

Linear-Time Graph Neural Networks for Scalable Recommendations

Jiahao Zhang*
The Hong Kong Polytechnic
University
22037278r@connect.polyu.hk

Rui Xue*
North Carolina State University
rxue@ncsu.edu

Wenqi Fan[†]
The Hong Kong Polytechnic
University
wenqi.fan@polyu.edu.hk

Xin Xu
The Hong Kong Polytechnic
University
xin.xu@polyu.edu.hk

Qing Li
The Hong Kong Polytechnic
University
qing-prof.li@polyu.edu.hk

Jian Pei
Duke University
j.pei@duke.edu

Xiaorui Liu
North Carolina State University
xliu96@ncsu.edu

ABSTRACT

In an era of information explosion, recommender systems are vital tools to deliver personalized recommendations for users. The key of recommender systems is to forecast users' future behaviors based on previous user-item interactions. Due to their strong expressive power of capturing high-order connectivities in user-item interaction data, recent years have witnessed a rising interest in leveraging Graph Neural Networks (GNNs) to boost the prediction performance of recommender systems. Nonetheless, classic Matrix Factorization (MF) and Deep Neural Network (DNN) approaches still play an important role in real-world large-scale recommender systems due to their scalability advantages. Despite the existence of GNN-acceleration solutions, it remains an open question whether GNN-based recommender systems can scale as efficiently as classic MF and DNN methods. In this paper, we propose a Linear-Time Graph Neural Network (LTGNN) to scale up GNN-based recommender systems to achieve comparable scalability as classic MF approaches while maintaining GNNs' powerful expressiveness for superior prediction accuracy. Extensive experiments and ablation studies are presented to validate the effectiveness and scalability of the proposed algorithm. Our implementation based on PyTorch is available ¹.

CCS CONCEPTS

• Information systems → Collaborative filtering.

KEYWORDS

Collaborative Filtering, Recommender Systems, Graph Neural Networks, Scalability.

ACM Reference Format:

Jiahao Zhang, Rui Xue, Wenqi Fan, Xin Xu, Qing Li, Jian Pei, and Xiaorui Liu. 2024. Linear-Time Graph Neural Networks for Scalable Recommendations. In *Proceedings of the ACM Web Conference 2024 (WWW '24)*, May 13–17, 2024.

*Equal contributions.

[†]Corresponding author: Wenqi Fan, Department of Computing, and Department of Management and Marketing, The Hong Kong Polytechnic University.

¹ <https://github.com/QwQ2000/TheWebConf24-LTGNN-PyTorch>

WWW '24, May 13–17, 2024, Singapore, Singapore.

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *Proceedings of the ACM Web Conference 2024 (WWW '24)*, May 13–17, 2024, Singapore, Singapore, <https://doi.org/10.1145/3589334.3645486>.

2024, Singapore, Singapore. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3589334.3645486>

1 INTRODUCTION

In an era of information explosion, recommender systems are playing an increasingly critical role in enriching users' experiences with various online applications, due to their remarkable abilities in providing personalized item recommendations. The main objective of recommender systems is to predict a list of candidate items that are likely to be clicked or purchased by capturing users' potential interests from their historical behaviors [23]. A prevailing technique in modern recommender systems is collaborative filtering (CF), which leverages the patterns across similar users and items to predict the users' preferences.

As one of the most representative CF methods, matrix factorization (MF) models are introduced to represent users and items in a low-dimensional embedding space by encoding the user-item interactions matrix. After the emergence of MF models, a remarkable stream of literature has made great efforts to improve the expressive capability of user and item representations. As discussed in many previous studies [35, 42, 43], we can divide these attempts into two branches based on their modeling ability of user-item interaction graphs. First, most early approaches in collaborative filtering focus on the *local connectivity* of users and items, such as item similarity models [24, 31] and deep neural networks (DNNs) [25, 47]. Second, due to the intrinsic limitation of modeling high-order connectivity in early CF models, recent years have witnessed a rising interest in graph neural networks (GNNs) in recommendations. To be specific, GNN-based CF models encode both *local and long-range collaborative signals* into user and item representations by iteratively aggregating embeddings along local neighborhood structures in the interaction graph [13, 23, 42], showing their superior performance in modeling complex user-item interaction graphs.

Despite the promising potential of GNNs in modeling high-order information in interaction graphs, GNN-based CF models have not been widely employed in industrial-level applications majorly due to their scalability limitations [22, 49]. In fact, classic CF models like MF and DNNs are still playing major roles in real-world applications due to their computational advantages, especially in large-scale industrial recommender systems [7, 10]. In particular, the computation complexity for training these conventional CF models, such as MF and DNNs, is *linear* to the number of user-item

interactions in the interaction matrix, while the computation complexity of training GNN-based CF models are *exponential* to the number of propagation layers or *quadratic* to the number of edges (as will be discussed in Section 2.3).

In web-scale recommender systems, the problem size can easily reach a billion scale towards the numbers of nodes and edges in the interaction graphs [28, 39]. Consequently, it is essential that scalable algorithms should have nearly linear or sub-linear complexity with respect to the problem size. Otherwise, they are infeasible in practice due to the unaffordable computational cost [38]. While numerous efforts have continued to accelerate the training of GNN-based recommender systems, including two main strategies focusing on neighbor sampling [34, 49] and design simplification [23, 44], none of them can achieve the linear complexity for GNN-based solutions, leading to inferior efficiency in comparison with conventional CF methods such as MF and DNNs. There is still an open question in academia and industry: *Whether GNN-based recommendation models can scale linearly as the classic MF and DNN methods, while exhibiting long-range modeling abilities and strong prediction performance.*

In this paper, our primary objective revolves around 1) *preserving the strong expressive capabilities inherent in GNNs* while simultaneously 2) *achieving a linear computation complexity* that is comparable to traditional CF models like MF and DNNs. However, it is highly non-trivial to pursue such a scalable GNN design, since the expressive power of high-order collaborative signals lies behind the number of recursive aggregations (i.e., GNN layers). Moreover, the embedding aggregation over a large number of neighbors is highly costly. To achieve a linear computation complexity, we propose a novel implicit graph modeling for recommendations with the *single-layer propagation* model design and an *efficient variance-reduced neighbor sampling* algorithm. Our contributions can be summarized as follows:

- We provide a critical complexity analysis and comparison of widely used collaboration filtering approaches, and we reveal their performance and efficiency bottlenecks.
- We propose a novel GNN-based model for large-scale collaborative filtering in recommendations, namely LTGNN (Linear Time Graph Neural Networks), which only incorporates *one propagation layer* while preserving the capability of capturing long-range collaborative signals.
- To handle large-scale user-item interaction graphs, we design an efficient and improved *variance-reduced neighbor sampling* strategy for LTGNN to significantly reduce the neighbor size in embedding aggregations. The random error caused by neighbor sampling is efficiently tackled by our improved variance reduction technique.
- We conduct extensive comparison experiments and ablation studies on three real-world recommendation datasets, including a large-scale dataset with millions of users and items. The experiment results demonstrate that our proposed LTGNN significantly reduces the training time of GNN-based CF models while preserving the recommendation performance on par with previous GNN models. We also perform detailed time complexity analyses to show our superior efficiency.

2 PRELIMINARIES

This section presents the notations used in this paper, and then briefly introduces preliminaries about GNN-based recommendations and the computation complexity of popular CF models.

2.1 Notations and Definitions

In personalized recommendations, the historical user-item interactions can be naturally represented as a bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the node set \mathcal{V} includes n user nodes $\{v_1, \dots, v_n\}$ and m item nodes $\{v_{n+1}, \dots, v_{n+m}\}$, and the edge set $\mathcal{E} = \{e_1, \dots, e_{|\mathcal{E}|}\}$ consists of undirected edges between user nodes and item nodes. It is clear that the number of undirected edges $|\mathcal{E}|$ equals to the number of observed user-item interactions $|\mathcal{R}^+|$ in the training data (i.e., $|\mathcal{E}| = |\mathcal{R}^+|$). The graph structure of \mathcal{G} can be denoted as the adjacency matrix $A \in \mathbb{R}^{(n+m) \times (n+m)}$, and its diagonal degree matrix are denoted as D . The normalized adjacency matrix with self-loops is defined as $\tilde{A} = (D + I)^{-\frac{1}{2}}(A + I)(D + I)^{-\frac{1}{2}}$. We use $\mathcal{N}(v)$ to denote the set of neighboring nodes of a node v , including v itself. In addition, the trainable embeddings of user and item nodes in graph \mathcal{G} are denoted as $E = [e_1, \dots, e_n, e_{n+1}, \dots, e_{n+m}]^T \in \mathbb{R}^{(n+m) \times d}$, where its first n rows are d -dimensional user embeddings and its $n + 1$ to $n + m$ rows are d -dimensional item embeddings.

In the training process of GNN-based collaborative filtering models, we use $(E_l^k)_B$ or $(e_l^k)_v$ to denote an embedding matrix or a single embedding vector, where k is the index of training iterations and l is the index of propagation layers. The subscript $(\cdot)_B$ or $(\cdot)_v$ denotes the embedding for a batch of nodes B or a single node v .

