Learning Binarized Graph Representations with Multi-faceted Quantization Reinforcement for Top-K Recommendation

Yankai Chen¹, Huifeng Guo², Yingxue Zhang², Chen Ma³, Ruiming Tang², Jingjie Li², Irwin King¹
¹The Chinese University of Hong Kong, ²Huawei Noah’s Ark Lab, ³City University of Hong Kong
{ykkchen,king}@cse.cuhk.edu.hk; chenna@cityu.edu.hk; {huifeng.guo,yingxue.zhang,tangruiming,lijingjie1}@huawei.com

Abstract
Learning vectorized embeddings is at the core of various recommender systems for user-item matching. To perform efficient online inference, representation quantization, aiming to embed the latent features by a compact sequence of discrete numbers, recently shows the promising potentiality in optimizing both memory and computation overheads. However, existing work merely focuses on numerical quantization whilst ignoring the concomitant information loss issue, which, consequently, leads to conspicuous performance degradation. In this paper, we propose a novel quantization framework to learn Binarized Graph Representations for Top-K Recommendation (BiGeaR). BiGeaR introduces multi-faceted quantization reinforcement at the pre-, mid-, and post-stage of binarized representation learning, which substantially retains the representation informativeness against embedding binarization. In addition to saving the memory footprint, BiGeaR further develops solid online inference acceleration with bitwise operations, providing alternative flexibility for the realistic deployment. The empirical results over five large real-world benchmarks show that BiGeaR achieves about 22%–40% performance improvement over the state-of-the-art quantization-based recommender system, and recovers about 95%–102% of the performance capability of the best full-precision counterpart with over 8× time and space reduction.

CCS Concepts
• Information systems → Recommender systems

Keywords
Recommender system; Quantization-based; Binarization; Graph Convolutional Network; Graph Representation

1 Introduction
Recommender systems, aiming to perform personalized information filtering [65], are versatile to many Internet applications. Learning vectorized user-item representations (i.e., embeddings), as the core of various recommender models, is the prerequisite for online inference of user-item interactions [21, 56]. With the explosive data expansion (e.g., Amazon owns over 150M users and 350M products [42, 52]), one major existing challenge however is to perform inference efficiently. This usually requires large memory and computation consumption (e.g., for Amazon 500M-scaled full-precision embedding table) on certain data centers [49], and therefore hinders the deployment to devices with limited resources [49].

To tackle this issue, representation quantization recently provides the promising feasibility. Generally, it learns to quantize latent features of users and items via converting the continuous full-precision representations into discrete low-bit ones. The quantized representations thus are conducive to model size reduction and inference speed-up with low-bit arithmetic on devices where CPUs are typically more affordable than expensive GPUs [2, 3]. Technically, quantization can be categorized into multi-bit, 2-bit (i.e., ternarized), and 1-bit (i.e., binarized) quantization. With only one bit-width, representation binarization for recommendation takes the most advantage of representation quantization and therefore draws the growing attention recently [28, 50].

Despite the promising potentiality, it is still challenging to develop realistic deployment mainly because of the large performance degradation in Top-k recommendation [28, 50]. The crux of the matter is the threefold information loss:

• Limited expressivity of latent features. Because of the discrete constraints, mapping full-precision embeddings into compact binary codes with equal expressivity is NP-hard [19]. Thus, instead of proposing complex and deep neural structures for quantization [13, 70], sign(·) function is widely adopted to achieve O(1) embedding binarization [37, 50, 55]. However, this only guarantees the sign (+/−) correlation for each embedding entry.

1It could be single-precision or double-precision; we use float32 as the default for explanation and conducting experiments throughout this paper.
Compared to the original full-precision embeddings, binarized targets produced from $\text{sign}(\cdot)$ are naturally less informative.

- **Degraded ranking capability.** Ranking capability, as the essential measurement of Top-K recommendation, is the main objective to work on. Apart from the inevitable feature loss in numerical quantization, previous work further ignores the discrepancy of hidden knowledge that is inferred by full-precision and binarized embeddings [28, 50]. However, such hidden knowledge is vital to reveal users’ preference towards different items, losing of which may thus draw degraded ranking capability and suboptimal model learning accordingly.

- **Inaccurate gradient estimation.** Due to the non-differentiability of quantization function $\text{sign}(\cdot)$, Straight-Through Estimator (STE) [4] is widely adopted to assume all propagated gradients as 1 in backpropagation [37, 43, 50]. Intuitively, the integral of 1 is a certain linear function other than $\text{sign}(\cdot)$, whereas this may lead to inaccurate gradient estimation and produce inconsistent optimization directions in the model training.

To address aforementioned problems, we propose a novel quantization framework, namely BiGeaR, to learn the Binarized Graph Representations for Top-K Recommendation. Based on the natural bipartite graph topology of user-item interactions, we implement BiGeaR with the inspiration from graph-based models, i.e., Graph Convolutional Networks (GCNs) [18, 30]. With the emphasis on deepening the exploration of multi-hop subgraph structures, GCN-based recommender models capture the high-order interactive relations and well simulate Collaborative Filtering for recommendation [21, 56, 65]. Specifically, BiGeaR sketches graph nodes (i.e., users and items) with binarized representations, which facilitates nearly one bit-width representation compression. Furthermore, our model BiGeaR decomposes the prediction formula (i.e., inner product) into bitwise operations (i.e., Popcount and XNOR). This dramatically reduces the number of floating-point operations (#FLOP) and thus introduces theoretically solid acceleration for online inference. To avoid large information loss, as shown in Figure 1(a), BiGeaR technically consists of multi-faceted quantization reinforcement at the pre-, mid-, and post-stage of binarized representation learning:

1. **At the pre-stage of model learning,** we propose the graph layer-wise quantization (from low- to high-order interactions) to consecutively binarize the user-item features with different semantics. Our analysis indicates that such layer-wise quantization can actually achieve the magnification effect of feature uniqueness, which significantly compensates for the limited expressivity of embeddings binarization. The empirical study also justifies that, this is more effective to enrich the quantization informativeness, rather than simply increasing the embedding sizes in the conventional manner [37, 43, 44, 50].

2. **During the mid-stage of embedding quantization,** BiGeaR introduces the self-supervised inference distillation to develop the low-loss ranking capability inheritance. Technically, it firstly extracts several pseudo-positive training samples that are inferred by full-precision embeddings of BiGeaR. Then these samples serve as the direct regularization target to the quantized embeddings, such that they can distill the ranking knowledge from full-precision ones to have similar inference results. Different from the conventional knowledge distillation, our approach is tailored specifically for Top-K recommendation and therefore boosts the performance with acceptable training costs.

