendation performance. PinSage algorithm utilized sampling the node neighborhood through short random walks and constructing a computation graph using sampled neighborhood in an on-the-fly way to perform efficient, localized convolutions. They proposed importance pooling, which uses scores to weight node features in the convolution layer, resulting in a 46% performance gain in offline evaluation metrics.

Inductive Matrix Completion (IMC) relies on side information to train the recommender system. These constraints on content quality often cause the model to perform inferiorly when high-quality content is unavailable.

However, Zhang et al. [18] introduced Inductive Graph-based Matrix Completion (IGMC) to remove this constraint, which is h-hop enclosing subgraph for each training user-item pair and feeding these subgraphs to Graph Neural Networks to learn rich graph pattern information from these subgraphs to map the subgraph to the rating that its center user gives to its center item, whereby corresponds to filling in the missing entries of the rating matrix.

SWAG (Sample, Weight, and AGgregate) in [19] proposed adapting GraphSAGE to weighted graphs. Their algorithm constitutes three components: Sampling, Weighting, and Aggregation. They apply sampling and aggregation operations to derive knowledge from edge weight. Then, weights on the graphs measure loss, sampling, and so on. MUlti-task Sampling and Inductive Learning on Graphs (MUSIG) in [20] is proposed to learn high-quality representations of tracks for several different use cases in music streaming platforms. This method has devised a strategy to ensure the embeddings can be used for other tasks and avoid a mismatch between original learning and downstream tasks.

D. Graph Attention Network Techniques for Recommender System

Graph Attention Networks (GAT) [21] introduce attention mechanisms into Graph Neural Networks to discriminatively learn the different importance and relevance of the nodes by specifying arbitrary weights to the neighbors. For the social recommender system, the user-item and social graphs provide information about users and social interactions between users from different perspectives. In [22], GraphRec is proposed to capture the interactions with heterogeneous strength when coherently modeling user-item graphs and social graphs. Three different attention mechanisms (item attention α, social attention β, and user attention) are introduced to extract the users with the most critical influence and suitable for characterizing users' social information.

In [23], a graph contextualized self-attention network (GC-SAN) is proposed for the session-based recommendation. They use all the historical session sequences to build a directed graph. Knowledge Graph Attention Network (KGAT) is proposed by Xiang et al. [24] to resolve the challenges in high-order connectivity modeling correspondingly. It is recursively embedding propagation. In Snapchat, Sankar et al. [25] propose GraFRank (Graph Attentional Friend Ranker) for multi-faceted friend ranking, contributing significantly to friend recommendation. Users can indirectly interact with friends by liking posts, or they can directly communicate with friends by texting and exchanging social content.

One of the impactful recommendation problems is the next-item recommendation, which is trying to predict what item the user will likely buy, and this will result in uplifting business revenue if the recommender system can achieve great recall. SequentiAl inTentiOn-aware Recommender proposes the next-item recommendation based on a user Interaction graph (Satori) in [26]. A user interaction graph is constructed to model relations among users, items, and categories. Next, user intention and user preference are the significant factors that contribute to whether users will buy the current item that is surfaced, so both are learned using Graph Attention Network. Then, the embedding of user intention is generated by feeding intention trajectory and utilizing self-attention with positional encoding [27].

E. Proposed Methodology

Below is the proposed methodology for this paper.

1) GNN-based Recommender System:

This research of the recommender system begins with the collection of data, including reviews and listings data. To simulate real-world scenarios, a time-based train-test split is performed on the review's dataset based on the timestamps of the reviews. This approach follows the principle of "training on the past and predicting the future," adhering to the limitation that the model can only access historical data. Next, the training and test data will be transformed into their respective heterogeneous bipartite graphs to capture the relationships between reviewers and listings. When a review is created, it signifies that a guest has rented an Airbnb listing and provided a rating and feedback. This creates a relationship between the reviewer and the listing. A heterogeneous bipartite graph that connects listings and reviewers is constructed by considering these relationships. In the next step, this paper will train the GraphSAGE model on the training graph, and the model will be optimized based on a customized loss function. Once the model training is completed, the best model with the most minor test loss is chosen. The GraphSAGE model will generate the embeddings for the nodes in the graph. Therefore, the recommender system will generate the reviewers' embeddings and listing' embeddings, which are in the same dimensional space after running model inference.