2.2 Mini-batch Training

To provide effective item recommendations from user-item interactions, a typical training objective is the pairwise loss function. We take the most widely adopted BPR [37] loss as an example:

$$\mathcal{L}_{BPR} = \sum_{(u,i,j) \in \mathcal{O}} -\ln \sigma(\hat{y}_{u,i} - \hat{y}_{u,j}), \quad (1)$$

where $\mathcal{O} = \{(u, i, j) | (u, i) \in \mathcal{R}^+, (u, j) \in \mathcal{R}^-\}$ denotes the pairwise training data. \mathcal{R}^+ and \mathcal{R}^- denotes observed and unobserved user-item interactions. In practice, the training data \mathcal{O} is hardly leveraged in a full-batch setting due to the large number of user-item interactions [23, 42]. Therefore, mini-batch training is a common choice that splits the original data \mathcal{O} into multiple components $\Omega = \{\mathcal{O}_{(u_1, i_1)}, \mathcal{O}_{(u_2, i_2)}, \dots, \mathcal{O}_{(u_{|\mathcal{R}^+|}, i_{|\mathcal{R}^+|})}\}$, where $\mathcal{O}_{(u_r, i_r)} = \{(u_r, i_r, j) | (u_r, j) \in \mathcal{R}^-\}$ contains all the training data including positive and negative samples for a specific interaction (u_r, i_r) . In each training iteration, we first sample B interactions from \mathcal{R}^+ , which is denoted as $\hat{\mathcal{R}}^+$, satisfying $|\hat{\mathcal{R}}^+| = B$. Afterward, we create the training data for $\hat{\mathcal{R}}^+$ by merging the corresponding components in Ω , which can be denoted as $\hat{\mathcal{O}}(\hat{\mathcal{R}}^+) = \bigcup_{(u,i) \in \hat{\mathcal{R}}^+} \mathcal{O}_{(u,i)}$. Thus, the mini-batch training objective can be formalized as follows:

$$\hat{\mathcal{L}}_{BPR}(\hat{\mathcal{R}}^+) = \sum_{(u,i,j) \in \hat{\mathcal{O}}(\hat{\mathcal{R}}^+)} -\ln \sigma(\hat{y}_{u,i} - \hat{y}_{u,j}). \quad (2)$$

In each training epoch, we iterate over all user-item interactions in \mathcal{R}^+ , so the mini-batch training objective $\hat{\mathcal{L}}_{BPR}$ needs to be evaluated for $|\mathcal{R}^+|/B$ times (i.e., $|\mathcal{E}|/B$ times).

2.3 GNNs and MF for Recommendations

In this subsection, we will briefly review MF and two representative GNN-based recommendation models, including LightGCN [23] and PinSAGE [49], and discuss their computation complexity.

LightGCN. Inspired by the graph convolution operator in GCN [30] and SGC [44], LightGCN [23] iteratively propagates the user embedding $(\mathbf{e}_l)_u$ and item embedding $(\mathbf{e}_l)_i$ as follows:

$$(\mathbf{e}_{l+1})_u = \frac{1}{\sqrt{|\mathcal{N}(u)|}} \sum_{i \in \mathcal{N}(u)} \frac{1}{\sqrt{|\mathcal{N}(i)|}} (\mathbf{e}_l)_i, \quad (3)$$

$$(\mathbf{e}_{l+1})_i = \frac{1}{\sqrt{|\mathcal{N}(i)|}} \sum_{u \in \mathcal{N}(i)} \frac{1}{\sqrt{|\mathcal{N}(u)|}} (\mathbf{e}_l)_u. \quad (4)$$

The embedding propagation of LightGCN can be re-written in matrix form as follows:

$$E_{l+1} = \tilde{A}E_l, \quad \forall l = 0, \dots, L-1 \quad (5)$$

$$Y = \frac{1}{L+1} \sum_{l=0}^L E_l, \quad (6)$$

where L denotes the number of GNN layers, and Y denotes the model output of LightGCN with layer-wise combination. As LightGCN computes full embedding propagation in Eq. (5) for L times to capture L -hop neighborhood information, the computation complexity of LightGCN in one training iteration is $O(L|\mathcal{E}|d)$ with the support of sparse matrix multiplications. Thus, the computation complexity for one training epoch is $O(\frac{|\mathcal{E}|}{B} \cdot L|\mathcal{E}|d) = O(\frac{1}{B}L|\mathcal{E}|^2d)$, where $\frac{|\mathcal{E}|}{B}$ is the number of training iterations in one epoch.

PinSAGE. The embedding propagation in LightGCN aggregates all the neighbors for a user or an item, which is less compatible with Web-scale item-to-item recommender systems. Another important embedding propagation rule in GNN-based recommendation is proposed in PinSAGE:

$$(\mathbf{n}_{l+1})_u = \text{Aggregate}(\{\text{ReLU}(\mathbf{Q} \cdot (\mathbf{e}_l)_v + \mathbf{q}) \mid v \in \hat{\mathcal{N}}(u)\}), \quad (7)$$

$$(\mathbf{e}_{l+1})_u = \text{Normalize}(\mathbf{W} \cdot \text{Concat}[(\mathbf{e}_l)_u; (\mathbf{n}_{l+1})_u] + \mathbf{w}), \quad (8)$$

where $\mathbf{Q}, \mathbf{q}, \mathbf{W}, \mathbf{w}$ are trainable parameters, and $\hat{\mathcal{N}}(u)$ denotes the randomly sampled neighbors for node u . If PinSAGE constantly samples D random neighbors for each node at each layer, and the sampled B edges have n_B target nodes without repetition, the computation complexity in each training iteration is $O(n_B D^L d^2)$ as discussed in previous studies [46]. Thus, the time complexity in the entire training epoch is $O(\frac{|\mathcal{E}|}{B} \cdot n_B D^L d^2) = O(|\mathcal{E}| D^L d^2)$, as n_B and B shares the same order. Moreover, the neighbor sampling in PinSAGE incurs large approximation errors that impact the prediction performance.

Matrix Factorization (MF). Matrix factorization and its neural variant NCF [25] are simple but strong baselines for recommendations at scale. Given learnable user embedding \mathbf{p}_u and item embedding \mathbf{q}_i , MF models their interaction directly by inner product as $\hat{y}_{u,i} = \mathbf{p}_u^T \mathbf{q}_i$, while NCF models the interaction by deep neural networks as follows:

$$\mathbf{e}_L = \phi(\mathbf{W}_L \cdots \phi(\mathbf{W}_2 \phi(\mathbf{W}_1 \begin{bmatrix} \mathbf{p}_u \\ \mathbf{q}_i \end{bmatrix} + \mathbf{b}_1) + \mathbf{b}_2) \cdots) + \mathbf{b}_L, \quad (9)$$

$$\hat{y}_{u,i} = \sigma(\mathbf{h}^T \mathbf{e}_L), \quad (10)$$

where \mathbf{W}, \mathbf{b} and \mathbf{h} are trainable parameters, and ϕ is a non-linear activation function. In each training iteration, the computation complexity for MF and NCF is $O(Bd)$ and $O(BLd^2)$, which stands for the complexity of dot products and MLPs, respectively. Thus, the time complexity in each training epoch for MF and NCF is $O(|\mathcal{E}|d)$ and $O(|\mathcal{E}|Ld^2)$.

Inefficiency of GNNs. In comparison with conventional MF models, GNNs' inefficiency lies behind their non-linear complexity with respect to the number of edges $|\mathcal{E}|$ or layers L . For example, the time complexity for LightGCN is $O(\frac{1}{B}L|\mathcal{E}|^2d)$, which grows quadratically with $|\mathcal{E}|$, and PinSAGE has a complexity of $O(|\mathcal{E}|D^L d^2)$, which grows exponentially with L . In this paper, we pursue a *linear-time* design for GNNs, which means the time complexity of our proposed model is expected to be $O(C|\mathcal{E}|d)$, where C is a small constant (e.g., $C = Ld$ for NCF).

3 THE PROPOSED METHOD

The scalability issue of GNN-based recommendation models inspires us to pursue a more efficient algorithm design with linear computation complexities. However, to reduce the computation complexity while preserving GNNs' long-range modeling ability, we need to overcome **two main challenges**:

- **Layer and expressiveness:** Increasing the number of layers L in GNNs can capture long-range dependencies, but the complexity (e.g., $O(|\mathcal{E}|D^L d^2)$ in PinSAGE) can hardly be linear when L is large.
- **Neighbor aggregation and random error:** The number of neighbors D aggregated for each target node in a GNN layer substantially affects both computation cost and approximation error. Aggregating all neighbors (e.g., LightGCN) is costly, while aggregation with random sampling incurs large errors (e.g., PinSAGE).

To address these challenges, we propose implicit graph modeling in Section 3.1 to reduce the layer number L significantly, and a variance-reduced neighbor sampling strategy in Section 3.2 to lower the neighbor aggregation cost for high-degree nodes. We carefully handle the numerical and random errors of our designs, and ensure the strong expressiveness of our proposed LTGNN.