3. **As for the post-stage backpropagation of quantization optimization,** we propose to utilize the approximation of Dirac delta function (i.e., $\delta$ function) [6] for more accurate gradient estimation. In contrast to STE, our gradient estimator provides the consistent optimization direction with $\text{sign}(\cdot)$ in both forward and backward propagation. The empirical study demonstrates its superiority over other gradient estimators.

**Empirical Results.** The extensive experiments over five real-world benchmarks show that, BiGeaR significantly outperforms the state-of-the-art quantization-based recommender model by 25.77%~40.12% and 22.08%~39.52% w.r.t. Recall and NDCG metrics. Furthermore, it attains 95.29%~100.32% and 95.32%~101.94% recommendation capability compared to the best-performing full-precision model, with over 8x inference acceleration and space compression.

**Discussion.** It is worthwhile mentioning that BiGeaR is related to hashing-based models (i.e., learning to hash) [27, 28], as, conceptually, binary hashing can be viewed as 1-bit quantization. However, as shown in Figure 1(b), they have different motivations. Hashing-based models are usually designed for fast candidate generation, followed by full-precision re-ranking algorithms for accurate prediction. Meanwhile, BiGeaR is end-to-end that aims to make predictions within the proposed architecture. Hence, we believe BiGeaR is technically related but motivationally orthogonal to them.

**Organization.** We present BiGeaR methodology and model analysis in § 2 and § 3. Then we report the experiments and review the related work in § 4 and § 5 with the conclusion in § 6.

### 2 BiGeaR Methodology

In this section, we formally introduce: (1) graph layer-wise quantization for feature magnification; (2) inference distillation for ranking capability inheritance; (3) gradient estimation for better model optimization. BiGeaR framework is illustrated in Figure 2.(a). The notation table and pseudo-codes are attached in Appendix A and B.

**Preliminaries: graph convolution.** Its general idea is to learn node representations by iteratively propagating and aggregating latent features of neighbors via the graph topology [21, 30, 58]. We adopt the graph convolution paradigm working on the continuous space from LightGCN [21] that recently shows good recommendation performance. Let $\phi^{(l)}_u, \phi^{(l)}_i \in \mathbb{R}^d$ denote the continuous feature embeddings of user $u$ and item $i$ computed after $l$ layers of information propagation. $N(x)$ represents $x$’s neighbor set. They can be iteratively updated by utilizing information from the $(l-1)$-th layer:

\[
\phi^{(l)}_u = \sum_{v \in N(u)} \frac{1}{\sqrt{|N(u)|} \cdot |N(v)|} \phi^{(l-1)}_v, \quad \phi^{(l)}_i = \sum_{u \in N(i)} \frac{1}{\sqrt{|N(i)|} \cdot |N(u)|} \phi^{(l-1)}_u.
\]

#### 2.1 Graph Layer-wise Quantization

We propose the graph layer-wise quantization mainly by computing quantized embeddings and embedding scalers: (1) these quantized embeddings sketch the full-precision embeddings with $d$-dimensional binarized codes (i.e., $\{-1, 1\}^d$); and (2) each embedding scaler reveals the value range of original embedding entries. Specifically, during the graph convolution at each layer, we track the intermediate information (e.g., $\phi^{(l)}_u$) and perform the layer-wise
1-bit quantization in parallel as:
\[ \hat{q}^{(l)}_u = \text{sign}(a^{(l)}_u), \quad \hat{q}^{(l)}_i = \text{sign}(a^{(l)}_i), \]
where embedding segments \( \hat{q}^{(l)}_u, \hat{q}^{(l)}_i \in \{-1, 1\}^d \) retain the node latent features directly from \( a^{(l)}_u \) and \( a^{(l)}_i \). To equip with the layer-wise quantized embeddings, we further include a layer-wise positive embedding scaler for each node (e.g., \( a^{(l)}_u \in \mathbb{R}^d \)), such that \( a^{(l)}_u = a^{(l)}_u \cdot \hat{q}^{(l)}_u \). Then for each entry in \( a^{(l)}_u \cdot \hat{q}^{(l)}_i \), it is still binarized by \( (-a^{(l)}_u, a^{(l)}_u) \). In this work, we compute the mean of L1-norm as:
\[ a^{(l)}_u = \frac{1}{d} \cdot ||a^{(l)}_u||_1, \quad a^{(l)}_i = \frac{1}{d} \cdot ||a^{(l)}_i||_1. \]
Instead of setting \( a^{(l)}_u \) and \( a^{(l)}_i \) as learnable, such deterministic computation is simple yet effective to provide the scaling functionality whilst substantially pruning the parameter search space. The experimental demonstration is in Appendix F.2.

After \( L \) layers of quantization and scaling, we have built the following binarized embedding table for each graph node \( x \) as:
\[ \mathcal{A}_x = \{a^{(0)}_u, a^{(1)}_u, \ldots, a^{(L)}_u\}, \quad \mathcal{Q}_x = \{q^{(0)}_u, q^{(1)}_u, \ldots, q^{(L)}_u\}. \]
(4)
From the technical perspective, BiGeaR binarizes the intermediate semantics at different layers of the receptive field [53, 59] for each node. This, essentially, achieves the magnification effect of feature uniqueness to enrich user-item representations via the interaction graph exploration. We leave the analysis in § 3.1.

2.2 Prediction Acceleration

Model Prediction. Based on the learned embedding table, we predict the matching scores by adopting the inner product:
\[ \hat{y}_{u,i} = \langle f(\mathcal{A}_u, \mathcal{Q}_u), f(\mathcal{A}_i, \mathcal{Q}_i) \rangle, \]
(5)
where function \( f(\cdot, \cdot) \) in this work is implemented as:
\[ f(\mathcal{A}_u, \mathcal{Q}_u) = \left\| w_l a^{(l)}_u \cdot q^{(l)}_u \right\|, \quad f(\mathcal{A}_i, \mathcal{Q}_i) = \left\| w_l a^{(l)}_i \cdot q^{(l)}_i \right\|. \]
Here \( \| \) represents concatenation of binarized embedding segments, in which weight \( w_l \) measures the contribution of each segment in prediction. \( w_l \) can be defined as a hyper-parameter or a learnable variable (e.g., with attention mechanism [53]). In this work, we set \( w_l = 1 \) to linearly increase \( w_l \) for segments from lower-layers to higher-layers, mainly for its computational simplicity and stability.