With the generated embeddings, the recommender system will compute a recommendation list for every reviewer. There are two ways of developing recommendations. The first is the user-to-item (U2I) recommendation, as the recommendations are ranked based on the similarity between the reviewer and the listing. For each reviewer, the system retrieves the top K listings most similar to the reviewer's embedding, measured by cosine similarity. The value of K represents the predetermined number of recommendations to be provided. A ground truth list is created for each reviewer by extracting all the listings the reviewer reviewed in the test data.

The second approach is known as item-to-item (I2I) recommendation, where recommendations are ranked based on the similarity between the reviewer's first-ever reviewed listing and other listings. Item-to-item recommendation suggests items to users based on their similarity to items they have previously shown interest in or engaged with. Firstly, the recommender system will filter out those reviewers who have only reviewed once in the test data. This step ensures sufficient test data is available for the evaluation stage. For each remaining reviewer, the recommender system generates recommendations by retrieving the top K listings that are most similar to the embedding of the first-ever reviewed listing, as measured by cosine similarity. A ground truth list is created for each reviewer by including all the listings the reviewer has reviewed, excluding the first-ever reviewed listing. The recommendations will then be used for performance evaluation.

2) GNN Modification and Improvement

GraphSAGE,, where the aggregation scheme uses mean,, is selected as the central GNN architecture. The proposed GraphSAGE is designed for homogeneous graphs where all nodes and edges have the same type. However, the reviewer-listing graph is heterogeneous, consisting of different kinds of nodes (reviewers and listings). To address this challenge, this paper utilizes one of the specialized functionalities offered by the PyTorch Geometric (PyG) library to handle heterogeneous graphs. This functionality ensures that the message-passing formulation in GraphSAGE is adapted to the heterogeneous nature of the graph. Specifically, the computation of message passing, and update functions considers the node or edge type.

In the original GraphSAGE paper, the model was designed for graphs with binary edges. However, in the reviewer-listing graph, it is necessary to consider weighted edges to capture the strength of the connection, such as the rating given by the reviewer to the listing. To address this, a modification is made by incorporating a customized loss, namely Rating-Weighted Triplet Ranking Loss, shown in Equation (1) below:

$$L = \sum_{(u,l)\epsilon E} \max\big(0, -(z_u, z_l) + (z_u, z_n) + (\Delta \times r)\big) \quad (1)$$

Given that a user is reviewing a listing once, a weighted edge between them exists in the graph where the rating defines the weight. The concept behind this approach is to ensure that the distance between the representations of user node u and a negative node n (a listing that the user has not reviewed) is larger than a certain margin compared to the distance between the representations of the same node u and a positive node l (a reviewed listing). A rating-weighted mechanism is introduced to overcome the issue of treating high-rating and low-rating edges interchangeably. The final margin is determined by multiplying the margin constant Δ with rating r. This results in a higher margin for higher-rating edges, requiring a more significant difference between positive and negative pairs within a triplet, as compared to another triplet with the same user node and negative node but a positive node with a lower rating. In latent space, the positive node with a higher rating is pulled closer to the user node than the positive node with a lower rating. However, both positive nodes remain considerably closer to the user node than the respective negative node [28]. This rating-weighted approach captures the user's preference for listings, allowing for personalized recommendations. The proposed GNN is named Rating-Weighted GraphSAGE. For details of model training, the number of hidden channels of the GNN model is 64. Adam algorithm is implemented as the optimizer, and the learning rate is set at 0.01. The number of epochs is set as 300.

3) Offline Simulation of Continuous Training

The offline simulation begins when initializing a retraining pipeline scheduled by a pipeline orchestration tool called Prefect at a scheduled frequency. Note that the expected frequency here is not precisely in accordance with the cadence of periodic retraining; instead, it is set to automate the retraining process. Then, the pipeline would connect with a Weights & Biases platform, which is a machine learning operations (MLOps) platform that would keep track of everything in the model retraining, including artifacts, datasets, codes, models, metrics, and so on. Each retraining run will follow precisely how the model training is done offline with the same model architecture, period of dataset split, and so on.

In each iteration of retraining, throughout the epochs, each model produced will be pushed to the model registry, which versions and keeps track of all the models, while the model that has the most minor test loss will be labeled as a contender model. If no production model exists, the contender model will be automatically pushed to production as if the data scientist deploys the best model they could train with the data up to the training time. The contender model will be benchmarked against the production model in subsequent runs. One of the crucial components of a CT pipeline of a machine learning model is offline model evaluation, which checks whether the contender model has outperformed the production model and, if so, by how much. This evaluation is essential in ensuring that the contender model meets performance standards and is ready for deployment.