3.1 Implicit Modeling for Recommendations

Personalized PageRank [32] is a classic approach for the measurement of the proximity between nodes in a graph. It is adopted by a popular GNN model, PPNP (Personalized Propagation of Neural Predictions) [19], to propagate node embeddings according to the personalized PageRank matrix:

$$E_{PPNP}^k = \alpha \left(\mathbf{I} - (1 - \alpha) \tilde{\mathbf{A}} \right)^{-1} E_{in}^k, \quad (11)$$

where α is the teleport factor and E_{in}^k is the input node embedding. Due to the infeasible cost of matrix inversion, APPNP approximates this by L propagation layers:

$$E_0^k = E_{in}^k, \quad (12)$$

$$E_{l+1}^k = (1 - \alpha) \tilde{\mathbf{A}} E_l^k + \alpha E_{in}^k, \quad \forall l = 0, \dots, L-1 \quad (13)$$

such that it can capture the L -hop high-order information in the graph without suffering from over-smoothing due to the teleport

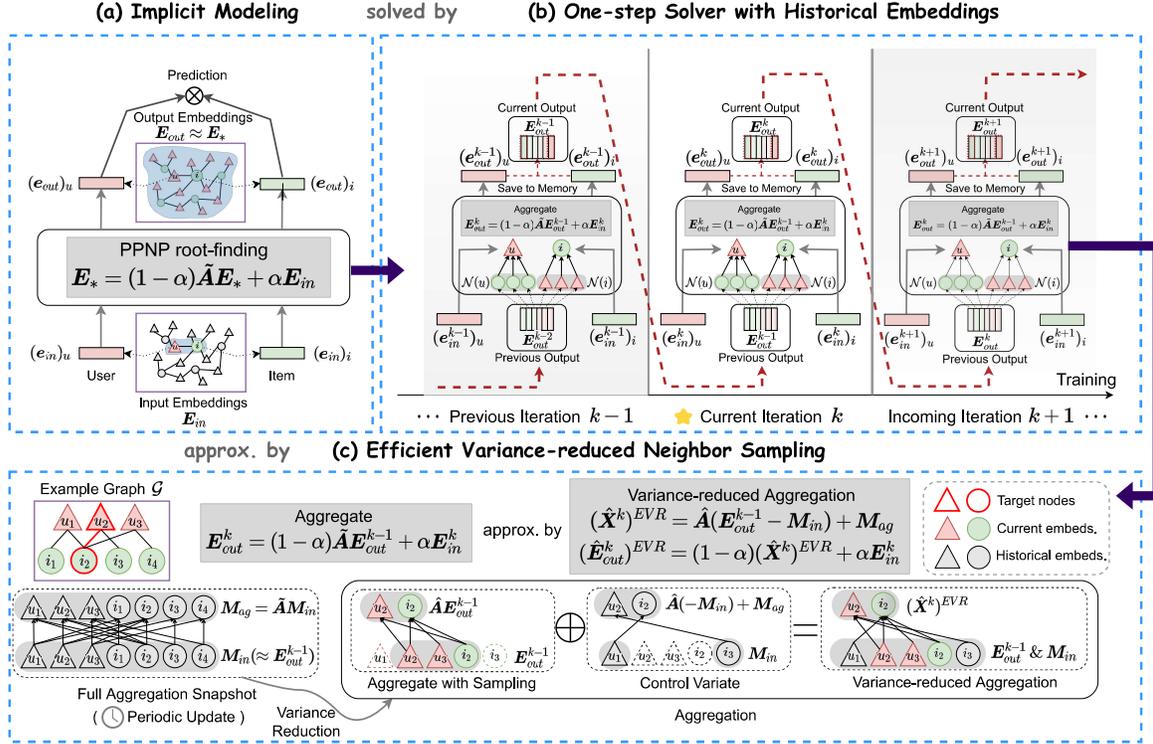


Figure 1: An illustration of our model architecture. (a) The forward process of our model aims to solve the PPNP fixed-point equation, which expresses an equilibrium state of the embedding propagations, and can be used to capture long-range relations between any pair of nodes regardless of their distance. (b) The PPNP fixed-point equation is solved with a single forward propagation layer, which leverages the historical output embeddings in previous training iterations. (c) The process of efficient variance-reduced neighbor sampling in LTGNN.

term αE_{in}^k . LightGCN exhibits a close relation with APPNP although the embedding from different propagation layers is averaged with a different weight (see the analysis in Section 3.2 of [23]). However, like most GNNs, both APPNP and LightGCN suffer from scalability issues due to the multi-layer recursive feature aggregations, which greatly limit their applications in large-scale recommendations.

Motivated by the previous success of Implicit Deep Learning [9] and Implicit GNNs [20, 33, 48], we propose implicit modeling for graph-based recommendations (as shown in Fig. 1(a)), which directly computes the fixed point E_*^k of the embedding propagation in Eq. (13). Particularly, the output of our implicit model can be formalized as the solution of a linear system:

$$E_{out}^k = \text{RootFind}(E_*^k), \quad \text{s.t.} \quad E_*^k = (1 - \alpha)\tilde{A}E_*^k + \alpha E_{in}^k, \quad (14)$$

where the relation between output embedding E_{out}^k and input embedding E_{in}^k is implicitly defined by a root-finding process of the fixed-point equation. Formulating graph-based recommendations implicitly with a fixed-point equation has two advantages: 1) The fixed-point E_*^k models the equilibrium state of embedding propagations, which is equivalent to the extreme state under an infinite number of propagations, effectively capturing the long-range node dependencies in graphs. 2) This implicit modeling provides flexibility for the GNN design, as we can use any equation solver to acquire E_*^k instead of stacking multiple GNN layers.

Specifically, to pave the way to linear-time computation, we propose to solve this fixed-point equation by a single forward propagation layer, as shown in Fig. 1(b):

$$E_{out}^k = (1 - \alpha)\tilde{A}E_{out}^{k-1} + \alpha E_{in}^k, \quad (15)$$

where E_{out}^{k-1} is the historical output embeddings at previous training iteration $k-1$ and serves as a better initialization for the fixed-point solver. This single-layer design is ultra-efficient compared with multi-layer embedding propagations but still captures multi-hop neighbor information through information accumulation across training iterations.

The backward propagation of implicit models is independent of the forward computation [9, 20, 48]. Given the gradient from the output embedding layer $\frac{\partial \mathcal{L}}{\partial E_{out}^k}$, the gradient of E_{in}^k can be computed based on the fixed-point equation in Eq. (14):

$$\frac{\partial \mathcal{L}}{\partial E_{in}^k} = \alpha \frac{\partial \mathcal{L}}{\partial E_{out}^k} \left(I - (1 - \alpha)\tilde{A} \right)^{-1}. \quad (16)$$

Due to the prohibitively high dimensionality of the adjacency matrix, computing its inverse is infeasible. Therefore, we propose to approximate this gradient by a single backward propagation layer:

$$\frac{\partial \mathcal{L}}{\partial E_{in}^k} = (1 - \alpha)\tilde{A} \frac{\partial \mathcal{L}}{\partial E_{in}^{k-1}} + \alpha \frac{\partial \mathcal{L}}{\partial E_{out}^k}, \quad (17)$$

where $\frac{\partial \mathcal{L}}{\partial E_{in}^{k-1}}$ is the historical gradient of input embedding from iteration $k-1$ and serves as a better initialization for the fixed-point

solver. In summary, the forward and backward computation of our single-layer GNN are formulated as:

$$\text{Forward: } E_{out}^k = (1 - \alpha)\tilde{A}E_{out}^{k-1} + \alpha E_{in}^k, \quad (18)$$

$$\text{Backward: } \frac{\partial \mathcal{L}}{\partial E_{in}^k} = (1 - \alpha)\tilde{A} \frac{\partial \mathcal{L}}{\partial E_{in}^{k-1}} + \alpha \frac{\partial \mathcal{L}}{\partial E_{out}^k}, \quad (19)$$

where the historical computations E_{out}^{k-1} and $\frac{\partial \mathcal{L}}{\partial E_{in}^{k-1}}$ can be obtained by maintaining the model outputs and gradients at the end of each training iteration.

3.2 Efficient Variance-Reduced Neighbor Sampling

The previous section presents an implicit modeling and single-layer design for GNNs, which significantly reduces GNNs' computation complexity. For example, PinSAGE has a complexity of $\mathcal{O}(|\mathcal{E}|Dd^2)$ given there is only one layer ($L = 1$), which can be linear if D is a small constant. Unfortunately, D affects the expressiveness of GNNs in recommendations and cannot be easily lowered.

As explained in Section 2.2, the mini-batch training process samples user-item interactions (i.e., links in the user-item graph) in each iteration, which means that nodes with higher degrees are more likely to be sampled. These nodes also need more neighbors to compute their output embeddings accurately. Therefore, a small D will introduce large approximation errors and degrade the performance. This is consistent with previous studies on the impact of neighbor sampling in large-scale OGB benchmarks [8]. Some methods, such as VR-GCN [4] and MVS-GNN [6], use variance-reduction (VR) techniques to reduce the random error in neighbor sampling. However, we will show that these methods still require the full aggregation of historical embeddings, which maintains an undesirable complexity. To address this issue, we will propose an efficient VR neighbor sampling approach that achieves linear complexity while controlling the random error.

Classic Variance-reduced Neighbor Aggregation. Recent research has investigated variance reduction on GNNs, such as VR-GCN and MVS-GNN [4, 6]:

$$(\hat{X}^k)^{VR} = \hat{A}(E_{in}^k - \bar{E}_{in}^k) + \tilde{A}\bar{E}_{in}^k \quad (\approx \tilde{A}E_{in}^k), \quad (20)$$

where \hat{A} is an unbiased estimator of \tilde{A} , $\hat{A}_{u,v} = \frac{|N(u)|}{D}\tilde{A}_{u,v}$ if node v is sampled as a neighbor of target node u , otherwise $\hat{A}_{u,v} = 0$. \bar{E}_{in}^k is the historical embeddings for approximating E_{in}^k . However, such approaches need to perform full neighbor aggregations on the historical embedding by computing $\tilde{A}(\bar{E}_{in}^k)$. Importantly, this computation has to be performed in each mini-batch iteration, leading to the quadratic computation complexity $\mathcal{O}(\frac{|\mathcal{E}|^2 d}{B})$ for the whole training epoch. Therefore, they seriously sacrifice the computational efficiency of neighbor sampling in large-scale recommender systems. Besides, it is noteworthy that other GNN acceleration approaches based on historical embeddings [18, 50] also suffer from the full aggregation problem.