Computation Acceleration. Notice that for each segment of \( f(\mathcal{A}_u, \mathcal{Q}_u) \), e.g., \( w_l a^{(l)}_u \cdot q^{(l)}_u \), entries are binarized by two values (i.e., \( -w_l a^{(l)}_u \) or \( w_l a^{(l)}_u \)). Thus, we can achieve the prediction acceleration by decomposing Equation (5) with bitwise operations. Concretely, in practice, \( q^{(l)}_u \) and \( q^{(l)}_i \) will be firstly encoded into basic \( d \)-bits binary codes, denoted by \( \tilde{q}^{(l)}_u, \tilde{q}^{(l)}_i \in \{0, 1\}^d \). Then we replace Equation (5) by introducing the following formula:
\[ \hat{y}_{u,i} = \sum_{l=0}^{L} w_l \tilde{a}^{(l)}_u \cdot \tilde{a}^{(l)}_i - (2 \text{Popcount}(\text{XOR}(\tilde{q}^{(l)}_u, \tilde{q}^{(l)}_i)) - d). \]

Compared to the original computation approach in Equation (5), our bitwise-operation-supported prediction in Equation (7) reduces the number of floating-point operations (#FLOP) with Popcount and XOR. We illustrate an example in Figure 2(b).

2.3 Self-supervised Inference Distillation

To alleviate the asymmetric inference capability issue between full-precision representations and binarized ones, we introduce the self-supervised inference distillation such that binarized embeddings can well inherit the inference knowledge from full-precision ones. Generally, we treat our full-precision intermediate embeddings (e.g., \( a^{(l)}_u \)) as the teacher embeddings and the quantized segments as the student embeddings. Given both teacher and student embeddings, we can obtain their prediction scores as \( \hat{y}^{\text{top}}_{u,i} \) and \( \hat{y}^{\text{top}}_{u,i} \). For Top-K recommendation, then our target is to reduce their discrepancy as:
\[ \text{argmin} \mathcal{D}(\hat{y}^{\text{top}}_{u,i}, \hat{y}^{\text{top}}_{u,i}). \]
(8)
A straightforward implementation of function \( \mathcal{D} \) from the conventional knowledge distillation [1, 25] is to minimize their Kullback-Leibler divergence (KLD) or mean squared error (MSE). Despite their effectiveness in classification tasks (e.g., visual recognition [1, 60]), they may not be appropriate for Top-K recommendation as:
- Firstly, both KLD and MSE in \( \mathcal{D} \) encourage the student logits (e.g., \( \hat{y}^{\text{top}}_{u,i} \)) to be similarly distributed with the teacher logits in a macro view. But for ranking tasks, they may not well learn the relative order of user preferences towards items, which, however, is the key to improving Top-K recommendation capability.
- Secondly, both methods develop the distillation over the whole item corpus, which may be computational excessive for realistic model training. As the item popularity usually follows the Long-tail distribution [41, 51], learning the relative order of those frequently interacted items located at the tops of ranking lists actually contributes more to the Top-K recommendation performance.
To develop effective inference distillation, we propose to extract additional pseudo-positive training samples from teacher embeddings to regularize the targeted embeddings on each convolutional layer. Concretely, let $\sigma$ represent the activation function (e.g., Sigmoid). We first pre-train the teacher embeddings to minimize the Bayesian Personalized Ranking (BPR) loss [46]:

$$
L_{BPR} = - \sum_{u \in V} \sum_{(i \in N(u)) \not= (j \in N(u))} \ln \sigma(b^i_{u,j} - b^j_{u,i}) + b_0^i, \quad (9)
$$

where $L_{BPR}$, forces the prediction of an observed interaction to be higher than its unobserved counterparts, and the teacher score $\hat{y}_{u,i}^{ch}$ is computed as $\hat{y}_{u,i}^{ch} = \langle \theta_{u,i}^t, \omega_{u}^{(l)} \rangle$. Please notice that we only disable binarization and its associated gradient estimation in pre-training. After it is well-trained, for each user $u$, we retrieve the layer-wise teacher inference towards all items $I$:

$$
\hat{y}_{u,i}^{ch,(l)} = \langle \theta_{u,i}^t, \omega_{u}^{(l)} \rangle , \quad (10)
$$

Based on the segment scores $\hat{y}_{u,i}^{ch,(l)}$ at the $l$-th layer, we first sort Top-$R$ items with the highest matching scores, denoted by $S_s^{(l)}(u)$. And hyper-parameter $R \ll |I|$. Inspired by [51], then we propose our layer-wise inference distillation as follows:

$$
L_{ID}(u) = \sum_{l=0}^{L} \frac{\lambda}{R} \sum_{k=1}^{R} \frac{1}{R} \sum_{i \in I} \ln \sigma(\hat{y}_{u,i}^{sd,(l)} - S_s^{(l)}(u)_{k}), \quad (11)
$$

where student scores $\hat{y}_{u,i}^{sd,(l)}$ is computed similarly to Equation (10) and $S_s^{(l)}(u)_{k}$ returns the $k$-th high-scored item from the pseudo-positive samples. $\lambda$ is the ranking-aware weight presenting two major effects: (1) since samples in $S_s^{(l)}(u)$ are not necessarily all ground-truth positive, $\lambda$ thus balances their contribution to the overall loss; (2) it dynamically adjusts the weight importance for different ranking positions in $S_s^{(l)}(u)$. To achieve these, $\lambda$ can be developed by following the parameterized geometric distribution for approximating the tailed item popularity [45]:

$$
\lambda = \lambda_1 \exp(-\lambda_2 \cdot k), \quad (12)
$$

where $\lambda_1$ and $\lambda_2$ control the loss contribution level and sharpness of the distribution. Intuitively, $L_{ID}$ encourages highly-recommended items from full-precision embeddings to more frequently appear in the student’s inference list. Moreover, our distillation approach regularizes the embedding quantization in a layer-wise manner as well; this will effectively narrow their inference discrepancy for a more correlated recommendation capability.

**Objective Function.** Combining $L_{BPR}^{tid}$ that calculates BPR loss (similar to Equation (9)) with the student predictions from Equation (5) and $L_{ID}$ for all training samples, our final objective function for learning embedding binarization is defined as:

$$
L = L_{BPR}^{tid} + L_{ID} + \lambda \| \theta \|^2_2, \quad (13)
$$

where $\| \theta \|^2_2$ is the L2-regularizer of node embeddings parameterized by hyper-parameter $\lambda$ to avoid over-fitting.

