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

Yankai Chen
The Chinese University of Hong Kong
ykchen@cse.cuhk.edu.hk

Huifeng Guo
Huawei Noah's Ark Lab
huifeng.guo@huawei.com

Yingxue Zhang
Huawei Noah's Ark Lab
yingxue.zhang@huawei.com

Chen Ma
City University of Hong Kong
chenma@cityu.edu.hk

Ruiming Tang
Jingjie Li
Huawei Noah's Ark Lab
tangruiming@huawei.com
lijingjie1@huawei.com

Irwin King
The Chinese University of Hong Kong
king@cse.cuhk.edu.hk

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). We introduce multi-faceted quantization reinforcement at the *pre-*, *mid-*, and *post-stage* of binarized representation learning, which substantially retains the informativeness against embedding binarization. In addition to saving the memory footprint, it 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; Embedding Binarization; Graph Convolutional Network; Graph Representation

ACM Reference Format:
Yankai Chen, Huifeng Guo, Yingxue Zhang, Chen Ma, Ruiming Tang, Jingjie Li, and Irwin King. 2022. Learning Binarized Graph Representations with Multi-faceted Quantization Reinforcement for Top-K Recommendation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534678.3539452

KDD '22, August 14–18, 2022, Washington, DC, USA.

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539452

1 Introduction

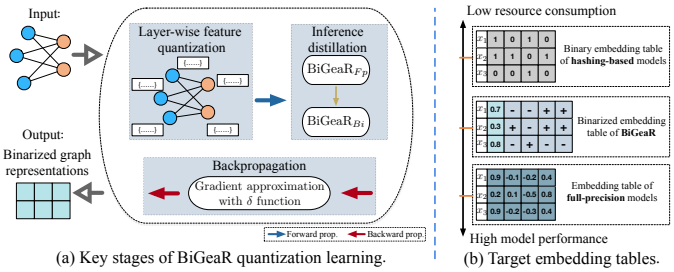
Recommender systems, aiming to perform personalized information filtering [64], are versatile to many Internet applications. Learning vectorized user-item representations (i.e., embeddings), as the core of various recommender models, is the prerequisite for online inference of user-item interactions [21, 55]. With the explosive data expansion (e.g., Amazon owns over 150M users and 350M products [51]), 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*[1] embedding table) on certain data centers [48], and therefore hinders the deployment to devices with limited resources [48].

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, 49].

Despite the promising potentiality, it is still challenging to develop realistic deployment mainly because of the large performance degradation in Top-K recommendation [28, 49]. 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, 69], sign(·) function is widely adopted to achieve $O(1)$ embedding binarization [37, 49, 54]. However, this only guarantees the sign (+/-) correlation for each embedding entry.

[1] It could be single-precision or double-precision; we use float32 as the default for explanation and conducting experiments throughout this paper.



(a) Key stages of BiGeaR quantization learning. (b) Target embedding tables.

Figure 1: Illustration of BiGeaR.

Compared to the original full-precision embeddings, binarized targets produced from sign(·) 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, 49]. However, such hidden knowledge is vital to reveal users' preference towards different items, losing of which may thus draw degraded ranking capability and sub-optimal model learning accordingly.
- **Inaccurate gradient estimation.** Due to the non-differentiability of quantization function sign(·), *Straight-Through Estimator* (STE) [4] is widely adopted to assume all propagated gradients as 1 in backpropagation [37, 42, 49]. Intuitively, the integral of 1 is a certain linear function other than sign(·), 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, 55, 64]. Specifically, BiGeaR sketches graph nodes (i.e., users and items) with binarized representations, which facilitates nearly one bit-width representation compression. Furthermore, 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, 42, 43, 49].

(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., δ function) [6] for more **accurate gradient estimation**. In contrast to STE, our gradient estimator provides the consistent optimization direction with sign(·) 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 8× 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, 57]. We adopt the graph convolution paradigm working on the continuous space from LightGCN [21] that recently shows good recommendation performance. Let $\boldsymbol{v}_u^{(l)}, \boldsymbol{v}_i^{(l)} \in \mathbb{R}^d$ denote the continuous feature embeddings of user u and item i computed after l layers of information propagation. $\mathcal{N}(x)$ represents x's neighbor set. They can be iteratively updated by utilizing information from the (l-1)-th layer:

$$\boldsymbol{v}_u^{(l)} = \sum_{i \in \mathcal{N}(u)} \frac{1}{\sqrt{|\mathcal{N}(u)| \cdot |\mathcal{N}(i)|}} \boldsymbol{v}_i^{(l-1)}, \quad \boldsymbol{v}_i^{(l)} = \sum_{u \in \mathcal{N}(i)} \frac{1}{\sqrt{|\mathcal{N}(i)| \cdot |\mathcal{N}(u)|}} \boldsymbol{v}_u^{(l-1)}. \quad (1)$$