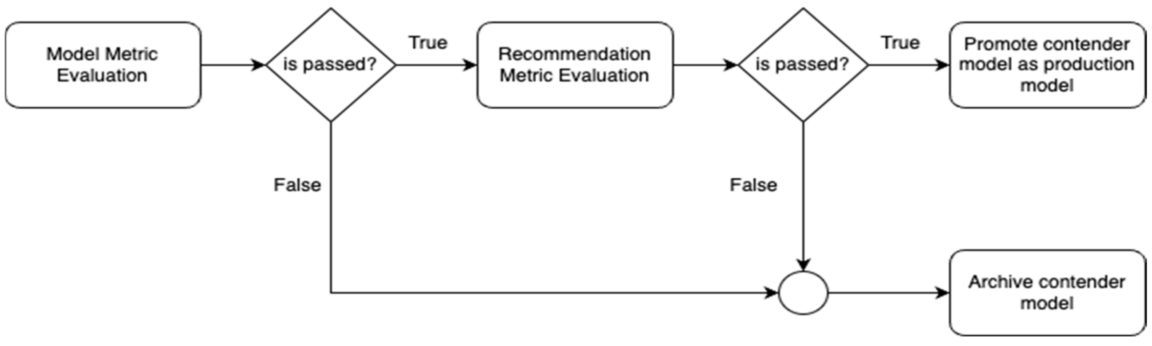

Fig. 1 Offline model evaluation

During the offline model evaluation phase, the production and contender models are assessed using the test set. The evaluation process follows the flow illustrated in Figure 1. The test loss is considered the primary metric for evaluating model performance, as it provides a simple measure. It is assumed that test loss improvements strongly correlate with recommendation metrics enhancements. If the contender model fails to outperform the production model in model metric evaluation, it is not advanced for further evaluation and is instead archived. This stringent criterion ensures that only high-quality models progress to the next stage and ultimately get deployed, thereby maintaining the overall quality of the recommender system. The primary recommendation metric used is hit-rate@5. Similar to evaluating the model metric, if the contender model fails to outperform the production model in terms of the recommendation metric, it will be archived. However, if the contender model surpasses the production model in both the model metric evaluation and the recommendation metric evaluation, it will be promoted as the new production model. Technically, this "promotion" involves updating the model version tag to "production" for the contender model, which will be retrieved by the model serving layer. It is important to note that the contender model, which is determined to be superior to the production model through offline evaluation, should ideally undergo online evaluation, such as A/B Testing, before being deployed to production.

III. Results and Discussion

A. Dataset

Two datasets are utilized: the review dataset, containing reviews provided by guests for Airbnb listings they have rented, and the listing dataset, comprising rows of listing details. Both datasets are obtained by scraping data from the Airbnb website using Python libraries -- BeautifulSoup and Selenium.

B. Evaluation of GNN-based Recommender System

1) Experimental Design: We compare the proposed GNN with the unsupervised variant of GraphSAGE regarding recommendation performance. A whole year of reviews from 24th October 2021 till 23rd October 2022 is mainly extracted for the model training and evaluation, consisting of 408596 reviews. A time-based train-test split is done on the reviews in the first place, where the first ten months of reviews are used for training, while the reviews from the two months following the split date are used for evaluating the recommender system performance. To be precise, the experiment will compute the K most relevant listings for each user of the test set using the proposed model and the two ways of generating recommendations: U2I recommendations and I2I recommendations.

Subsequently, generated recommendations are evaluated concerning the listings reviewed by each user during the first two months following August 24th, 2022, using two standard recommendation metrics: hit rate and coverage. The value of K representing the predetermined number of recommendations to be provided is set as 5. It is because the average number of rented listings per reviewer in the test set is 1, which is insufficient to amount to a decently good value of K for precision@k and recall@k, let alone measuring ranking quality.