### 2.4 Gradient Estimation

Although Straight-Through Estimator (STE) [4] enables an executable gradient flow for backpropagation, it however may cause the issue of inconsistent optimization direction in forward and backward propagation: as the integral of the constant 1 in STE is a linear function, other than $\text{sign}(\cdot)$ function. To furnish more accurate gradient estimation, in this paper, we utilize the approximation of Dirac delta function [6] for gradient estimation.

Concretely, let $u(\phi)$ denote the unit-step function, a.k.a., Heaviside step function [14], where $u(\phi) = 1$ for $\phi > 0$ and $u(\phi) = 0$ otherwise. Obviously, we can take a translation from step function to sign($\phi$) by $\text{sign}(\phi) = 2u(\phi) - 1$, and thus theoretically $\frac{\partial \text{sign}(\phi)}{\partial \phi} = 2u(\phi)$. As for $\frac{\partial u(\phi)}{\partial \phi}$, it has been proved [6] that, $\frac{\partial u(\phi)}{\partial \phi} = 0$ if $\phi \neq 0$, and $\frac{\partial u(\phi)}{\partial \phi} = \infty$ otherwise, which exactly is the Dirac delta function, a.k.a., the unit impulse function $\delta(\cdot)$ shown in Figure 3(a). However, it is still intractable to directly use $\delta(\cdot)$ for gradient estimation. A feasible solution is to approximate $\delta(\cdot)$ by introducing zero-centered Gaussian probability density as follows:

$$
\delta(\phi) = \lim_{\beta \rightarrow \infty} \frac{1}{\sqrt{2\pi(\beta^2)}} \exp(-\beta^2 \phi^2), \quad (14)
$$

which implies that:

$$
\frac{\partial \text{sign}(\phi)}{\partial \phi} = \delta(\phi) \equiv \frac{2}{\sqrt{\pi}} \exp(-\phi^2). \quad (15)
$$

As shown in Figure 3(b)-(c), hyper-parameter $\theta$ determines the sharpness of the derivative curve for approximation to $\text{sign}(\cdot)$.

Intuitively, our proposed gradient estimator follows the main direction of factual gradients with $\text{sign}(\cdot)$ in model optimization. This will produce a coordinated value quantization from continuous embeddings to quantized ones, and thus a series of evolving gradients can be estimated for the inputs with diverse value ranges. As we will show in §4.6 of experiments, our gradient estimator can work better than other recent estimators [12, 17, 38, 43, 62].

### 3 Model Analysis

#### 3.1 Magnification of Feature Uniqueness

We take user $u$ as an example for illustration and the following analysis can be popularized to other nodes without loss of generality. Similar to sensitivity analysis in statistics [31] and influence diffusion in social networks [61], we measure how the latent feature of a distant node $x$ finally affects $u$’s representation segments before binarization (e.g., $e^{(l)}_u$), supposing $x$ is a multi-hop neighbor of $u$. We denote the feature enrichment ratio $\varepsilon^{(l)}_{x \rightarrow u}$ as the L1-norm of Jacobian matrix $\left[ \frac{\partial e^{(l)}_u}{\partial e^{(0)}_x} \right]$, by detecting the absolute influence of all fluctuation in entries of $e^{(0)}_x$ to $e^{(l)}_u$, i.e., $\varepsilon^{(l)}_{x \rightarrow u} = \left\| \left[ \frac{\partial e^{(l)}_u}{\partial e^{(0)}_x} \right] \right\|_1$, focusing on a $l$-length path $h$ connected by the node sequence: $x_1, x_2, \cdots, x_l$, where $x_l = u$ and $x_0 = x$, we follow the chain rule to derive the derivative decomposition as:

$$
\frac{\partial e^{(l)}_u}{\partial e^{(0)}_x} = \sum_{h=1}^{H} \frac{H}{h} \prod_{k=1}^{h-1} \sqrt{\frac{N(x_k)}{N(x_k+h)}} \prod_{k=h}^{H} \frac{1}{\sqrt{N(x_k+h)}} I, \quad (16)
$$

where $H$ is the number of paths between $x$ and $x$ in total. Since all factors in the computation chain are positive, then:

$$
\varepsilon^{(l)}_{x \rightarrow u} = \left\| \frac{\partial e^{(l)}_u}{\partial e^{(0)}_x} \right\|_1 = d \cdot \sqrt{\frac{N(x)}{N(x+h)}} \prod_{h=1}^{H} \frac{1}{\sqrt{N(x+h)}}, \quad (17)
$$

![Figure 3: Gradient estimation.](image-url)
Note that here the term $\sum_{i=1}^{k} \prod_{j=1}^{d} \frac{1}{|N(x_i)|}$ is exactly the probability of the $l$-length random walk starting at $u$ that finally arrives at $x$, which can be interpreted as:

$$\mathbb{P}_{x \rightarrow \alpha}^{(l)} = \frac{1}{\sqrt{|N(x)|}} \cdot \text{Prob}(l\text{-step random walk from $u$ arrives at $x$}). \quad (18)$$

**Magnification Effect of Feature Uniqueness.** Equation (18) implies that, with the equal probability to visit adjacent neighbors, distant nodes with fewer degrees (i.e., $|N(x)|$) will contribute more feature influence to user $u$. But most importantly, in practice, these $l$-hop neighbors of user $u$ usually represent certain esoteric and unique objects with less popularity. By collecting the intermediate information in different depth of the graph convolution, we can achieve the feature magnification effect for all unique nodes that live within $L$ hops of graph exploration, which finally enrich $u$’s semantics in all embedding segments for quantization.

### 3.2 Complexity Analysis

To discuss the feasibility for realistic deployment, we compare BiGeaR with the best full-precision model LightGCN [21], as they are end-to-end with offline model training and online prediction.