Efficient Variance-reduced Neighbor Sampling. To further reduce the quadratic computation complexity, we propose to compute the historical embedding aggregation periodically instead of computing them in every training iteration. Specifically, we allocate

two memory variables M_{in} and M_{ag} to store the historical input embeddings and fully aggregated embeddings, where $M_{ag} = \tilde{A}M_{in}$. The input memory variable M_{in} is updated periodically at the end of each training epoch, and the aggregated embedding M_{ag} are updated based on the renewed inputs. We name it as Efficient Variance Reduction (EVR), which can be formulated as:

$$(\hat{X}^k)^{EVR} = \hat{A}(E_{out}^{k-1} - M_{in}) + M_{ag} \quad (\approx \tilde{A}E_{out}^{k-1}), \quad (21)$$

$$(\hat{E}_{out}^k)^{EVR} = (1 - \alpha)(\hat{X}^k)^{EVR} + \alpha E_{in}^k \quad (\approx E_{out}^k), \quad (22)$$

where the first term in Eq. (18) is approximated by $(\hat{X}^k)^{EVR}$, and the second term remains unchanged. An illustration of this sampling algorithm is shown in Fig. 1(c).

This variance reduction method can also be adapted to backward computations. Symmetrically, the backward computation in Eq. (19) can be computed with our proposed EVR as:

$$(\hat{G}^k)^{EVR} = \hat{A}(\frac{\partial \mathcal{L}}{\partial E_{in}^{k-1}} - M'_{in}) + M'_{ag} \quad (\approx \tilde{A} \frac{\partial \mathcal{L}}{\partial E_{in}^{k-1}}), \quad (23)$$

$$(\frac{\partial \mathcal{L}}{\partial E_{in}^k})^{EVR} = (1 - \alpha)(\hat{G}^k)^{EVR} + \alpha \frac{\partial \mathcal{L}}{\partial E_{out}^k} \quad (\approx \frac{\partial \mathcal{L}}{\partial E_{in}^k}), \quad (24)$$

where M'_{in} stores the historical input gradients and $M'_{ag} = \tilde{A}M'_{in}$ maintains the fully aggregated gradients. Extra implementation details of LTGNN can be found in Appendix. A.1.

Complexity analysis. In each training epoch, the efficiency bottleneck lies in the forward and backward computations, which costs $\mathcal{O}(n_B D d)$, as we compute the variance-reduced neighbor aggregation for n_B target nodes, where each target node has D random neighbors. Thus, the overall complexity of our method is $\mathcal{O}(\frac{|\mathcal{E}|}{B} \cdot n_B D d) = \mathcal{O}(|\mathcal{E}| D d)$, as each training epoch includes $\frac{|\mathcal{E}|}{B}$ iterations. Given that D is a small constant, the complexity no longer preserves an undesirable dependence on $|\mathcal{E}|^2$ or L , and instead, it becomes linear. As a result, it significantly reduces the computational cost in comparison to previous GNN-based recommendation models, as shown in Table. 1.

Table 1: Complexity Comparisons.

Models	Computation Complexity
LightGCN	$\mathcal{O}(\frac{1}{B}L \mathcal{E} ^2 d)$
PinSAGE	$\mathcal{O}(\mathcal{E} D^L d^2)$
MF	$\mathcal{O}(\mathcal{E} d)$
NCF	$\mathcal{O}(L \mathcal{E} d^2)$
LTGNN	$\mathcal{O}(\mathcal{E} Dd)$

4 EXPERIMENTS

In this section, we will verify the effectiveness and efficiency of the proposed LTGNN framework with comprehensive experiments. Specifically, we aim to answer the following research questions:

- **RQ1:** Can LTGNN achieve promising prediction performance on large-scale recommendation datasets? (Section 4.2)
- **RQ2:** Can LGTNN handle large user-item interaction graphs more efficiently than existing GNN approaches? (Section 4.3)
- **RQ3:** How does the effectiveness of the proposed LTGNN vary when we ablate different parts of the design? (Section 4.4)

Table 2: The comparison of overall prediction performance.

Dataset	Yelp2018		Alibaba-iFashion		Amazon-Large	
Method	Recall@20	NDCG@20	Recall@20	NDCG@20	Recall@20	NDCG@20
MF	0.0436	0.0353	0.05784	0.02676	0.02752	0.01534
NCF	0.0450	0.0364	0.06027	0.02810	0.02785	0.01807
GC-MC	0.0462	0.0379	0.07738	0.03395	OOM	OOM
PinSAGE	0.04951	0.04049	0.07053	0.03186	0.02809	0.01973
NGCF	0.0581	0.0475	0.07979	0.03570	OOM	OOM
DGCF	0.064	0.0522	0.08445	0.03967	OOM	OOM
LightGCN (L=3)	0.06347	0.05238	0.08793	0.04096	0.0331	0.02283
LightGCN-NS (L=3)	0.06256	0.05140	<u>0.08804</u>	<u>0.04147</u>	0.02835	0.02035
LightGCN-VR (L=3)	0.06245	0.05141	0.08814	0.04082	0.02903	0.02093
LightGCN-GAS (L=3)	0.06337	0.05207	0.08169	0.03869	0.02886	0.02085
LTGNN (L=1)	0.06393	0.05245	0.09335	0.04387	<u>0.02942</u>	0.02585

4.1 Experimental Settings

We first introduce the datasets, baselines, evaluation metrics, and hyperparameter settings as follows.

Table 3: Dataset statistics.

Dataset	# Users	# Items	# Interactions
Yelp2018	31, 668	38, 048	1, 561, 406
Alibaba-iFashion	300, 000	81, 614	1, 607, 813
Amazon-Large	872, 557	453, 553	15, 236, 325

Datasets. We evaluate the proposed LTGNN and baselines on two medium-scale datasets, including *Yelp2018* and *Alibaba-iFashion*, and one large-scale dataset *Amazon-Large*. *Yelp2018* dataset is released by the baselines NGCF [42] and LightGCN [23], and the *Alibaba-iFashion* dataset can be found in the GitHub repository². For the large-scale setting, we construct the large-scale dataset, *Amazon-Large*, based on the rating files from the Amazon Review Data website³. Specifically, we select the three largest subsets (i.e., Books, Clothing Shoes and Jewelry, Home and Kitchen) from the entire Amazon dataset, and then keep the interactions from users who are shared by all the three subsets (7.9% of all the users). The rating scale is from 1 to 5, and we transform the explicit ratings into implicit interactions by only keeping the interactions with ratings bigger than 4. To ensure the quality of our *Amazon-Large* dataset, we follow a widely used 10-core setting [25, 41, 42] and remove the users and items with interactions less than 10. The statistical summary of the datasets can be found in Table 3.

Baselines. The main focus of this paper is to enhance the scalability of GNN-based collaborative filtering methods. Therefore, we compare our method with the most widely used GNN backbone in recommendations, *LightGCN* [23] and its scalable variants that employ typical GNN scalability techniques, including GraphSAGE [22], VR-GCN [4] and GAS [18]. The corresponding variants of *LightGCN* are denoted as *LightGCN-NS* (for Nighbor Sampling), *LightGCN-VR*, and *LightGCN-GAS*.

To demonstrate the effectiveness of our method, we also compare it with various representative recommendation models, including MF [31], NCF [25], GC-MC [1], PinSAGE [49], NGCF [42], and DGCF [23]. Moreover, since we are designing an efficient collaborative filtering backbone that is independent of the loss function, our method is orthogonal to SSL-based methods [45, 51] and negative sampling algorithms [27, 36]. We will explore the combination of

our method and these orthogonal designs in future work. Extra comparison results with one of the latest accuracy-driven GNN backbones can be found in Appendix. A.3.

Evaluation and Parameter Settings. Due to the limited space, more implementation details are presented in Appendix. A.2.

4.2 Recommendation Performance

In this section, we mainly examine the recommendation performance of our proposed LTGNN, with a particular focus on comparing LTGNN with the most widely adopted GNN backbone *LightGCN*. We use out-of-memory (OOM) to indicate the methods that cannot run on the dataset due to memory limitations. The recommendation performance summarized in Table 2 provides the following observations:

- Our proposed LTGNN achieves comparable or better results on all three datasets compared to the strongest baselines. In particular, LTGNN outperforms all the baselines on Yelp and Alibaba-iFashion. The only exception is that the Recall@20 of *LightGCN* (L=3) outperforms LTGNN (L=1) on the *Amazon-Large* dataset. However, our NDCG@20 outperforms *LightGCN* (L=3), and LTGNN (L=1) is much more efficient compared with *LightGCN* (L=3), as LTGNN only uses one embedding propagation layer and very few randomly sampled neighbors.
- The scalable variants of *LightGCN* improve the scalability of *LightGCN* by sacrificing its recommendation performance in most cases. For instance, the results for *LightGCN-VR*, *LightGCN-NS*, and *LightGCN-GAS* are much worse than *LightGCN* with full embedding propagation on *Amazon-Large*. In contrast, the proposed LTGNN has better efficiency than these variants and preserves the recommendation performance.
- The performance of GNN-based methods like NGCF and *LightGCN* consistently outperforms earlier methods like MF and NCF. However, GNNs without scalability techniques can hardly be run large-scale datasets. For instance, GC-MC, NGCF, and DGCF significantly outperform MF, but they are reported as OOM on the *Amazon-Large* dataset. This suggests the necessity of pursuing scalable GNNs for improving the recommendation performance in large-scale industry scenarios.