2) Evaluation of Result: According to Table 1, Rating-Weighted GraphSAGE demonstrated remarkable improvements in U2I and I2I recommendations. In the evaluation of the U2I recommendation, Rating-Weighted GraphSAGE achieved a substantial increase of 99.88 points in hit-rate@5 compared to Unsupervised GraphSAGE. Regarding coverage, Rating-Weighted GraphSAGE outperformed the unsupervised variant with a relative increase of 446.5%. In the evaluation of the I2I recommendation, although the differences in metrics were not as significant as previously, Rating-Weighted GraphSAGE still achieved a higher hit-rate@5 (5.15% increase) and higher coverage (1.81% increase) compared to Unsupervised GraphSAGE. This improvement is considered highly significant, highlighting the superiority of the supervised setting and the use of the ranking loss function. The difference in performance between the two GraphSAGE models can be explained by the nature of the loss functions they use.

In the case of the proposed GNN, Rating-Weighted Triplet Ranking Loss is used to train the model to learn embeddings that can clearly distinguish between positive and negative examples. The loss function works by taking a triplet of embeddings (anchor, positive, negative) and minimizing the distance between the anchor and positive embeddings while maximizing the distance between the anchor and negative embeddings. This encourages the model to learn embeddings that are close together for positive pairs and far apart for opposing pairs [29]. This is well advocated in the recommender system as the recommender system should recognize well the user preferences, such as what the user prefers and dislikes [30]. Rating-Weighted GraphSAGE has likely captured the preference towards listings of the majority of users during training.

On the other hand, an unsupervised loss function does not rely on labeled data for training. Instead, it focuses on learning representations that capture the underlying structure of the data. In the case of the GraphSAGE model, an unsupervised loss function is used to learn embeddings that capture the local and global structure of the nodes with graph, assuming that users and listings closer to each other would have shared similar preferences. However, the significantly poor recommendation performance doesn't justify it. The reason why the GraphSAGE model that uses a triplet ranking loss outperforms the unsupervised variant substantially is that the triplet ranking loss is specifically tailored to optimize for predicting the listing nodes that are most likely rented by the users. On the contrary, an unsupervised loss function may not be as effective in optimizing for such an objective, since it focuses on learning representations that capture the structure of the nodes within graph, rather than explicitly distinguishing between positive and negative examples. Overall, the choice of loss function can significantly impact the performance of a GNN model, and it is essential to choose a loss function that is appropriate for the specific task at hand. In the case of the GraphSAGE model, the triplet ranking loss is a better choice.

Table. 1 Result of recommender model performances

Type of Recommendation	**Model Used**	**Metric**	
		Hit-Rate@5 (%)	**Coverage (%)**
U2I	Rating-Weighted GraphSAGE	99.88	98.15
	Unsupervised GraphSAGE	0.70	17.96
I2I	Rating-Weighted GraphSAGE	5.85	25.65
	Unsupervised GraphSAGE	0.70	23.84

As shown in Figure 2 and Figure 3, due to the limited availability of ground truth data, the increase in K has minimal impact on the hit rate. Hence, the difference in hit rate between each K and the subsequent one is negligible for both models. However, the same cannot be said for coverage. Intuitively, an increase in K implies that the recommender system has more opportunities to recommend additional listings, thereby increasing the number of unique recommended listings. In summary, Rating-Weighted GraphSAGE consistently outperforms the unsupervised variant regardless of the value of K.

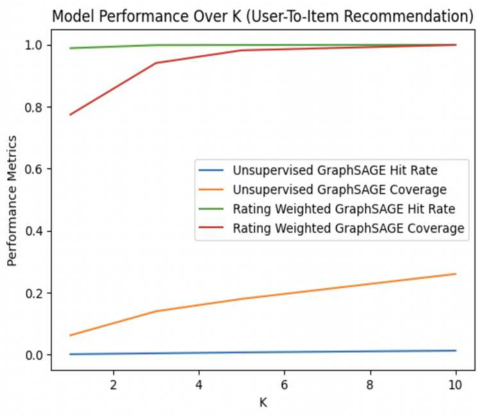

Fig. 2 Recommendation metrics over K by different models using U2I Recommendation

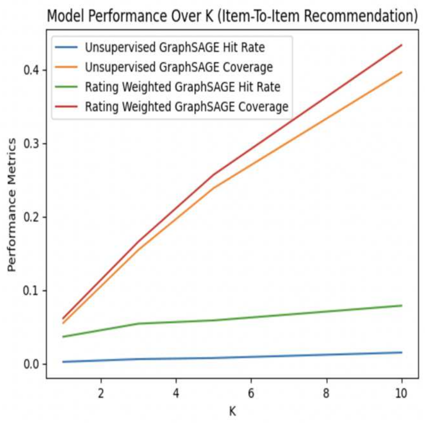

Fig. 3 Recommendation metrics over K by different models using I2I Recommendation

C. Evaluation of Continuous Training of Recommender System

1) Experimental Design: The offline simulation is set to simulate periodic retraining that begins on October 23, 2016, with subsequent retraining runs from the same date each year until October 23, 2022. To evaluate the effectiveness of Continuous Training, the experiment outcomes will be illustrated and assessed by analyzing the performance of both the production and contender models over time. The same recommendations, metrics, and aspects are used here.