**Training Time Complexity.** $M$, $N$, and $E$ represent the number of users, items, and edges; $S$ and $B$ are the epoch number and batch size. We use BiGeaR$_{tech}$ and BiGeaR$_{std}$ to denote the pre-training version and binarized one, respectively. As we can observe from Table 1, (1) both BiGeaR$_{tech}$ and BiGeaR$_{std}$ takes asymptotically similar complexity of graph convolution with LightGCN (where BiGeaR$_{std}$ takes additional $O(2SD(L+1)E)$ complexity for binarization). (2) For $L_{BPR}$ computation, to prevent over-smoothing issue [33, 35], usually $L \leq 4$; compare to the convolution operation, the complexity of $L_{BPR}$ is acceptable. (3) For $L_{ID}$ preparation, after the training of BiGeaR$_{tech}$, we firstly obtain the layer-wise prediction scores with $O(MNd(L + 1))$ time complexity and then sort out the Top-$R$ pseudo-positive samples for each user with $O(N + R \times N)$. For BiGeaR$_{std}$, it takes a layer-wise inference distillation from BiGeaR$_{tech}$ with $O(SRD(L+1)E)$. (4) To estimate the gradients for BiGeaR$_{std}$, it takes $O(2SD(L + 1)E)$ for all training samples.

| Table 1: Training time complexity. |
|-----------------------------------|
| |
| **Graph Normalization** | LightGCN | BiGeaR$_{tech}$ | BiGeaR$_{std}$ |
| **Conv. and Quant.** | $O(2SD)$ | $O(2SD)$ | $O(2SD) \cdot \frac{1}{S} \cdot \frac{1}{B}$ | $O(2SD(L + 1)E)$ |
| $L_{BPR}$ Loss | $O(SRD)$ | $O(SRD(L+1)E)$ | $O(SRD(L+1)E)$ |
| $L_{ID}$ Loss | $O(MNd(L + 1)+N \times R)$ | $O(SRD(L+1)E)$ |
| Gradient Estimation | $O(2SD(L + 1)E)$ | $O(2SD(L + 1)E)$ |

**Memory overhead and Prediction Acceleration.** We measure the memory footprint of embedding tables for online prediction. As we can observe from the results in Table 2: (1) Theoretically, the embedding size ratio of our model over LightGCN is $O(\frac{1}{S})$. Normally, $L \leq 4$ and $d \geq 64$, thus our model achieves at least $4 \times$ space cost compression. (2) As for the prediction time cost, we compare the number of binary operations ($\#BOP$) and floating-point operations ($\#FLOP$) between our model and LightGCN. We find that BiGeaR replaces most of floating-point arithmetics (e.g., multiplication) with bitwise operations.

### 4 Experimental Results

We evaluate our model on Top-K recommendation task with the aim of answering the following research questions:

- **RQ1.** How does BiGeaR perform compared to state-of-the-art full-precision and quantization-based models?
- **RQ2.** How is the practical resource consumption of BiGeaR?
- **RQ3.** How do proposed components affect the performance?

#### 4.1 Experiment Setup

**Datasets.** To guarantee the fair comparison, we directly use five widely evaluated datasets (including the training/test splits) from: MovieLens$^4$ [9, 10, 24, 50], Gowalla$^5$ [21, 30, 56, 57], Pinterest$^6$ [15, 50], Yelp2018$^7$ [21, 56, 57], Amazon-Book$^8$ [21, 56, 57]. Dataset statistics and descriptions are reported in Table 3 and Appendix C.

#### 4.2 Evaluation Metric

**In Top-K recommendation evaluation, we select two widely-used evaluation protocols Recall@K and NDCG@K to evaluate Top-K recommendation capability.**

**Competing Methods.** We include the following recommender models: (1) 1-bit quantization-based methods including graph-based (GumbelRec [26, 40, 68], HashGNN [50]) and general model-based (LSH [16], HashNet [7], CIGAR [28]), and (2) full-precision models including neural-network-based (NeuCF [23]) and graph-based (NGCF [56], DGCN [57], LightGCN [21]). Detailed introduction of these methods are attached in Appendix D.

We exclude early quantization-based recommendation baselines, e.g., CH [39], DiscreteCF [66], DPR [67], and full-precision solutions, e.g., GC-MC [5], PinSage [65], mainly because the above competing models [21, 23, 28, 56] have validated the superiority.

**Experiment Settings.** Our model is implemented by Python 3.7 and PyTorch 1.14.0 with non-distributed training. The experiments are run on a Linux machine with 1 NVIDIA V100 GPU, 4 Intel Core i7-8700 CPUs, 32 GB of RAM with 3.20GHz. For all the baselines, we follow the official reported hyper-parameter settings, and for methods lacking recommended settings, we apply a grid search for hyper-parameters. The embedding dimension is searched in [32, 64, 128, 256, 512, 1024]. The learning rate $\eta$ is tuned within $[10^{-4}, 10^{-3}, 10^{-2}]$ and the coefficient of $L_2$ normalization $\lambda$ is tuned among $[10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}]$.

| Table 2: Complexity of space cost and online prediction. |
|----------------------------------------------------------|
| Embedding size | #FLOP | #BOP |
|----------------|--------|------|
| LightGCN | $O(2(M + N)d)$ | $O(2MNd)$ | - |
| BiGeaR | $O((M + N)(L + 1)(32 + d))$ | $O(2MN(L + 1)d)$ | - |

---

$^4$https://grouplens.org/datasets/movielens/1m/
$^5$https://github.com/gusye1234/LightGCN-PyTorch/tree/master/data/gowalla
$^6$https://sites.google.com/site/xueatalphabeta/dataset-1/pinterest_iccv
$^7$https://github.com/gusye1234/LightGCN-PyTorch/tree/master/data/yelp2018
$^8$https://github.com/gusye1234/LightGCN-PyTorch/tree/master/data/amazon-book
We initialize and optimize all models with default normal initializer and Adam optimizer [29]. We report all hyper-parameters in Appendix E for reproducibility.

4.2 Performance Analysis (RQ1)

We evaluate Top-K recommendation by varying K in [20, 40, 60, 80, 100]. We summarize the Top@20 results in Table 4 for detailed comparison and plot the Top-K recommendation curves in Appendix F.1. From Table 4, we have the following observations:

- **Our model offers a competitive recommendation capability to state-of-the-art full-precision recommender models.**
  1. BiGeaR generally outperforms most of full-precision recommender models excluding LightGCN over five benchmarks. The main reason is that our model and LightGCN take similar graph convolution methodology with network simplification [21], e.g., removing self-connection and feature transformation, which is proved to be effective for Top-K ranking and recommendation. Moreover, BiGeaR collects the different levels of interactive information in multi-depths of graph exploration, which significantly enriches semantics to user-item representations for binarization. (2) Compared to the state-of-the-art method LightGCN, our model develops about 95% capability w.r.t. Recall@20 and NDCG@20 throughout all datasets. This shows that our proposed model designs are effective to narrow the performance gap with full-precision model LightGCN. Although the binarized embeddings learned by BiGeaR may not achieve the exact expressivity parity with the full-precision ones learned by LightGCN, considering the advantages of space compression and inference acceleration that we will present later, we argue that such performance capability is acceptable, especially for those resource-limited deployment scenarios.
  2. Compared to all binarization-based recommender models, BiGeaR presents the empirically remarkable and statistically significant performance improvement. (1) Two conventional methods (LSH, HashNet) for general item retrieval tasks underperform CIGAR, HashGNN and BiGeaR, showing that a direct model adaptation may be too trivial for Top-K recommendation. (2) Compared to CIGAR, graph-based models generally work better. This is mainly because, CIGAR combines general neural networks with \textit{learning to hash} techniques for fast candidate generation; on the contrary, graph-based models are more capable of exploring multi-hop interaction subgraphs to directly simulate the high-order collaborative filtering process for model learning. (3) Our model further outperforms HashGNN by about 26%–40% and 22%–40% w.r.t. Recall@20 and NDCG@20, proving the effectiveness of our proposed multi-faceted optimization components in embedding binarization. (4) Moreover, the significance test in which p-value < 0.05 indicates that the improvements over all five benchmarks are statistically significant.

4.3 Resource Consumption Analysis (RQ2)

We analyze the resource consumption in \textit{training}, \textit{online inference}, and \textit{memory footprint} by comparing to the best two competing models, i.e., LightGCN and HashGNN. Due to the page limits, we report the empirical results of MovieLens dataset in Table 5.

(1) \(T_{\text{train}}\): we set batch size \( B = 2048 \) and dimension size \( d = 256 \) for all models. We find that HashGNN is fairly time-consuming than LightGCN and BiGeaR. This is because HashGNN adopts the early GCM framework [18] as the backbone; LightGCN and BiGeaR utilize more simplified graph convolution architecture in which operations such as self-connection, feature transformation, and nonlinear activation are all removed [21]. Furthermore, BiGeaR needs 5.1s and 6.2s per epoch for pretraining and quantization, both of which take slightly more yet asymptotically similar time cost with LightGCN, basically following the complexity analysis in § 3.2.

(2) \(T_{\text{infer}}\): we randomly generate 1,000 queries for online prediction and conduct experiments with the vanilla NumPy\(^7\) on CPUs.

\[\text{https://www.lfd.uci.edu/~gohlke/pythonlibs/}\]
4.4 Study of Layer-wise Quantization (RQ3.A)  
To verify the magnification of feature uniqueness in layer-wise quantization, we modify BiGeaR and propose two variants, denoted as BiGeaR\(_{w/o LW}\) and BiGeaR\(_{w/o FU}\). We report the results in Figure 4 by denoting Recall@20 and NDCG@20 in red and blue, respectively, and vary color brightness for different variants. From these results, we have the following explanations.  
- Firstly, BiGeaR\(_{w/o LW}\) discards the layer-wise quantization and adopts the conventional manner by quantizing the last outputs from L convolution iterations. We fix dimension \(d = 256\) and vary layer number \(L\) for BiGeaR, and only vary dimension \(d\) for variant BiGeaR\(_{w/o LW}\) with fixed \(L = 2\). (1) Even by continuously increasing the dimension size from 64 to 1024, BiGeaR\(_{w/o LW}\) slowly improves both Recall@20 and NDCG@20 performance. (2) By contrast, our layer-wise quantization presents a more efficient capability in improving performance by increasing \(L\) from 0 to 3. When \(L = 4\), BiGeaR usually exhibits a conspicuous performance decay, mainly because of the common over-smoothing issue in graph-based models [33, 35]. Thus, with a moderate size \(d = 256\) and convolution number \(L \leq 3\), BiGeaR can attain better performance with acceptable computational complexity.  
- Secondly, BiGeaR\(_{w/o FU}\) omits the feature magnification effect by adopting the way used in HashGNN [18, 50] as:  
\[
\hat{x}_k^{(l)} = \frac{1}{|N(x)|} \sum_{z \in N(x)} x_z^{(l-1)}.
\]  
(19)  
Similar to the analysis in § 3.1, such modification will finally disable the “magnification term” in Equation (18) and simplify it to the vanilla random walk for graph exploration. Although BiGeaR\(_{w/o FU}\) presents similar curve trends with BiGeaR when \(L\) increases, the general performance throughout all five datasets is unsatisfactory compared to BiGeaR. This validates the effectiveness of BiGeaR’s effort in magnifying unique latent features, which enriches user-item representations and boosts Top-K recommendation performance accordingly.

### 4.5 Study of Inference Distillation (RQ3.B)  
4.5.1 Effect of Layer-wise Distillation. We study the effectiveness of our inference distillation by proposing two ablation variants, namely noID and endID. While noID totally removes our information distillation in model training, endID downgrades the original layer-wise distillation to only distill information from the last layer of graph convolution. As shown in Table 6, both noID and endID draw notable performance degradation. Furthermore, the performance gap between endID and BiGeaR shows that it is efficacious to conduct our inference distillation in a layer-wise manner for further performance enhancement.

| Table 6: Learning inference distillation. |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| Variant         | MovieLens       | Gowalla         | Pinterest       | Yelp2018        | Amazon-book     |
|                 | R@20 | N@20 | R@20 | N@20 | R@20 | N@20 | R@20 | N@20 | R@20 | N@20 |
| noID            | 24.40 | 44.06 | 17.85 | 24.28 | 15.23 | 16.38 | 6.18 | 11.22 | 4.07 | 7.31 |
| endID           | 25.82 | 44.79 | 18.05 | 24.73 | 15.28 | 16.58 | 6.29 | 11.37 | 4.44 | 7.78 |
| KLD             | 24.32 | 44.38 | 17.63 | 24.07 | 14.78 | 15.92 | 5.82 | 10.36 | 4.13 | 7.21 |
| BiGeaR          | 25.57 | 45.56 | 18.36 | 24.96 | 15.57 | 16.83 | 6.47 | 11.60 | 4.68 | 8.12 |

4.5.2 Conventional Knowledge Distillation. To compare with the conventional approach, we modify BiGeaR by applying KL divergence for layer-wise teacher and student logits, i.e., \(\tilde{y}_u^{(l)}\) vs. \(y_u^{(l)}\). We denote this variant as KLD. As we can observe from Table 6, using conventional knowledge distillation with KL divergence leads to suboptimal performance. This is because KL divergence encourages the teacher and student training objects to have a similar logit distribution, but users’ relative order of item preference can not be well learned from this process. Compared to the conventional approach, our proposed layer-wise Inference distillation is thus more effective for ranking information distillation.