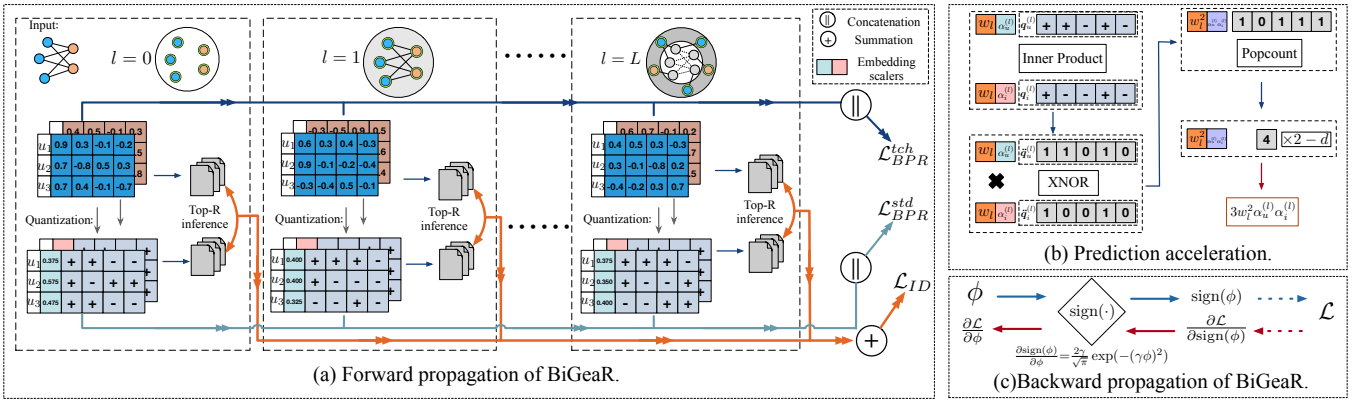


Figure 2: BiGeaR first pre-trains the full-precision embeddings and then triggers the (1) graph layer-wise quantization, (2) inference distillation, and (3) accurate gradient estimation to learn the binarized representations (Best view in color).

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., $\boldsymbol{v}_u^{(l)}$) and perform the layer-wise 1-bit quantization in parallel as:

$$\boldsymbol{q}_u^{(l)} = \text{sign}\left(\boldsymbol{v}_u^{(l)}\right), \quad \boldsymbol{q}_i^{(l)} = \text{sign}\left(\boldsymbol{v}_i^{(l)}\right), \tag{2}$$

where embedding segments $\boldsymbol{q}_u^{(l)}, \boldsymbol{q}_i^{(l)} \in \{-1,1\}^d$ retain the node latent features directly from $\boldsymbol{v}_u^{(l)}$ and $\boldsymbol{v}_i^{(l)}$. To equip with the layer-wise quantized embeddings, we further include a layer-wise positive embedding scaler for each node (e.g., $\alpha_u^{(l)} \in \mathbb{R}^+$), such that $\boldsymbol{v}_u^{(l)} \doteq \alpha_u^{(l)} \boldsymbol{q}_u^{(l)}$. Then for each entry in $\alpha_u^{(l)} \boldsymbol{q}_u^{(l)}$, it is still binarized by $\{-\alpha_u^{(l)}, \alpha_u^{(l)}\}$. In this work, we compute the mean of L1-norm as:

$$\alpha_u^{(l)} = \frac{1}{d} \cdot ||\boldsymbol{v}_u^{(l)}||_1, \quad \alpha_i^{(l)} = \frac{1}{d} \cdot ||\boldsymbol{v}_i^{(l)}||_1. \tag{3}$$

Instead of setting $\alpha_u^{(l)}$ and $\alpha_i^{(l)}$ 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:

$$\mathcal{A}_x = \{\alpha_x^{(0)}, \alpha_x^{(1)}, \cdots, \alpha_x^{(L)}\}, \quad \mathcal{Q}_x = \{\boldsymbol{q}_x^{(0)}, \boldsymbol{q}_x^{(1)}, \cdots, \boldsymbol{q}_x^{(L)}\}. \tag{4}$$

From the technical perspective, BiGeaR binarizes the intermediate semantics at different layers of the receptive field [52, 58] 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:

$$\widehat{y}_{u,i} = \left\langle f(\mathcal{A}_u, \mathcal{Q}_u), f(\mathcal{A}_i, \mathcal{Q}_i) \right\rangle, \tag{5}$$

where function $f(\cdot,\cdot)$ in this work is implemented as:

$$f(\mathcal{A}_u, \mathcal{Q}_u) = \Big\|_{l=0}^{L} w_l \alpha_u^{(l)} \boldsymbol{q}_u^{(l)}, \quad f(\mathcal{A}_i, \mathcal{Q}_i) = \Big\|_{l=0}^{L} w_l \alpha_i^{(l)} \boldsymbol{q}_i^{(l)}. \tag{6}$$