2) Evaluation of Result: In Figure 4, for U2I recommendations, it is clearly seen that the performance of the contender model is superior to the initial production model most of the time. However, there is no comparability between the contender model and production model in I2I recommendations, as both perform similarly. Also, the performance of the production model degrades over time, which could be detrimental to the business as it could not capture user preference, reducing the user retention rate.

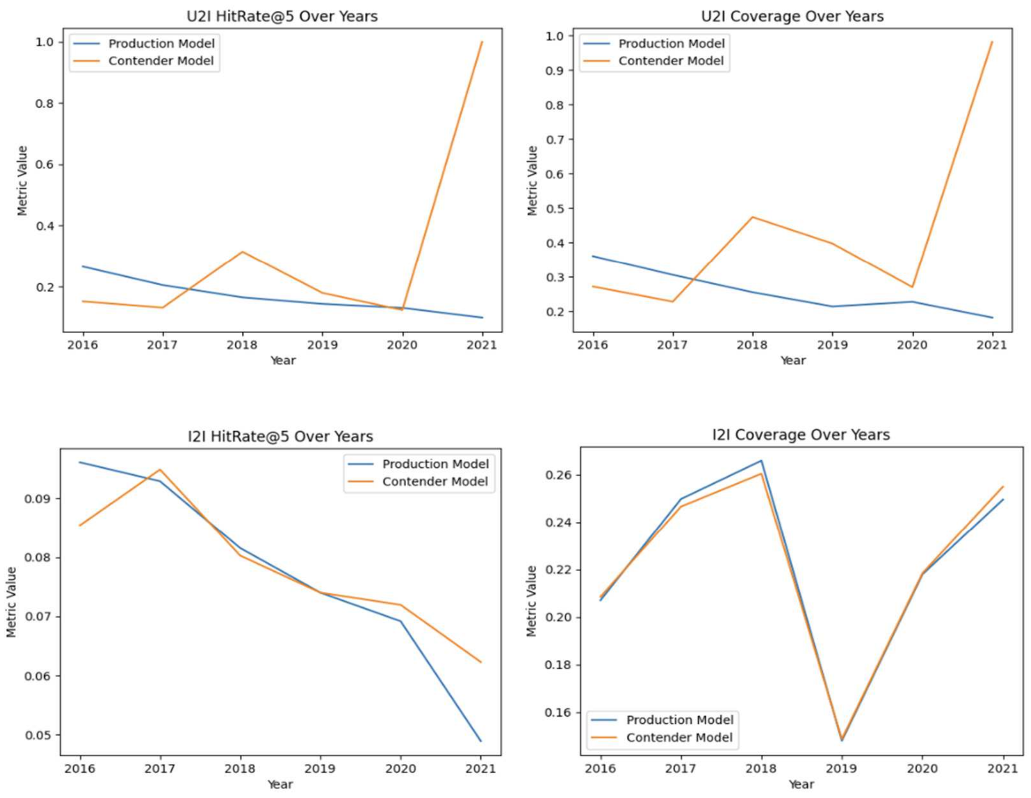

Fig. 4 Recommendation metrics over the years by different models

One key advantage of Continuous Training is its ability to leverage the latest data, allowing the model to learn and improve its performance over time continuously. Unlike the production model, which remains static and does not update itself regularly, the contender model benefits from the continuous inflow of fresh data. The recommender system becomes more adaptable and responsive to evolving conditions by incorporating Continuous Training. For instance, if there is a sudden shift in user behavior or preferences, the contender model can quickly fine-tune its recommending capability based on the latest data.