### 4.6 Study of Gradient Estimation (RQ3.C)  
We compare our gradient estimation with several recently studied estimators, such as Tanh-like [17, 43], SSwish [12], Sigmoid [62],...
and **projected-based estimator** [38] (denoted as PBE), by implementing them in BiGeaR. We report their Recall@20 in Figure 5 and compute the performance gain of our approach over these estimators accordingly. We have two main observations:

(1) Our proposed approach shows the consistent superiority over all other gradient estimators. These estimators usually use *visually similar* functions, e.g., tanh(·), to approximate sign(·). However, these functions are not necessarily *theoretically relevant* to sign(·). This may lead to inaccurate gradient estimation. On the contrary, as we explain in § 2.4, we transfer the unit-step function \( u(\cdot) \) to sign(·) by sign(·) = 2u(·) - 1. Then we can further estimate the gradients of sign(·) with the approximated derivatives of \( u(\cdot) \). In other words, our approach follows the main optimization direction of factual gradients with sign(·); and different from previous solutions, this guarantees the coordination in both forward and backward propagation.

(2) Furthermore, compared to the last four datasets, MovieLens dataset confronts a larger performance disparity between our approach and others. This is because MovieLens dataset is much denser than the other datasets, i.e., \( \text{Interactions} = 0.0419 > \{0.00084, 0.00267, 0.0013, 0.00062\} \), which means that users tend to have more item interactions and complicated preferences towards different items. Consequently, this posts a higher requirement for the gradient estimation capability in learning ranking information. Fortunately, the empirical results in Figure 5 demonstrate that our solution well fulfills these requirements, especially for dense interaction graphs.

5 Related Work

**Full-precision recommender models.** (1) **Collaborative Filtering (CF)** is a prevalent methodology in modern recommender systems [11, 64, 65]. Earlier CF methods, e.g., **Matrix Factorization** [32, 46], reconstruct historical interactions to learn user-item embeddings. Recent neural-network-based models, e.g., **NeuCF** [23] and attention-based models [8, 22], further boost performance with neural networks. (2) **Graph-based methods** focus on studying the interaction graph topology for recommendation. Graph convolutional networks (GCNs) [18, 30] inspire early work, e.g., **GC-MC** [5], **PinSage** [65], and recent models, e.g., **NGCF** [56], **DGCF** [57], and **LightGCN** [21], mainly because they can well simulate the high-order CF signals among high-hop neighbors for recommendation.

**Learning to hash.** Hashing-based methods map dense floating-point embeddings into binary spaces for **Approximate Nearest Neighbor (ANN)** search acceleration. A representative model LSH [16] has inspired a series of work for various tasks, e.g., fast retrieval of images [7], documents [34, 69], and categorical information [27]. For Top-K recommendation, models like **DCF** [66], **DPR** [67] include neural network architectures. A recent work **CIGAR** [28] proposes adaptive model designs for fast candidate generation. To investigate the graph structure of user-item interactions, model **HashGNN** [50] applies hashing techniques into graph neural networks for recommendation. However, one major issue is that solely using learned binary codes for prediction usually draws a large performance decay. Thus, to alleviate the issue, **CIGAR** further equips with additional full-precision recommender models (e.g., **BPR-MF** [46]) for fine-grained *re-ranking*; **HashGNN** proposes relaxed version by mixing full-precision and binary embedding codes.

**Quantization-based models.** Quantization-based models share similar techniques with hashing-based methods, e.g., **sign(·)** is usually adopted mainly because of its simplicity. However, quantization-based models do not pursue extreme encoding compression, and thus they develop multi-bit, 2-bit, and 1-bit quantization for performance adaptation. Recently, there is growing attention to quantize graph-based models, such as **Bi-GCN** [54] and **BGCN** [2]. However, these two models are mainly designed for geometric classification tasks, but their capability in product recommendation is unclear. Thus, in this paper, we propose BiGeaR to learn 1-bit user-item representation quantization for Top-K recommendation. Different from binary hashing-based methods, BiGeaR aims to make predictions within its own framework, making a balanced trade-off between efficiency and performance.

6 Conclusion and Future Work

In this paper, we present BiGeaR to learn binarized graph representations for recommendation with multi-faceted binarization techniques. The extensive experiments not only validate the performance superiority over competing binarization-based recommender systems, but also justify the effectiveness of all proposed model components. In the future, we plan to investigate two major possible problems. (1) It is worth developing binarization techniques for model-agnostic recommender systems with diverse learning settings [47, 48, 63]. (2) Instead of using **sign(·)** for quantization, developing compact multi-bit quantization methods with similarity-preserving is promising to improve ranking accuracy.

Acknowledgments

The work described in this paper was partially supported by the National Key Research and Development Program of China (No. 2018AAA0100204), the Research Grants Council of the Hong Kong Special Administrative Region, China (CUHK 2410021, Research Impact Fund, No. R5034-18), and the CUHK Direct Grant (4055147).

References

[1] Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E Dahl, and Geoffrey E Hinton. 2018. Scalable methods for 8-bit training of neural networks. arXiv.

[2] Mehdi Bahri, Gaëtan Bahl, and Stefanos Zafeiriou. 2021. Binary Graph Neural Networks. In CVPR, 9492–9501.

[3] Ron Banner, Kay Hubara, Eldad Hoffer, and Daniel Soudry. 2018. Scalable methods for 8-bit training of neural networks. NeurIPS 52.

[4] Yoshua Bengio, Nicholas Leonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv.

[5] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2017. Graph convolutional matrix completion. arXiv preprint arXiv:1706.02263.
A Notation Table

We list key notations in Table 1.

B Pseudo-codes of BiGeaR

The pseudo-codes of BiGeaR are attached in Algorithm 1.

C Datasets

- **MovieLens** [9, 10, 24, 50] is a widely adopted benchmark for movie recommendation. Similar to the setting in [9, 24, 50], \( y_{ui} = 1 \) if user \( u \) has an explicit rating towards item \( i \), otherwise \( y_{ui} = 0 \). In this paper, we use the MovieLens-1M data split.

- **Gowalla** [21, 50, 56, 57] is the check-in dataset [36] collected from Gowalla, where users share their locations by check-in. To guarantee the quality of the dataset, we extract users and items with no less than 10 interactions similar to [21, 50, 56, 57].