4.3 Efficiency Analysis

To verify the scalability of LTGNN, we provide efficiency analysis in comparison with MF, *LightGCN*, and scalable variants of *LightGCN*

²<https://github.com/wenyuer/POG>

³https://cseweb.ucsd.edu/~jmcauley/datasets/amazon_v2/

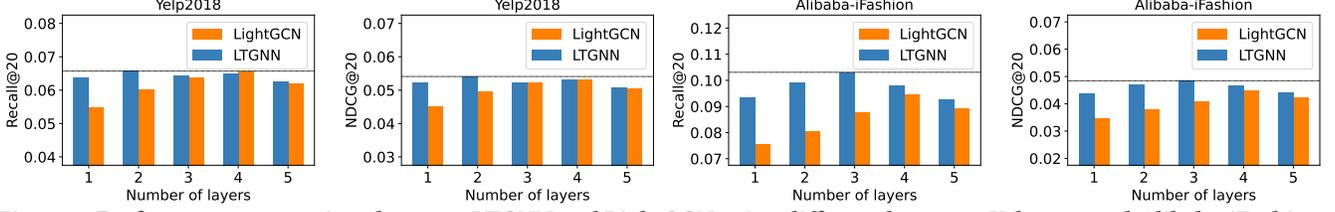


Figure 2: Performance comparison between LTGNN and LightGCN using different layers on Yelp2018 and Alibaba-iFashion.

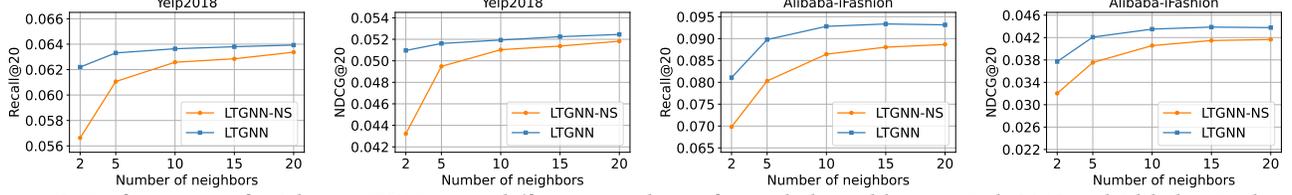


Figure 3: Performance of a 1-layer LTGNN w.r.t different numbers of sampled neighbors on Yelp2018 and Alibaba-iFashion.

Table 4: The comparison of running time on three datasets.

Dataset		Yelp2018	Alibaba-iFashion	Amazon-Large
Method	# Layer	Running Time	Running Time	Running Time
LightGCN		52.83s	51.4s	2999.35s
LightGCN-NS	$L = 3$	46.09s	51.70s	4291.37s
LightGCN-VR		53.15s	59.79s	4849.59s
LightGCN-GAS		23.22s	26.576s	932.03s
LightGCN		30.92s	30.78s	2061.75s
LightGCN-NS	$L = 2$	37.17s	26.89s	1305.25s
LightGCN-VR		38.77s	30.33s	1545.34s
LightGCN-GAS		22.92s	25.04s	903.78s
LightGCN		16.95s	18.02s	1117.51s
LightGCN-NS	$L = 1$	13.90s	12.74s	684.84s
LightGCN-VR		15.52s	13.92s	870.82s
LightGCN-GAS		14.53s	13.35s	729.22s
MF	-	4.31s	4.60s	127.24s
LTGNN	$L = 1$	14.72s	13.68s	705.91s

with different layers on all three datasets: Yelp2018, Alibaba-iFashion, and Amazon-Large. From the running time shown in Table 4, we draw the following conclusions:

- Our proposed single-layer LTGNN achieves comparable running time with one-layer LightGCN with sampling, and outperforms the original LightGCN. This is consistent with our complexity analysis in Section 3.2. Moreover, LTGNN is faster than one-layer LightGCN with variance reduction, owing to our improved and efficient variance reduction (EVR) techniques. Although LTGNN is not substantially more efficient than some of the one-layer GNNs, it has much better recommendation performance, as shown in Figure 2.
- LTGNN demonstrates significantly improved computational efficiency compared to baseline models with more than one layer.

When combined with the results from Table 2, it becomes evident that LTGNN can maintain high performance while achieving a substantial enhancement in computational efficiency.

- While the running time of LTGNN is a few times longer than that of Matrix Factorization (MF) due to their constant factor difference in the complexity analysis (Table. 1), it’s important to note that LTGNN already exhibits a nice and *similar scaling behavior as MF*. This supports the better scalability of LTGNN in large-scale recommendations compared with other GNN-based methods.
- An interesting observation is that on large-scale datasets, full-graph LightGCN surpasses LightGCN with neighbor sampling

on efficiency. This is mainly because of the high CPU overhead of random sampling, which limits the utilization of GPUs.

To further understand the efficiency of each component of LTGNN, a fine-grained efficiency analysis can be found in Appendix. A.3.

4.4 Ablation Study

In this section, we provide extensive ablation studies to evaluate the effectiveness of different parts in our proposed framework. Extra experiments on the effect of hyperparameter α and different variance-reduction designs can be found in Appendix. A.3.

Effectiveness of implicit graph modeling. We conduct an ablation study to show the effect of embedding propagation layers and long-range collaborative signals. Particularly, we use the same setting for LightGCN and LTGNN and change the number of propagation layers L . As illustrated in Figure 2, we have two key observations: 1) LTGNN with only 1 or 2 propagation layers can reach better performance in comparison with LightGCN with more than 3 layers, which demonstrates the strong long-range modeling capability of our proposed model; 2) Adding more propagation layers into LTGNN will not significantly improve its performance, which means $L = 1$ or $L = 2$ are the best choices for LTGNN to balance its performance and scalability.

Effectiveness of efficient variance reduction. In this study, we aim to demonstrate the effectiveness of our proposed EVR algorithm by showing the impact of the number of neighbors on recommendation performance. As shown in Figure 3, LTGNN with efficient variance reduction consistently outperforms its vanilla neighbor sampling variant (i.e., LTGNN-NS) regardless of the number of neighbors, illustrating its effect in reducing the large approximation error caused by neighbor sampling. The recommendation performance of LTGNN with efficient variance reduction is remarkably stable, even under extreme conditions like sampling only 2 neighbors for each target node. This indicates the great potential of our proposed LTGNN in large-scale recommendations, as a GNN with only one propagation layer and two random neighbors will be ultra-efficient compared with previous designs that incur a large number of layers and neighbors.

Numerical Analysis. In this experiment, we compute the PPNP embedding propagation result E_{PPNP}^k for the target nodes as an indicator of long-range modeling ability, and we compute the

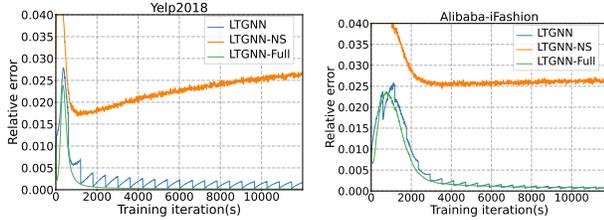


Figure 4: The relative error between the model output E_{out}^k and the exact PPNP propagation result E_{PPNP}^k of the embeddings (i.e., $\|E_{out}^k - E_{PPNP}^k\|_F / \|E_{PPNP}^k\|_F$).

relative error between the model output E_L^k and the PPNP computation result. We use $L = 1$ for LTGNN and its two inferior variants without efficient variance reduction - LTGNN-NS and LTGNN-Full, which denotes LTGNN with random neighbor sampling and LTGNN with exact full neighbor aggregation. From Figure 4, we have two observations as follows: 1) On both datasets, the output of LTGNN converges to PPNP after around 4000 training iterations (i.e., less than 10 training epochs), which means our proposed LTGNN can capture the long-range dependencies in user-item graphs by using only one propagation layer; 2) By comparing LTGNN with its variants, it is obvious that neighbor sampling without variance reduction seriously hurts the modeling ability of LTGNN, and our proposed LTGNN has similar convergence curves in comparison to LTGNN with full aggregation, showing the effectiveness of our proposed efficient variance reduction method.

5 RELATED WORK

In this section, we summarize the related works on graph-based collaborative filtering and scalable GNNs.

5.1 Graph Collaborative Filtering Models for Recommendations

In modern recommender systems, collaborative filtering (CF) is one of the most representative paradigm [12, 14, 16] to understand users’ preferences. The basic idea of CF is to decompose user-item interaction data into trainable user and item embeddings, and then reconstruct the missing interactions [15, 17, 25, 31]. Early works in CF mainly model the user-item interactions with scaled dot-product [5, 12, 31], MLPs [11, 25], and LSTMs [15, 21]. However, these models fail to model the high-order collaborative signals between users and items, leading to sub-optimal representations of users and items.

In recent years, a promising line of studies has incorporated GNNs into CF-based recommender systems. The key advantage of utilizing GNN-based recommendation models is that GNNs can easily capture long-range dependencies via the information propagation mechanism on the user-item graph. For example, an early exploration, GC-MC, completes the missing user-item interactions with graph convolutional autoencoders [1]. For large-scale recommendation scenarios, PinSAGE [49] adapts the key idea of GraphSAGE [22] to recommendations and achieves promising results. Another significant milestone in GNN-based recommendations is the NGCF [42], which explicitly formulates the concept of collaborative signals and models them as high-order connectivities by message-passing propagations. Afterward, LightGCN [23] indicates the non-linear activation functions and feature transformations in

GNN-based recommender systems are redundant, and proposes to simplify existing GNNs while achieving promising performance. However, despite the success of previous GNN-based models in capturing user preferences, existing works fail to address the neighborhood explosion problem on large-scale recommendation scenarios, which indicates that the scalability of GNN-based recommender systems remains an open question.