IV. Conclusion

In conclusion, this paper incorporated a customized loss named Rating-Weighted Triplet Ranking Loss into GraphSAGE for the recommender system of Airbnb listings. By employing the loss above function, the GraphSAGE model can accommodate the bipartite graph made of user-listing interactions and consider the weightage of different ratings given by the user to curate personalized recommendations for the users. It significantly outperforms the unsupervised variant of GraphSAGE in both U2I recommendations and I2I recommendations. For the effectiveness of Continuous Training, an offline simulation is conducted, and the result proves that the performance of the contender model could outperform the production model most of the time.

One limitation of the proposed GNN is that the new users or new listings must have at least one interaction with others to exist within the graph and be converted into embeddings for downstream usage. This could be a significant drawback of the proposed recommender system, which will be ineffective when recommending listings to cold users. There are approaches like link prediction, predicting cold user's embeddings by leveraging node embeddings, and so on, which could potentially overcome the aforementioned drawback. On the side of MLOps, a CI/CD system can be set up in the Continuous Training pipeline to enable changes to be tested and built, as well as deployment in the machine learning systems in a timely, reproducible, and secure manner by introducing automation into the lifecycle. This would then help make the pipeline more robust, less prone to unexpected error, and even reduce the time of delivery to iterate faster on improvement on the overall system.

This paper drives the study towards a personalized recommender system by incorporating users' preferences for items into graph learning, which resembles how the level of connectivity/relationship between entities correlates with the closeness between items in the real world.

References

[1] Y. F. Lim, S. C. Haw, and E. A. Anaam, "Hybrid-based Recommender System for Online Shopping: A Review," *Journal of Engineering Technology and Applied Physics*, vol. 5, no. 1, pp. 12–34, 2023, doi: 10.33093/jetap.2023.5.1.

[2] S. C. Haw, L. J. Chew, K. Ong, K. W. Ng, P. Naveen, and E. A. Anaam, "Content-based Recommender System with Descriptive Analytics," *Journal of System and Management Sciences*, vol. 12, no. 5, pp. 105–120, 2022, doi: 10.33168/JSMS.2022.0507.

[3] J. Yu, H. Yin, X. Xia, T. Chen, J. Li, and Z. Huang, "Self-Supervised Learning for Recommender Systems: A Survey," Mar. 2022, [Online]. Available: http://arxiv.org/abs/2203.15876

[4] S. Munikoti, D. Agarwal, L. Das, M. Halappanavar, and B. Natarajan, "Challenges and Opportunities in Deep Reinforcement Learning with Graph Neural Networks: A Comprehensive Review of Algorithms and Applications," Jun. 2022, [Online]. Available: http://arxiv.org/abs/2206.07922

[5] J. Chen, H. Dong, X. Wang, F. Feng, M. Wang, and X. He, "Bias and Debias in Recommender System: A Survey and Future Directions," *ACM Trans Inf Syst*, vol. 41, no. 3, Feb. 2023, doi: 10.1145/3564284.

[6] P. Liu, L. Zhang, and J. A. Gulla, "Pre-train, Prompt, and Recommendation: A Comprehensive Survey of Language Modelling Paradigm Adaptations in Recommender Systems," Feb. 2023, [Online]. Available: http://arxiv.org/abs/2302.03735

[7] Z. Y. Poo, C. Y. Ting, Y. P. Loh, and K. I. Ghauth, "Multi-Label Classification with Deep Learning for Retail Recommendation," *Journal of Informatics and Web Engineering*, vol. 2, no. 2, pp. 218-232, 2023. https://doi.org/10.33093/jiwe.2023.2.2.16

[8] M. Casillo, F. Colace, D. Conte, M. Lombardi, D. Santaniello, and C. Valentino, "Context-aware recommender systems and cultural heritage: a survey," *J Ambient Intell Humaniz Comput*, vol. 14, no. 4, pp. 3109–3127, Apr. 2023, doi: 10.1007/s12652-021-03438-9.

[9] M. Asad, S. Shaukat, E. Javanmardi, J. Nakazato, and M. Tsukada, "A Comprehensive Survey on Privacy-Preserving Techniques in Federated Recommendation Systems," *Applied Sciences (Switzerland)*, vol. 13, no. 10, May 2023, doi: 10.3390/app13106201.

[10] C. Gao *et al.*, "A Survey of Graph Neural Networks for Recommender Systems: Challenges, Methods, and Directions," *ACM Transactions on Recommender Systems*, vol. 1, no. 1, pp. 1–51, Mar. 2023, doi: 10.1145/3568022.