- **Pinterest** [15, 50] is an implicit feedback dataset for image recommendation [15]. Users and images are modeled in a graph.

 Edges represent the pins over images initiated by users. In this dataset, each user has at least 20 edges.

- **Yelp2018** [21, 56, 57] is collected from Yelp Challenge 2018 Edition, where local businesses such as restaurants are treated as items. We retain users and items with over 10 interactions similar to [21, 56, 57].

- **Amazon-Book** [21, 56, 57] is organized from the book collection of Amazon-review for product recommendation [20]. Similarly to [21, 56, 57], we use the 10-core setting to graph nodes.
D Competing Methods

- LSH [16] is a representative hashing method to approximate the similarity search for massive high-dimensional data. We follow the adaptation in [50] to it for Top-K recommendation.
- HashNet [7] is a state-of-the-art deep hashing method that is originally proposed for multimedia retrieval tasks. We use the same adaptation strategy in [50] to it for recommendation.
- CIGAR [28] is a hashing-based method for fast item candidate generation, followed by complex full-precision re-ranking algorithms. We use its quantization part for fair comparison.
- GumbelRec is a variant of our model with the implementation of Gumbel-softmax for categorical variable quantization [26, 40, 68]. GumbelRec utilizes the Gumbel-softmax trick to replace sign(·) function for embedding binarization.
- HashGNN [50] is the state-of-the-art end-to-end 1-bit quantization recommender system. HashGNN denotes its vanilla hard encoding version; and HashGNN* is the relaxed version of replacing several quantized digits with the full-precision ones.
- NeurCF [23] is a classical neural network model to capture user-item nonlinear feature interactions for collaborative filtering.
- NGCF [56] is a state-of-the-art graph-based collaborative filtering model that largely follows the standard GCN [30].
- DGCF [57] is one of the latest graph-based models that learns disentangled user intents for better Top-K recommendation.
- LightGCN [21] is another latest GCN-based recommender system that presents a more concise and powerful model structure with state-of-the-art performance.

E Hyper-parameter Settings

We report all hyper-parameter settings in Table 2.

Table 2: Hyper-parameter settings for the five datasets.

|        | MovieLens | Gowalla | Pinterest | Yelp2018 | Amazon-Book |
|--------|-----------|---------|-----------|----------|-------------|
| B      | 2048      | 2048    | 2048      | 2048     | 2048        |
| d      | 256       | 256     | 256       | 256      | 256         |
| η      | 1 × 10^{-3}| 1 × 10^{-3}| 5 × 10^{-4}| 5 × 10^{-4}| 5 × 10^{-4}|
| λ      | 1 × 10^{-4}| 5 × 10^{-3}| 1 × 10^{-4}| 1 × 10^{-4}| 1 × 10^{-4}|
| λ₁      | 1         | 1       | 1         | 1        | 1           |
| λ₂      | 0.1       | 0.1     | 0.1       | 0.1      | 0.1         |
| γ      | 1         | 1       | 1         | 1        | 1           |
| L      | 2         | 2       | 2         | 2        | 2           |

F Additional Experimental Results

F.1 Top-K Recommendation Curve

We curve the Top-K recommendation by varying K from 20 to 100 and compare BiGeaR with several selected models. As shown in Figure 1, BiGeaR consistently presents the performance superiority over HashGNN, and shows the competitive recommendation accuracy with DGCF and LightGCN.

F.2 Implementation of Embedding Scaler α(I)

We set the embedding scaler to learnable (denoted by LB) and show the results in Table 3. We observe that, the design of learnable embedding scaler does not achieve the expected performance. This is probably because there is no direct mathematical constraint to it and thus the parameter search space is too large to find the optimum by stochastic optimization.

F.3 Implementation of w₁.

We try the following three additional implementation of w₁ and report the results in Tables 4.

1. w₁ = 1 + x denotes equally contributes for all embedding segments.
2. w₁ = 1 + x + x is positively correlated to the 1 value, so as to highlight higher-order structures of the interaction graph.
3. w₁ = z^{s(x-1)} is positively correlated to l with exponentiation.

The experimental results show that implementation (2) performs fairly well compared to the others, demonstrating the importance of highlighting higher-order graph information. This corroborates the design of our implementation in BiGeaR, i.e., w₁ ∝ l, which however is simpler and effective with better recommendation accuracy.

F.4 Implementation of w_k.

We further evaluate different w_k:

1. w_k = 1 is inversely proportional to the ranking position k.
2. w_k = 1 is inversely proportional to position k.
3. w_k = z^{-k} is exponential to the value of k.

We observe from Table 5 that the implementation (3) works slightly worse than Equation (12) but generally better than the other two methods. This show that the exponential modeling is more effective to depict the importance contribution of items for approximating the tailed item popularity [45]. Moreover, Equation (12) introduces hyper-parameters to provide the flexibility of adjusting the function properties for different datasets.

Table 3: Implementation of Embedding Scaler.

|        | MovieLens | Gowalla | Pinterest | Yelp2018 | Amazon-Book |
|--------|-----------|---------|-----------|----------|-------------|
| LB     | 23.07     | 41.42   | 17.01     | 23.11    | 14.19       |
|        | 0.78%     | 0.69%   | 7.35%     | 7.41%    | 8.86%       |
| BiGeaR | 25.37     | 45.56   | 18.36     | 24.96    | 15.57       |
|        | 6.47      | 11.60   | 6.47      | 11.60    | 4.68        |

Table 4: Implementation of w₁.

|        | MovieLens | Gowalla | Pinterest | Yelp2018 | Amazon-Book |
|--------|-----------|---------|-----------|----------|-------------|
| (1)    | 22.75     | 41.13   | 16.15     | 21.82    | 14.16       |
| (2)    | 25.07     | 44.64   | 17.81     | 24.46    | 15.26       |
| (3)    | 21.23     | 37.81   | 15.24     | 20.71    | 12.93       |
| Best   | 25.57     | 45.56   | 18.36     | 24.96    | 15.57       |

Table 5: Implementation of w_k.

|        | MovieLens | Gowalla | Pinterest | Yelp2018 | Amazon-Book |
|--------|-----------|---------|-----------|----------|-------------|
| (1)    | 24.97     | 44.33   | 17.96     | 24.97    | 15.11       |
| (2)    | 25.08     | 45.19   | 17.95     | 24.95    | 15.18       |
| (3)    | 25.16     | 44.92   | 18.32     | 24.81    | 15.26       |
| Best   | 25.57     | 45.56   | 18.36     | 24.96    | 15.57       |