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 [52]). In this work, we set $w_l \propto l$ 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., $w_l \alpha_u^{(l)} \boldsymbol{q}_u^{(l)}$, entries are binarized by two values (i.e., $-w_l \alpha_u^{(l)}$ or $w_l \alpha_u^{(l)}$). Thus, we can achieve the prediction acceleration by decomposing Equation (5) with *bitwise operations*. Concretely, in practice, $\boldsymbol{q}_u^{(l)}$ and $\boldsymbol{q}_i^{(l)}$ will be firstly encoded into basic d-bits binary codes, denoted by $\ddot{\boldsymbol{q}}_u^{(l)}, \ddot{\boldsymbol{q}}_i^{(l)} \in \{0,1\}^d$. Then we replace Equation (5) by introducing the following formula:

$$\widehat{y}_{u,i} = \sum_{l=0}^{L} w_l^2 \alpha_u^{(l)} \alpha_i^{(l)} \cdot \left(2\texttt{Popcount}\left(\texttt{XNOR}(\ddot{\boldsymbol{q}}_u^{(l)}, \ddot{\boldsymbol{q}}_i^{(l)})\right) - d\right). \tag{7}$$

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 `XNOR`. 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., $\boldsymbol{v}_u^{(l)}$) 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 $\widehat{y}_{u,i}^{tch}$ and $\widehat{y}_{u,i}^{std}$. For Top-K recommendation, then our target is to reduce their discrepancy as:

$$\text{argmin } \mathcal{D}(\widehat{y}_{u,i}^{tch}, \widehat{y}_{u,i}^{std}). \tag{8}$$

A straightforward implementation of $\mathcal{D}$ 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, 59]), they may not be appropriate for Top-K recommendation, because:

- Firstly, both KLD and MSE in $\mathcal{D}$ encourage the student logits (e.g., $\widehat{y}_{u,i}^{std}$) 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, they both 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, 50], 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 σ represent the activation function (e.g., Sigmoid). We first pre-train the **teacher** embeddings to minimize *Bayesian Personalized Ranking* (BPR) loss [45]:

$$\mathcal{L}_{BPR}^{tch} = -\sum_{u\in\mathcal{U}}\sum_{\substack{i\in\mathcal{N}(u)\\ j\notin\mathcal{N}(u)}} \ln\sigma(\widehat{y}_{u,i}^{tch} - \widehat{y}_{u,j}^{tch}), \tag{9}$$

where $\mathcal{L}_{BPR}^{tch}$ forces the prediction of an observed interaction to be higher than its unobserved counterparts, and the teacher score $\widehat{y}_{u,i}^{tch}$ is computed as $\widehat{y}_{u,i}^{tch} = \langle \|_{l=0}^{L} w_l \boldsymbol{v}_u^{(l)}, \|_{l=0}^{L} w_l \boldsymbol{v}_i^{(l)} \rangle$. 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 $\mathcal{I}$:

$$\widehat{\boldsymbol{y}}_u^{tch,(l)} = \langle w_l \widehat{\boldsymbol{v}}_u^{(l)}, w_l \widehat{\boldsymbol{v}}_i^{(l)} \rangle_{i\in\mathcal{I}}. \tag{10}$$

Based on the segment scores $\widehat{\boldsymbol{y}}_u^{tch,(l)}$ at the l-th layer, we first sort out Top-R items with the highest matching scores, denoted by $S_{tch}^{(l)}(u)$. And hyper-parameter $R \ll |\mathcal{I}|$. Inspired by [50], then we propose our layer-wise inference distillation as follows:

$$\mathcal{L}_{ID}(u) = \sum_{l=0}^{L} \mathcal{L}_{ID}^{(l)}(\widehat{\boldsymbol{y}}_u^{std,(l)}, S_{tch}^{(l)}(u)) = -\frac{1}{R}\sum_{l=0}^{L}\sum_{k=1}^{R} w_k \cdot \ln\sigma(\widehat{y}_{u,S_{tch}^{(l)}(u,k)}^{std,(l)}), \tag{11}$$

where student scores $\widehat{\boldsymbol{y}}_u^{std,(l)}$ is computed similarly to Equation (10) and $S_{tch}^{(l)}(u,k)$ returns the k-th high-scored item from the pseudo-positive samples. w_k is the ranking-aware weight presenting two major effects: (1) since samples in $S_{tch}^{(l)}(u)$ are not necessarily all ground-truth positive, w_k thus balances their contribution to the overall loss; (2) it dynamically adjusts the weight importance for different ranking positions in $S_{tch}^{(l)}(u)$. To achieve these, w_k can be developed by following the parameterized geometric distribution for approximating the tailed item popularity [44]:

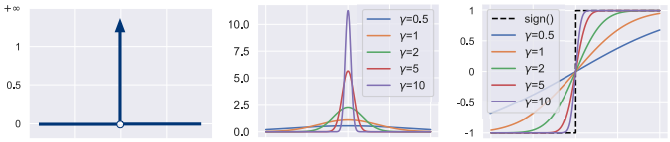
$$w_k = \lambda_1 \exp(-\lambda_2 \cdot k), \tag{12}$$

where λ_1 and λ_2 control the loss contribution level and sharpness of the distribution. Intuitively, $\mathcal{L}_{ID}$ encourages the 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 $\mathcal{L}_{BPR}^{std}$ that calculates BPR loss (similar to Equation (9)) with the student predictions from Equation (5) and $\mathcal{L}_{ID}$ for all training samples, our final objective function for learning embedding binarization is defined as:

$$\mathcal{L} = \mathcal{L}_{BPR}^{std} + \mathcal{L}_{ID} + \lambda||\Theta||_2^2, \tag{13}$$

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



(a) Dirac delta function. (b) Gradient estimation. (c) Cumulative distribution.

Figure 3: Gradient estimation.

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 sign(·) 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(·) by $\text{sign}(\phi) = 2u(\phi) - 1$, and thus theoretically $\frac{\partial \text{sign}(\phi)}{\partial\phi} = 2\frac{\partial u(\phi)}{\partial\phi}$. 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 $\delta(\cdot)$ [6] 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:

$$\delta(\phi) = \lim_{\beta\to\infty} \frac{|\beta|}{\sqrt{\pi}} \exp(-(\beta\phi)^2), \tag{14}$$

which imlies that:

$$\frac{\partial \text{sign}(\phi)}{\partial\phi} \doteq \frac{2\gamma}{\sqrt{\pi}} \exp(-(\gamma\phi)^2). \tag{15}$$

As shown in Figure 3(b)-(c), hyper-parameter γ determines the sharpness of the derivative curve for approximation to sign(·).

Intuitively, our proposed gradient estimator follows the main direction of factual gradients with sign(·) 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, 42, 61].

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 [60], we measure how the latent feature of a distant node x finally affects u's representation segments before binarization (e.g., $\boldsymbol{v}_u^{(l)}$), supposing x is a multi-hop neighbor of u. We denote the **feature enrichment ratio** $\mathbb{E}_{x\to u}^{(l)}$ as the L1-norm of Jacobian matrix $\left[\partial\boldsymbol{v}_u^{(l)} / \partial\boldsymbol{v}_{x_u}^{(0)}\right]$, by detecting the absolute influence of all fluctuation in entries of $\boldsymbol{v}_x^{(0)}$ to $\boldsymbol{v}_u^{(l)}$, i.e., $\mathbb{E}_{x\to u}^{(l)} = \left\|\left[\partial\boldsymbol{v}_u^{(l)} / \partial\boldsymbol{v}_x^{(0)}\right]\right\|_1$. Focusing on a l-length path h connected by the node sequence: $x_h^l, x_h^{l-1}, \cdots, x_h^1, x_h^0$, where $x_h^l = u$ and $x_h^0 = x$, we

follow the chain rule to develop the derivative decomposition as:

$$\frac{\partial \boldsymbol{v}_u^{(l)}}{\partial \boldsymbol{v}_x^{(0)}} = \sum_{h=1}^{H} \left[\frac{\partial \boldsymbol{v}_{x_h^l}^{(l)}}{\partial \boldsymbol{v}_{x_h^0}^{(0)}} \right]_h = \sum_{h=1}^{H} \prod_{k=l}^{1} \frac{1}{\sqrt{|\mathcal{N}(x_h^k)|}} \cdot \frac{1}{\sqrt{|\mathcal{N}(x_h^{k-1})|}} \cdot \boldsymbol{I} \\ = \sqrt{\frac{|\mathcal{N}(u)|}{|\mathcal{N}(x)|}} \sum_{h=1}^{H} \prod_{k=1}^{l} \frac{1}{|\mathcal{N}(x_h^k)|} \cdot \boldsymbol{I}, \tag{16}$$

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

$$\mathbb{E}_{x \to u}^{(l)} = \left\| \left[\frac{\partial \boldsymbol{v}_u^{(l)}}{\partial \boldsymbol{v}_x^{(0)}} \right] \right\|_1 = d \cdot \sqrt{\frac{|\mathcal{N}(u)|}{|\mathcal{N}(x)|}} \cdot \sum_{h=1}^{H} \prod_{k=1}^{l} \frac{1}{|\mathcal{N}(x_h^k)|}. \tag{17}$$