[11] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," Jun. 2017, [Online]. Available: http://arxiv.org/abs/1706.02216

[12] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Association for Computing Machinery, Jul. 2018, pp. 974–983. doi: 10.1145/3219819.3219890.

[13] J. Wang, P. Huang, H. Zhao, Z. Zhang, B. Zhao, and D. L. Lee, "Billion-scale commodity embedding for E-commerce recommendation in Alibaba," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Association for Computing Machinery, Jul. 2018, pp. 839–848. doi: 10.1145/3219819.3219869.

[14] C.-M. Chen, C.-J. Wang, M.-F. Tsai, and Y.-H. Yang, "Collaborative Similarity Embedding for Recommender Systems," Feb. 2019, [Online]. Available: http://arxiv.org/abs/1902.06188

[15] J. Sun *et al.*, "Multi-Graph Convolution Collaborative Filtering," Jan. 2020, [Online]. Available: http://arxiv.org/abs/2001.00267

[16] H. Wang, D. Lian, and Y. Ge, "Binarized Collaborative Filtering with Distilling Graph Convolutional Networks," Jun. 2019, [Online]. Available: http://arxiv.org/abs/1906.01829

[17] K. Liu, F. Xue, X. He, D. Guo, and R. Hong, "Joint Multi-grained Popularity-aware Graph Convolution Collaborative Filtering for Recommendation," Oct. 2022, doi: 10.1109/tcss.2022.3151822.

[18] M. Zhang and Y. Chen, "Inductive Matrix Completion Based on Graph Neural Networks," Apr. 2019, [Online]. Available: http://arxiv.org/abs/1904.12058

[19] A. Pande, K. Ni, and V. Kini, "SWAG: Item Recommendations using Convolutions on Weighted Graphs," Nov. 2019, doi: 10.1109/BigData47090.2019.9005633.

[20] A. Saravanou, F. Tomasi, R. Mehrotra, and M. Lalmas, "Multi-Task Learning of Graph-Based Inductive Representations of Music Content," 2021. [Online]. Available: https://www.spotify.com

[21] P. Veličkovi´veličkovi´c, G. Cucurull, A. Casanova, A. Romero, P. Lì, and Y. Bengio, "Graph Attention Networks," 2017.

[22] W. Fan *et al.*, "Graph neural networks for social recommendation," in *The Web Conference 2019 - Proceedings of the World Wide Web Conference, WWW 2019*, Association for Computing Machinery, Inc, May 2019, pp. 417–426. doi: 10.1145/3308558.3313488.

[23] C. Xu *et al.*, "Graph Contextualized Self-Attention Network for Session-based Recommendation," 2019.

[24] X. Wang, X. He, Y. Cao, M. Liu, and T. S. Chua, "KGAT: Knowledge graph attention network for recommendation," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Association for Computing Machinery, Jul. 2019, pp. 950–958. doi: 10.1145/3292500.3330989.

[25] A. Sankar, Y. Liu, J. Yu, and N. Shah, “Graph neural networks for friend ranking in large-scale social platforms,” in *The Web Conference 2021 - Proceedings of the World Wide Web Conference, WWW 2021*, Association for Computing Machinery, Inc, Apr. 2021, pp. 2535–2546. doi: 10.1145/3442381.3450120.

[26] J. Chen, Y. Cao, F. Zhang, P. Sun, and K. Wei, “Sequential Intention-aware Recommender based on User Interaction Graph,” in *ICMR 2022 - Proceedings of the 2022 International Conference on Multimedia Retrieval*, Association for Computing Machinery, Inc, Jun. 2022, pp. 118–126. doi: 10.1145/3512527.3531390.

[27] K.-M. Kim *et al.*, “Tripartite Heterogeneous Graph Propagation for Large-scale Social Recommendation,” Jul. 2019, [Online]. Available: http://arxiv.org/abs/1908.02569

[28] C. Park, D. Kim, J. Han, and H. Yu, “Unsupervised Attributed Multiplex Network Embedding,” Nov. 2019, [Online]. Available: http://arxiv.org/abs/1911.06750

[29] S. Wang *et al.*, “Graph Learning based Recommender Systems: A Review,” May 2021, [Online]. Available: http://arxiv.org/abs/2105.06339

[30] A. J. Bose, A. Jain, P. Molino, and W. L. Hamilton, “Meta-Graph: Few Shot Link Prediction via Meta-Learning,” Dec. 2019, [Online]. Available: http://arxiv.org/abs/1912.09867.