5.2 Scalability of Graph Neural Networks

Recently, extensive literature has explored the scalability of GNNs on large-scale graphs [2, 40], with a wide range of research focusing on sampling methods, pre-computing methods, post-computing methods, and memory-based methods. Sampling-based methods lower the computation and memory requirements of GNNs by using a mini-batch training strategy on GNNs, which samples a limited number of neighbors for target nodes in a node-wise [4, 6, 22], layer-wise [3, 55], or subgraph-wise [52] manner, mitigating the well-known neighborhood explosion problem. However, sampling-based methods inevitably omit a large number of neighbors for aggregation, resulting in large random errors. As a remedy for this limitation, memory-based methods [18, 48, 50] leverage the historical feature memories to complement the missing information of out-of-batch nodes, approximating the full aggregation outputs while enjoying the efficiency of only updating the in-batch nodes. Unfortunately, these methods may have complexities similar to full neighbor aggregations, which block the way to linear complexity. Besides, pre-computing or post-computing methods decouple feature transformation and feature aggregations, enabling capturing long-range dependencies in graphs while only training a feature transformation model. In particular, pre-computing methods pre-calculate the feature aggregation result before training [44, 53], while post-computing methods firstly train a feature transformation model and leverage unsupervised algorithms, such as label propagation, to predict the labels [26, 54]. In retrospect, pre-computing/post-computing methods may sacrifice the advantage of end-to-end training, and the approximation error of these methods is still unsatisfactory. These limitations in existing scalable GNNs strongly necessitate our pursuit of scalable GNNs for large-scale recommender systems in this paper.

6 CONCLUSION

Scalability is a major challenge for GNN-based recommender systems, as they often require many computational resources to handle large-scale user-item interaction data. To address this challenge, we propose a novel scalable GNN model for recommendation, which leverages implicit graph modeling and variance-reduced neighbor sampling to capture long-range collaborative signals while achieving a desirable linear complexity. The proposed LTGNN only needs one propagation layer and a small number of one-hop neighbors, which reduces the computation complexity to be linear to the number of edges, showing great potential in industrial recommendation applications. Extensive experiments on three real-world datasets are conducted to demonstrate the effectiveness and efficiency of our proposed model. For future work, we plan to extend our design to more recommendation tasks, such as CTR prediction, and explore the deployment of LTGNN in industrial recommender systems.

ACKNOWLEDGMENTS

The research described in this paper has been partly supported by NSFC (project no. 62102335), General Research Funds from the Hong Kong Research Grants Council (project no. PolyU 15200021, 15207322, and 15200023), internal research funds from The Hong Kong Polytechnic University (project no. P0036200, P0042693, P0048625, P0048752), Research Collaborative Project no. P0041282, and SHTM Interdisciplinary Large Grant (project no. P0043302). Xiaorui Liu is partially supported by the Amazon Research Award.

REFERENCES

- [1] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2017. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263* (2017).
- [2] Hao Chen, Yuanchen Bei, Qijie Shen, Yue Xu, Sheng Zhou, Wenbing Huang, Feiran Huang, Senzhang Wang, and Xiao Huang. 2024. Macro Graph Neural Networks for Online Billion-Scale Recommender Systems. In *WWW*.
- [3] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. In *ICLR*.
- [4] Jianfei Chen, Jun Zhu, and Le Song. 2018. Stochastic Training of Graph Convolutional Networks with Variance Reduction. In *ICML*.
- [5] Xiao Chen, Wenqi Fan, Jingfan Chen, Haochen Liu, Zitao Liu, Zhaoxiang Zhang, and Qing Li. 2023. Fairly adaptive negative sampling for recommendations. In *WWW*.
- [6] Weilin Cong, Rana Forsati, Mahmut Kandemir, and Mehrdad Mahdavi. 2020. Minimal variance sampling with provable guarantees for fast training of graph neural networks. In *KDD*.
- [7] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *RecSys*.
- [8] Keyu Duan, Zirui Liu, Peihao Wang, Wenqing Zheng, Kaixiong Zhou, Tianlong Chen, Xia Hu, and Zhangyang Wang. 2022. A comprehensive study on large-scale graph training: Benchmarking and rethinking. *NeurIPS* (2022).
- [9] Laurent El Ghaoui, Fangda Gu, Bertrand Travacca, Armin Askari, and Alicia Tsai. 2021. Implicit deep learning. *SIAM Journal on Mathematics of Data Science* 3, 3 (2021), 930–958.
- [10] Ali Mamdouh Elkahky, Yang Song, and Xiaodong He. 2015. A multi-view deep learning approach for cross domain user modeling in recommendation systems. In *WWW*.
- [11] Wenqi Fan, Qing Li, and Min Cheng. 2018. Deep modeling of social relations for recommendation. In *AAAI*.
- [12] Wenqi Fan, Xiaorui Liu, Wei Jin, Xiangyu Zhao, Jiliang Tang, and Qing Li. 2022. Graph trend filtering networks for recommendation. In *SIGIR*.
- [13] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *WWW*.
- [14] Wenqi Fan, Yao Ma, Qing Li, Jianping Wang, Guoyong Cai, Jiliang Tang, and Dawei Yin. 2020. A graph neural network framework for social recommendations. *TKDE* 34, 5 (2020), 2033–2047.
- [15] Wenqi Fan, Yao Ma, Dawei Yin, Jianping Wang, Jiliang Tang, and Qing Li. 2019. Deep social collaborative filtering. In *RecSys*.
- [16] Wenqi Fan, Shijie Wang, Xiao-yong Wei, Xiaowei Mei, and Qing Li. 2023. Untargeted Black-box Attacks for Social Recommendations. *arXiv preprint arXiv:2311.07127* (2023).
- [17] Wenqi Fan, Xiangyu Zhao, Qing Li, Tyler Derr, Yao Ma, Hui Liu, Jianping Wang, and Jiliang Tang. 2023. Adversarial Attacks for Black-Box Recommender Systems Via Copying Transferable Cross-Domain User Profiles. *TKDE* (2023).
- [18] Matthias Fey, Jan E Lenssen, Frank Weichert, and Jure Leskovec. 2021. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. In *ICML*.
- [19] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *ICLR*.
- [20] Fangda Gu, Heng Chang, Wenwu Zhu, Somayeh Sojoudi, and Laurent El Ghaoui. 2020. Implicit graph neural networks. In *NeurIPS*.
- [21] Qing Guo, Zhu Sun, Jie Zhang, and Yin-Leng Theng. 2020. An attentional recurrent neural network for personalized next location recommendation. In *AAAI*.
- [22] Will Hamilton, Zitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*.
- [23] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *SIGIR*.
- [24] Xiangnan He, Zhankui He, Jingkuan Song, Zhenguang Liu, Yu-Gang Jiang, and Tat-Seng Chua. 2018. NAIS: Neural attentive item similarity model for recommendation. *TKDE* 30, 12 (2018), 2354–2366.
- [25] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW*.
- [26] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin Benson. 2020. Combining Label Propagation and Simple Models out-performs Graph Neural Networks. In *ICLR*.
- [27] Tinglin Huang, Yuxiao Dong, Ming Ding, Zhen Yang, Wenzheng Feng, Xinyu Wang, and Jie Tang. 2021. Mixgcf: An improved training method for graph neural network-based recommender systems. In *KDD*.
- [28] Wei Jin, Haitao Mao, Zheng Li, Haoming Jiang, Chen Luo, Hongzhi Wen, Haoyu Han, Hanqing Lu, Zhengyang Wang, Ruirui Li, et al. 2023. Amazon-M2: A Multilingual Multi-locale Shopping Session Dataset for Recommendation and Text Generation. In *NeurIPS*.
- [29] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.
- [30] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.
- [31] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*.
- [32] Page Lawrence. 1998. The pagerank citation ranking: Bringing order to the web. *Technical report* (1998).
- [33] Mingjie Li, Yifei Wang, Yisen Wang, and Zhouchen Lin. 2022. Unbiased Stochastic Proximal Solver for Graph Neural Networks with Equilibrium States. In *ICLR*.
- [34] Zhao Li, Xin Shen, Yuhang Jiao, Xuming Pan, Pengcheng Zou, Xianling Meng, Chengwei Yao, and Jiajun Bu. 2020. Hierarchical bipartite graph neural networks: Towards large-scale e-commerce applications. In *ICDE*.
- [35] Chengyi Liu, Wenqi Fan, Yunqing Liu, Jiatong Li, Hang Li, Hui Liu, Jiliang Tang, and Qing Li. 2023. Generative diffusion models on graphs: Methods and applications. In *IJCAI*.
- [36] Kelong Mao, Jieming Zhu, Jinpeng Wang, Quanyu Dai, Zhenhua Dong, Xi Xiao, and Xiuqiang He. 2021. SimpleX: A simple and strong baseline for collaborative filtering. In *CIKM*.
- [37] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UIAI*.
- [38] Shang-Hua Teng et al. 2016. Scalable algorithms for data and network analysis. *Foundations and Trends® in Theoretical Computer Science* 12, 1–2 (2016), 1–274.
- [39] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *KDD*.
- [40] Lin Wang, Wenqi Fan, Jiatong Li, Yao Ma, and Qing Li. 2024. Fast graph condensation with structure-based neural tangent kernel. In *WWW*.
- [41] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. 2019. Kgat: Knowledge graph attention network for recommendation. In *KDD*.
- [42] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *SIGIR*.
- [43] Xiang Wang, Hongye Jin, An Zhang, Xiangnan He, Tong Xu, and Tat-Seng Chua. 2020. Disentangled graph collaborative filtering. In *SIGIR*.
- [44] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *ICML*.
- [45] Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. 2021. Self-supervised graph learning for recommendation. In *SIGIR*.
- [46] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *TNNLS* 32, 1 (2020), 4–24.
- [47] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. 2017. Deep matrix factorization models for recommender systems. In *IJCAI*.
- [48] Rui Xue, Haoyu Han, MohamadAli Torkamani, Jian Pei, and Xiaorui Liu. 2023. LazyGNN: Large-Scale Graph Neural Networks via Lazy Propagation. In *ICML*.
- [49] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD*.
- [50] Haiyang Yu, Limei Wang, Bokun Wang, Meng Liu, Tianbao Yang, and Shuiwang Ji. 2022. GraphFM: Improving large-scale GNN training via feature momentum. In *ICML*.
- [51] Junliang Yu, Hongzhi Yin, Xin Xia, Tong Chen, Lizhen Cui, and Quoc Viet Hung Nguyen. 2022. Are graph augmentations necessary? simple graph contrastive learning for recommendation. In *SIGIR*.
- [52] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *ICLR*.
- [53] Wentao Zhang, Ziqi Yin, Zeang Sheng, Yang Li, Wen Ouyang, Xiaosen Li, Yangyu Tao, Zhi Yang, and Bin Cui. 2022. Graph attention multi-layer perceptron. In *KDD*.
- [54] Xiaojin Zhu. 2005. *Semi-supervised learning with graphs*. Carnegie Mellon University.
- [55] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-dependent importance sampling for training deep and large graph convolutional networks. In *NeurIPS*.