Note that here the term $\sum_{h=1}^{H} \prod_{k=1}^{l} 1/|\mathcal{N}(x_h^k)|$ is exactly the probability of the l-length random walk starting at u that finally arrives at x, which can be interpreted as:

$$\mathbb{E}_{x \to u}^{(l)} \propto \frac{1}{\sqrt{|\mathcal{N}(x)|}} \cdot Prob(l\text{-step random walk from } u \text{ arrives at } x). \tag{18}$$

Magnification Effect of Feature Uniqueness. Equation (18) implies that, with the equal probability to visit adjacent neighbors, distant nodes with fewer degrees (i.e., $|\mathcal{N}(x)|$) 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$_{tch}$ and BiGeaR$_{std}$ to denote the pre-training version and binarized one, respectively. As we can observe from Table 1, (1) both BiGeaR$_{tch}$ 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 $\mathcal{L}_{BPR}$ computation, to prevent *over-smoothing* issue [33, 35], usually $L \leq 4$; compare to the convolution operation, the complexity of $\mathcal{L}_{BPR}$ is acceptable. (3) For $\mathcal{L}_{ID}$ preparation, after the training of BiGeaR$_{tch}$, 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 $O(N + R \ln R)$. For BiGeaR$_{std}$, it takes a layer-wise inference distillation from BiGeaR$_{tch}$ 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: Traing time complexity.

	LightGCN	BiGeaR$_{tch}$	BiGeaR$_{std}$
Graph Normalization	$O(2E)$	$O(2E)$	-
Conv. and Quant.	$O(\frac{2SdE^2L}{B})$	$O(\frac{2SdE^2L}{B})$	$O(2Sd(\frac{E^2L}{B} + (L+1)E))$
$\mathcal{L}_{BPR}$ Loss	$O(2SdE)$	$O(2Sd(L+1)E)$	$O(2Sd(L+1)E)$
$\mathcal{L}_{ID}$ Loss	-	$O(MNd(L+1)(N+R\ln R))$	$O(SRd(L+1)E)$
Gradient Estimation	-	-	$O(2Sd(L+1)E)$

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 $\frac{32d}{(L+1)(32+d)}$. Normally, $L \leq 4$ and $d \geq 64$, thus our model achieves at least 4× 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.

Table 2: Complexity of space cost and online prediction.

	Embedding size	#FLOP	#BOP
LightGCN	$O(32(M+N)d)$	$O(2MNd)$	-
BiGeaR	$O((M+N)(L+1)(32+d))$	$O(4MN(L+1))$	$O(2MN(L+1)d)$

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 experimented datasets (including the training/test splits) from: MovieLens[2] [9, 10, 24, 49], Gowalla[3] [21, 49, 55, 56], Pinterest[4] [15, 49], Yelp2018[5] [21, 55, 56], Amazon-Book[6] [21, 55, 56]. Dataset statistics and detailed descriptions are reported in Table 3 and Appendix C.

Table 3: The statistics of datasets.

	MovieLens	Gowalla	Pinterest	Yelp2018	Amazon-Book
#Users	6,040	29,858	55,186	31,668	52,643
#Items	3,952	40,981	9,916	38,048	91,599
#Interactions	1,000,209	1,027,370	1,463,556	1,561,406	2,984,108

Evaluation Metric. In the evaluation of Top-K recommendation, 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, 67], HashGNN [49]) and general model-based (LSH [16], HashNet [7], CIGAR [28]), and (2) full-precision models including neural-network-based (NeurCF [23]) and graph-based (NGCF [55], DGCF [56], LightGCN [21]). Detailed introduction of these methods are attached in Appendix D.

We exclude early quantization-based recommendation baselines, e.g., CH [39], DiscreteCF [65], DPR [66], and full-precision solutions,

[2] https://grouplens.org/datasets/movielens/1m/
[3] https://github.com/gusye1234/LightGCN-PyTorch/tree/master/data/gowalla
[4] https://sites.google.com/site/xueatalphabeta/dataset-1/pinterest_iccv
[5] https://github.com/gusye1234/LightGCN-PyTorch/tree/master/data/yelp2018
[6] https://github.com/gusye1234/LightGCN-PyTorch/tree/master/data/amazon-book

Table 4: Performance comparison (waveline and underline represent the best performing full-precision and quantization-based models).

Model	MovieLens (%)		Gowalla (%)		Pinterest (%)		Yelp2018 (%)		Amazon-Book (%)	
	Recall@20	NDCG@20	Recall@20	NDCG@20	Recall@20	NDCG@20	Recall@20	NDCG@20	Recall@20	NDCG@20
NeurCF	21.40 ± 1.51	37.91 ± 1.14	14.64 ± 1.75	23.17 ± 1.52	12.28 ± 1.88	13.41 ± 1.13	4.28 ± 0.71	7.24 ± 0.53	3.49 ± 0.75	6.71 ± 0.72
NGCF	24.69 ± 1.67	39.56 ± 1.26	16.22 ± 0.90	24.18 ± 1.23	14.67 ± 0.56	13.92 ± 0.44	5.89 ± 0.35	9.38 ± 0.52	3.65 ± 0.73	6.90 ± 0.65
DGCF	25.28 ± 0.39	45.98 ± 0.58	18.64 ± 0.30	25.20 ± 0.41	15.52 ± 0.42	16.51 ± 0.56	6.37 ± 0.55	11.08 ± 0.48	4.32 ± 0.34	7.73 ± 0.27
LightGCN	26.28 ± 0.20	46.04 ± 0.18	19.02 ± 0.19	25.71 ± 0.25	15.33 ± 0.28	16.29 ± 0.24	6.79 ± 0.31	12.17 ± 0.27	4.84 ± 0.09	8.11 ± 0.11
BiGeaR	**25.57** ± 0.33	**45.56** ± 0.31	**18.36** ± 0.14	**24.96** ± 0.17	**15.57** ± 0.22	**16.83** ± 0.46	**6.47** ± 0.14	**11.60** ± 0.18	**4.68** ± 0.11	**8.12** ± 0.12
Capability	97.30%	98.96%	96.53%	97.08%	100.32%	101.94%	95.29%	95.32%	96.69%	100.12%
LSH	11.38 ± 1.23	14.87 ± 0.76	8.14 ± 0.98	12.19 ± 0.86	7.88 ± 1.21	9.84 ± 0.90	2.91 ± 0.51	5.06 ± 0.67	2.41 ± 0.95	4.39 ± 1.16
HashNet	15.43 ± 1.73	24.78 ± 1.50	11.38 ± 1.25	16.50 ± 1.42	10.27 ± 1.48	11.64 ± 0.91	3.37 ± 0.78	7.31 ± 1.16	2.86 ± 1.51	4.75 ± 1.33
CIGAR	14.84 ± 1.44	24.63 ± 1.77	11.57 ± 1.01	16.77 ± 1.29	10.34 ± 0.97	11.87 ± 1.20	3.65 ± 0.90	7.87 ± 1.03	3.05 ± 1.32	4.98 ± 1.24
GumbelRec	16.62 ± 2.17	29.36 ± 2.53	12.26 ± 1.58	17.49 ± 1.08	10.53 ± 0.79	11.86 ± 0.86	3.85 ± 1.39	7.97 ± 1.59	2.69 ± 0.55	4.32 ± 0.47
HashGNN_h	14.21 ± 1.67	24.39 ± 1.87	11.63 ± 1.47	16.82 ± 1.35	10.15 ± 1.43	11.96 ± 1.58	3.77 ± 1.02	7.75 ± 1.39	3.09 ± 1.29	5.19 ± 1.03
HashGNN_s	19.87 ± 0.93	37.32 ± 0.81	13.45 ± 0.65	19.12 ± 0.68	12.38 ± 0.86	13.63 ± 0.75	4.86 ± 0.36	8.83 ± 0.27	3.34 ± 0.25	5.82 ± 0.24
BiGeaR	**25.57** ± 0.33	**45.56** ± 0.31	**18.36** ± 0.14	**24.96** ± 0.17	**15.57** ± 0.22	**16.83** ± 0.46	**6.47** ± 0.14	**11.60** ± 0.18	**4.68** ± 0.11	**8.12** ± 0.12
Gain	28.69%	22.08%	36.51%	30.54%	25.77%	23.48%	33.13%	31.37%	40.12%	39.52%
***p*-value**	5.57e-7	2.64e-8	2.21e-7	7.69e-8	2.5e-5	3.51e-5	3.27e-6	5.30e-8	3.49e-6	7.14e-8

e.g., GC-MC [5], PinSage [64], mainly because the above competing models [21, 23, 28, 55] have validated the superiority to them.

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 η is tuned within $\{10^{-4}, 10^{-3}, 10^{-2}\}$ and the coefficient of $L2$ normalization λ is tuned among $\{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\}$. 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%∼102% of performance 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.
- **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 *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 of BiGeaR over all five benchmarks are statistically significant.