A APPENDIX

Algorithm 1: The training process of LTGNN

Input: User-item interactions \mathcal{R} ; BPR batch size B ; Epochs E

Output: Optimized user-item embeddings E_{in}^{final}

- 1 Initialize training iteration count $k \leftarrow 0$;
- 2 Initialize the user-item embeddings $E_{in}^0 \sim \mathcal{N}(\mu, \sigma^2)$;
- 3 **for** $epoch = 1 \dots E$ **do**
- 4 Compute variance reduction memory $[M_{in}; M'_{in}] \leftarrow [E_{out}^{k-1}, \frac{\partial \mathcal{L}}{\partial E_{in}^{k-1}}], [M_{ag}; M'_{ag}] \leftarrow \tilde{A}[M_{in}; M'_{in}]; \triangleright \mathcal{O}(|\mathcal{E}|d)$
- 5 **for** $sample\ B\ interactions\ \hat{\mathcal{R}}^+$ from \mathcal{R}^+ **do** $\triangleright \frac{|\mathcal{E}|}{B}$ batches
- 6 Obtain training data $\hat{\mathcal{O}} = \{(u, i, j) | (u, i) \in \hat{\mathcal{R}}^+, (u, j) \in \mathcal{R}^-\}$;
- 7 Obtain the set of target nodes $\mathbf{B} = \bigcup_{(u,i,j) \in \hat{\mathcal{O}}} \{u, i, j\}$;
- 8 Sample the random neighbors for each target node in \mathbf{B} to obtain $\hat{\mathbf{A}}$;
- 9 Obtain the forward output E_{out}^k according to Eq. (21) and Eq. (22); $\triangleright \mathcal{O}(n_B D d)$
- 10 Compute the loss function \mathcal{L}_{BPR} in Eq. (2) and the gradients at the output layer $\frac{\partial \mathcal{L}}{\partial E_{out}^k}$; $\triangleright \mathcal{O}(Bd)$
- 11 Compute the implicit gradients $\frac{\partial \mathcal{L}}{\partial E_{in}^k}$ according to Eq. (23) and Eq. (24); $\triangleright \mathcal{O}(n_B D d)$
- 12 Update the embedding table E_{in} with an arbitrary optimizer $E_{in}^{k+1} \leftarrow \text{UPDATE}(E_{in}^k, \frac{\partial \mathcal{L}}{\partial E_{in}^k})$; $\triangleright \mathcal{O}(n_B d)$
- 13 Save E_{out}^k and $\frac{\partial \mathcal{L}}{\partial E_{in}^k}$ to memory for the next training iteration;
- 14 $k \leftarrow k + 1$; \triangleright The for-loop costs $\mathcal{O}(|\mathcal{E}|Dd)$

15 **return** the optimized embeddings $E_{in}^{final} = E_{in}^k$.

A.1 Model Details

In this subsection, we will introduce the model details of our proposed Linear Time Graph Neural Network (LTGNN), which are not fully covered in previous Section 3.

Model Training. The detailed training process of LTGNN is presented in Algorithm 1. First, we initialize the training iteration count k and the trainable user-item embeddings E_{in} (line 1-2). Next, we repeat the outer training loop E times for E training epochs (lines 3-16). In each training epoch, we first compute the variance reduction memory for variance reduced aggregation (line 4), and then start the mini-batch training iterations (lines 5-15).

In each training iteration, the mini-batch interactions $\hat{\mathcal{R}}^+$ are sampled from the observed user-item interactions \mathcal{R}^+ . For every mini-batch, we first conduct negative sampling to obtain the training data $\hat{\mathcal{O}}$ (line 6), and then sample the neighbors for the target nodes in GNN aggregations (lines 7-8). After the negative and neighbor sampling process, we compute the forward and backward propagations of LTGNN, which are detailed in previous lines (lines 9-12). At the end of each training iteration, the forward and backward outputs are maintained for the incoming training iteration (line 13), enabling us to solve the PPNP fixed point with these historical computations.

Random Aggregation with $\hat{\mathbf{A}}$. For each target node in \mathbf{B} , which consists of all the users, positive items, and negative items, we sample D random neighbors without repetition (including the node itself). We denote the set of random neighbors for a target node u as $\hat{\mathcal{N}}(u)$, and its original neighbor set as $\mathcal{N}(u)$. We define the elements of the random adjacency matrix $\hat{\mathbf{A}}$ as follows:

$$\hat{A}_{u,v} = \begin{cases} \frac{|\mathcal{N}(u)|}{D} \tilde{A}_{u,v}, & \text{if } u \in \mathbf{B} \text{ and } v \in \hat{\mathcal{N}}(u) \\ 0, & \text{otherwise} \end{cases}. \quad (25)$$

Thus, the matrix form embedding propagation $\hat{\mathbf{X}} = \hat{\mathbf{A}}\mathbf{E}$ is equivalent to the node-wise random aggregation:

$$\hat{x}_u = \sum_{v \in \hat{\mathcal{N}}(u)} \frac{|\mathcal{N}(u)|}{D} \tilde{A}_{u,v} e_v, \quad \forall u \in \mathbf{B}. \quad (26)$$

It is clear that \hat{x}_u is an unbiased estimator of exact aggregation $x_u = \sum_{v \in \mathcal{N}(u)} A_{u,v} e_v$, since

$$\begin{aligned} \mathbb{E}[\hat{x}_u] &= \frac{|\mathcal{N}(u)|}{D} \mathbb{E}[\sum_{v \in \hat{\mathcal{N}}(u)} \tilde{A}_{u,v} e_v \mathbb{I}(v | u)] \\ &= \frac{|\mathcal{N}(u)|}{D} \sum_{v \in \mathcal{N}(u)} \tilde{A}_{u,v} e_v \mathbb{E}[\mathbb{I}(v | u)] \\ &= \frac{|\mathcal{N}(u)|}{D} \sum_{v \in \mathcal{N}(u)} \tilde{A}_{u,v} e_v \frac{C^{D-1} |\mathcal{N}(u)| - 1}{C^D |\mathcal{N}(u)|} \\ &= \sum_{v \in \mathcal{N}(u)} A_{u,v} e_v = x_u. \end{aligned}$$

where $\mathbb{I}(v | u)$ is an indicator function that equals to 1 when v is sampled for target node u and 0 otherwise. This unbiasedness holds for both forward (line 9) and backward (line 11) computations in Algorithm 1, and is in line with previous discussions in VR-GCN [4]. **Detailed Complexity Analysis.** In each training epoch, the efficiency bottleneck lies in the forward and backward computations in line 9 and line 11. According to Eq. (25), there are $n_B D$ edges in the random adjacency matrix $\hat{\mathbf{A}}$, so the variance-reduced neighbor aggregation has complexity $\mathcal{O}(n_B D d)$ with the help of sparse matrix multiplications. Thus, the overall complexity of our method is $\mathcal{O}(\frac{|\mathcal{E}|}{B} \cdot n_B D d) = \mathcal{O}(|\mathcal{E}|Dd)$, since the inner training loop in lines 5-15 repeats $\frac{|\mathcal{E}|}{B}$ times, and B and n_B have the same order. Given that D is a small constant, the complexity becomes linear to the number of edges $|\mathcal{E}|$, demonstrating high scalability potential.

It is also noteworthy that our variance-reduced aggregation strategy does not affect the linear complexity of LTGNN. Particularly, the variance-reduced aggregations in lines 9 and 11 have the same complexity as vanilla neighbor sampling methods in PinSAGE, while the extra computational overhead of computing the variance reduction memory is $\mathcal{O}(|\mathcal{E}|d)$, which is as efficient as the simplest MF model (line 2). This means our proposed variance reduction approach reduces the random error without sacrificing the training efficiency.