4.3 Resource Consumption Analysis (RQ2)

We analyze the resource consumption in *training*, *online inference*, and *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_{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

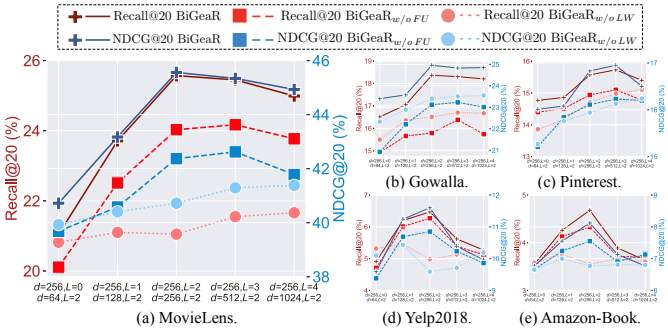


Figure 4: Study of graph layer-wise quantization.

the early GCN 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 pre-training 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_{infer}: we randomly generate 1,000 queries for online prediction and conduct experiments with the vanilla NumPy[7] on CPUs. We observe that HashGNN$_s$ takes a similar time cost with LightGCN. This is because, while HashGNN$_h$ purely binarizes the continuous embeddings, its relaxed version HashGNN$_s$ adopts a Bernoulli random variable to provide the probability of replacing the quantized digits with original real values [49]. Thus, although HashGNN$_h$ can use Hamming distance for prediction acceleration, HashGNN$_s$ with the improved recommendation accuracy can only be computed by floating-point arithmetics. As for BiGeaR, thanks to its bitwise-operation-supported capability, it runs about 8× faster than LightGCN whilst maintaining the similar performance on MovieLens dataset.

(3) S_{ET}: we only store the embedding tables that are necessary for online inference. As we just explain, HashGNN$_s$ interprets embeddings by randomly selected real values, which, however, leads to the expansion of space consumption. In contrast to HashGNN$_s$, BiGeaR can separately store the binarized embeddings and corresponding scalers, making a balanced trade-off between recommendation accuracy and space overhead.

Table 5: Resource consumption on MovieLens dataset.

	LightGCN	HashGNN$_h$	HashGNN$_s$	BiGeaR
T_{train}/#epcoch	4.91s	186.23s	204.53s	(5.16+6.22)s
T_{infer}/#query	32.54ms	2.45ms	31.76ms	3.94ms
S_{ET}	9.79MB	0.34MB	9.78MB	1.08MB
Recall@20	26.28%	14.21%	19.87%	25.57%

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 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 dimension 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, 49] as:
$$\boldsymbol{v}_x^{(l)} = \sum_{z \in \mathcal{N}(x)} \frac{1}{|\mathcal{N}(z)|} \boldsymbol{v}_z^{(l-1)}. \tag{19}$$
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 R@20	MovieLens N@20	Gowalla R@20	Gowalla N@20	Pinterest R@20	Pinterest N@20	Yelp2018 R@20	Yelp2018 N@20	Amazon-book R@20	Amazon-book N@20
noID	24.40	44.06	17.85	24.28	15.23	16.38	6.18	11.22	4.07	7.31
	-4.58%	*-3.29%*	*-2.78%*	*-2.72%*	*-2.18%*	*-2.85%*	*-4.48%*	*-3.28%*	*-13.03%*	*-9.98%*
endID	25.02	44.75	18.05	24.73	15.28	16.58	6.29	11.37	4.44	7.78
	-2.15%	*-1.78%*	*-1.69%*	*-0.92%*	*-1.86%*	*-1.49%*	*-2.78%*	*-1.98%*	*-5.13%*	*-4.19%*
KLD	24.32	44.38	17.63	24.07	14.78	15.92	5.83	10.36	4.13	7.21
	-4.89%	*-2.59%*	*-3.98%*	*-3.57%*	*-5.07%*	*-5.41%*	*-9.89%*	*-10.69%*	*-11.75%*	*-11.21%*
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., $\widehat{\boldsymbol{y}}_u^{tch,(l)}$ v.s. $\widehat{\boldsymbol{y}}_u^{std,(l)}$. We denote this variant as *KLD*. As we can observe from Table 6, using conventional knowledge distillation with KL divergence

[7] https://www.lfd.uci.edu/~gohlke/pythonlibs/

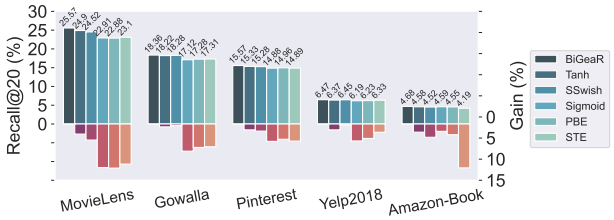


Figure 5: Gradient estimator comparison *w.r.t.* Recall@20.

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, 42], *SSwish* [12], *Sigmoid* [61], 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(·) = $2u(\cdot)$ - 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., $\frac{\#Interactions}{\#Users \cdot \#Items} = 0.0419 \gg \{0.00084, 0.00267, 0.0013, 0.00062\}$, 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, 63, 64]. Earlier CF methods, e.g., *Matrix Factorization* [32, 45], reconstruct historical interactions to learn user-item embeddings. Recent neural-network-based models, e.g., NeurCF [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 [64], and recent models, e.g., NGCF [55], DGCF [56], 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, 68], and categorical information [27]. For Top-K recommendation, models like DCF [65], DPR [66] 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, HashGNN [49] 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 [45]) 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 [53] 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 [46, 47, 62]. (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. Large scale distributed neural network training through online distillation. ICLR.