Model Inference. After the model training process described in Algorithm 1, we have several options to predict user preference with the optimized input embeddings E_{in}^{final} . The simplest solution is to directly infer the output embeddings with Eqs. (21) - (22), and then

predict the future user-item interaction with an inner product. However, the training process may introduce small errors due to reusing the historical computations, which could cause E_{out}^{final} to deviate from the exact PPNP fixed-point E_{PPNP}^{final} w.r.t. E_{in}^{final} . Therefore, a possible improvement is to compute E_{PPNP}^{final} with APPNP layers in Eqs. (12) - (13). This strategic choice can reduce the error due to reusing the previous computations, despite slightly compromising the behavior consistency between training and inference.

We find that both inference choices can accurately predict the user preference, and choosing either of them does not have a significant impact on the recommendation performance. In practice, we compute E_{PPNP}^{final} with a 3-layer APPNP which takes E_{in}^{final} as the input.

A.2 Parameter and Evaluation Settings

We implement the proposed LTGNN using PyTorch and PyG libraries. We strictly follow the settings of NGCF [42] and LightGCN [23] to implement our method and the scalable LightGCN variants for a fair comparison. All the methods use an embedding size of 64, a BPR batch size of 2048, and a negative sample size of 1. For the proposed LTGNN, we tune the learning rate from $\{5e-4, 1e-3, 1.5e-3, 2e-3\}$ and the weight decay from $\{1e-4, 2e-4, 3e-4\}$. We employ an Adam [29] optimizer to minimize the objective function. For the coefficient α in PPNP, we perform a grid search over the hyperparameter in the range of $[0.3, 0.7]$ with a step size of 0.05. To ensure the scalability of our model, the number of propagation layers L is fixed to 1 by default, and the number of sampled neighbors D is searched in $\{5, 10, 15, 20\}$. For the GNN-based baselines, we follow their official implementations and suggested settings in their papers. For the LightGCN variants with scalability techniques, the number of layers L is set based on the best choice of LightGCN, and we search other hyperparameters in the same range as LTGNN and report the best results.

In this paper, we adopt two widely used evaluation metrics in recommendations: Recall@K and Normalized Discounted Cumulative Gain (NDCG@K) [23, 42]. We set the K=20 by default, and we report the average result for all test users. All the experiments are repeated five times with different random seeds, and we report the average results.

A.3 Additional Experiments

Comparison to GTN. Despite numerous works being orthogonal to our contributions and not tackling the scalability issue, as highlighted in Section 4.1, we remain receptive to comparing our method with more recent, accuracy-focused baselines in the field. We compare our proposed LTGNN with GTN [12], one of the latest GNN backbones for recommendations, and the results are presented in Table. 5 and Table. 6.

Table 5: Recommendation performance comparison results with extra baseline GTN. Results marked with (*) are obtained in GTN’s official settings.

Dataset	Yelp2018		Alibaba-iFashion		Amazon-Large	
	Recall@20	NDCG@20	Recall@20	NDCG@20	Recall@20	NDCG@20
GTN	0.0679*	0.0554*	0.0994*	0.0474*	OOM	OOM
LTGNN	0.0681*	0.0562*	0.1079*	0.0510*	0.0294	0.0259

Table 6: Running time comparison results with extra baseline GTN. Results marked with (*) are obtained in GTN’s official settings.

Dataset	Yelp2018	Alibaba-iFashion	Amazon-Large
Method	Running Time	Running Time	Running Time
GTN	97.8s*	99.6s*	OOM
LTGNN	24.5s*	22.4s*	705.9s

To ensure a fair comparison, we closely followed the official settings of GTN ⁴, which differ from our evaluation settings detailed in Section 4.1 in embedding size and BPR batch size. From the experiment result, we find that our proposed LTGNN shows advantages in both recommendation performance and scalability, even in comparison with one of the latest accuracy-driven GNN backbones for recommendations.

Effect of Hyperparameter α . Similar to APPNP [19], the teleport factor α in LTGNN controls the trade-off between capturing long-range dependencies and graph localities, directly impacting recommendation performance. As shown in Fig. 5, we find that LTGNN is rather robust to α , as the performance does not drop sharply in $\alpha \in [0.35, 0.55]$. In practice, we can set $\alpha = 0.5$, which is the midpoint of capturing long-range dependency and preserving the local structure expressiveness, and conduct a rough grid search to find the optimal setting of α .

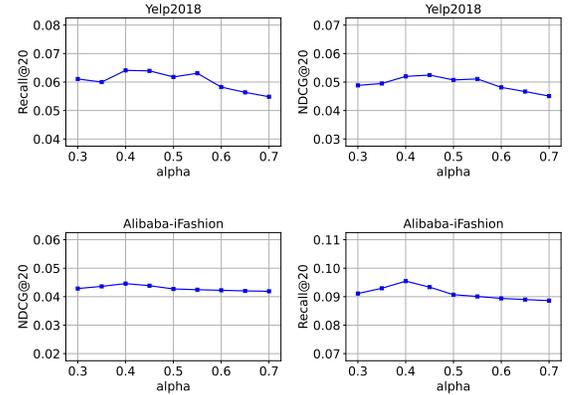


Figure 5: The effect of hyper-parameter α under Recall@20 and NDCG@20 metrics.

Effect of Different Variance Reduction Designs. We developed an efficient variance reduction (EVR) mechanism in Section 3.2, which can be applied to both forward and backward computations. To evaluate the impact of different variance reduction designs on the recommendation performance, we compare four variants of LTGNN, namely *LTGNN-NS*, *LTGNN-FVR*, *LTGNN-BVR*, and *LTGNN-BiVR*. *LTGNN-NS* is the baseline model that uses implicit modeling and vanilla neighbor sampling, while *LTGNN-FVR*, *LTGNN-BVR*, and *LTGNN-BiVR* use forward variance reduction, backward variance reduction, and bidirectional variance reduction, respectively.

As can be seen from Table. 7, LTGNN variants with variance reduction outperform *LTGNN-NS* without variance reduction, demonstrating the effectiveness of our proposed EVR approach. Moreover, we observe that forward variance reduction (FVR) alone is sufficient

⁴ <https://github.com/wenqifan03/GTN-SIGIR2022>

Table 7: Ablation study on different variance reduction designs.

Dataset Method	Yelp2018		Alibaba-iFashion	
	Recall@20	NDCG@20	Recall@20	NDCG@20
LTGNN-NS	0.06285	0.05137	0.08804	0.04147
LTGNN-BVR	0.06309	0.05160	0.08878	0.04185
LTGNN-BiVR	<u>0.06321</u>	<u>0.05194</u>	<u>0.09241</u>	<u>0.04338</u>
LTGNN-FVR	0.6380	0.05224	0.09335	0.04387

to achieve satisfactory recommendation performance. Surprisingly, bidirectional variance reduction does not outperform forward variance reduction. We intend to investigate the reason behind this phenomenon and explore the potential of bidirectional variance reduction (BiVR) in our future work.

Detailed Efficiency Analysis. As discussed in Section 3.2, LTGNN has linear time complexity with respect to the number of edges $|\mathcal{E}|$ in the user-item interaction graph, which is comparable to MF. However, as shown in Table. 4, the running time of LTGNN is worse than MF on the large-scale Amazon-Large dataset, which seems to contradict our complexity analysis. To understand this phenomenon and demonstrate LTGNN’s scalability advantage, we perform a comprehensive efficiency analysis of LTGNN and MF, which identifies the main source of overhead and suggests potential enhancements.

From the results presented in Table. 8, we have the following observations:

- For both MF and LTGNN, the computation cost of negative sampling is negligible, which has a minimal impact on the total running time.
- For LTGNN, excluding the neighbor sampling time, the model training time is linear in the number of edges $|\mathcal{E}|$,

which is similar to MF. This indicates LTGNN’s high scalability in large-scale and industrial recommendation scenarios.

- Besides, the computational overhead incurred by maintaining and updating the memory spaces for variance reduction is insignificant, which validates the high efficiency of our proposed EVR approach.
- For LTGNN, the main source of extra computational overhead is the neighbor sampling process for the target nodes, which accounts for more than 50% of the total running time, preventing LTGNN from achieving a perfect linear scaling behavior. This is an implementation issue that can be improved by better engineering solutions.

To further enhance the efficiency of LTGNN, we can adopt some common engineering techniques. For example, we can follow the importance sampling approach in PinSAGE [49], which assigns a fixed neighborhood for each node in the interaction graph, avoiding the expensive random sampling process. We plan to leave this as a future work.

Table 8: Detailed efficiency comparison of LTGNN with MF.

Model	Stage	Yelp2018	Alibaba-iFashion	Amazon-Large
		$ \mathcal{E} = 1.56m$	$ \mathcal{E} = 1.61m$	$ \mathcal{E} = 15.24m$
MF	Neg. Samp.	0.12s	0.21s	1.23s
	Training	<u>4.19s</u>	<u>4.39s</u>	<u>126.01s</u>
	Total	4.31s	4.60s	127.24s
LTGNN	Neg. Samp.	0.12s	0.21s	1.23s
	Neigh. Samp.	7.98s	6.48s	512.55s
	Training	<u>6.62s</u>	<u>6.99s</u>	<u>192.13s</u>
	Memory Access	<0.005s	<0.005s	<0.005s
	Total	14.72s	13.68s	705.91s