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

[3] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. 2018. Scalable methods for 8-bit training of neural networks. NeurIPS 31.

[4] Yoshua Bengio, Nicholas Léonard, 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.

[6] Ronald Newbold Bracewell and Ronald N Bracewell. 1986. The Fourier transform and its applications. Vol. 31999. McGraw-Hill New York.

[7] Zhangjie Cao, Mingsheng Long, Jianmin Wang, and Philip S Yu. 2017. Hashnet: Deep learning to hash by continuation. In ICCV. 5608–5617.

[8] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. 2017. Attentive collaborative filtering: Multimedia recommendation with item-and component-level attention. In SIGIR. 335–344.

[9] Yankai Chen, Menglin Yang, Yingxue Zhang, Mengchen Zhao, Ziqiao Meng, Jianye Hao, and Irwin King. 2022. Modeling Scale-free Graphs with Hyperbolic Geometry for Knowledge-aware Recommendation. WSDM.

[10] Yankai Chen, Yaming Yang, Yujing Wang, Jing Bai, Xiangchen Song, and Irwin King. 2022. Attentive Knowledge-aware Graph Convolutional Networks with Collaborative Guidance for Personalized Recommendation. ICDE.

[11] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In Recsys. 191–198.

[12] Sajad Darabi, Mouloud Belbahri, Matthieu Courbariaux, and Vahid Partovi Nia. 2018. Bnn+: Improved binary network training. arXiv.

[13] Venice Erin Liong, Jiwen Lu, Gang Wang, Pierre Moulin, and Jie Zhou. 2015. Deep hashing for compact binary codes learning. In CVPR. 2475–2483.

[14] Step function. 2022. https://en.wikipedia.org/wiki/Heaviside_step_function.

[15] Xue Geng, Hanwang Zhang, Jingwen Bian, and Tat-Seng Chua. 2015. Learning image and user features for recommendation in social networks. In ICCV.

[16] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In VLDB, Vol. 99. 518–529.

[17] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. 2019. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In ICCV. 4852–4861.

[18] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In NeurIPS. 1025–1035.

[19] Johan Håstad. 2001. Some optimal inapproximability results. Journal of the ACM (JACM) 48, 4, 798–859.

[20] Ruining He and Julian McAuley. 2016. Modeling the visual evolution of fashion trends with one-class collaborative filtering. In WWW. 507–517.

[21] 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. 639–648.

[22] 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, 2354–2366.

[23] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In WWW. 173–182.

[24] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast matrix factorization for recommendation with implicit feedback. In SIGIR.

[25] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.

[26] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical reparameterization with gumbel-softmax. In 5th ICLR.

[27] Wang-Cheng Kang, Derek Zhiyuan Cheng, Tiansheng Yao, Xinyang Yi, Ting Chen, Lichan Hong, and Ed H Chi. 2021. Learning to embed categorical features without embedding tables for recommendation. SIGKDD.

[28] Wang-Cheng Kang and Julian McAuley. 2019. Candidate generation with binary codes for large-scale top-n recommendation. In CIKM. 1523–1532.

[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 5th ICLR.

[31] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In ICML. PMLR, 1885–1894.

[32] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. Computer 42, 8, 30–37.

[33] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2019. Deepgcns: Can gcns go as deep as cnns?. In ICCV. 9267–9276.

[34] Hao Li, Wei Liu, and Heng Ji. 2014. Two-Stage Hashing for Fast Document Retrieval.. In ACL (2). 495–500.

[35] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In AAAI.

[36] Dawen Liang, Laurent Charlin, James McInerney, and David M Blei. 2016. Modeling user exposure in recommendation. In WWW. 951–961.

[37] Xiaofan Lin, Cong Zhao, and Wei Pan. 2017. Towards accurate binary convolutional neural network. In NeurIPS.

[38] Chunlei Liu, Wenrui Ding, Xin Xia, Yuan Hu, Baochang Zhang, Jianzhuang Liu, Bohan Zhuang, and Guodong Guo. 2019. RBCN: Rectified binary convolutional networks for enhancing the performance of 1-bit DCNNs. arXiv.

[39] Xianglong Liu, Junfeng He, Cheng Deng, and Bo Lang. 2014. Collaborative hashing. In CVPR. 2139–2146.

[40] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. 2017. The concrete distribution: A continuous relaxation of discrete random variables. In 5th ICLR.

[41] Yoon-Joo Park and Alexander Tuzhilin. 2008. The long tail of recommender systems and how to leverage it. In RecSys. 11–18.

[42] Haotong Qin, Ruihao Gong, Xianglong Liu, Mingzhu Shen, Ziran Wei, Fengwei Yu, and Jingkuan Song. 2020. Forward and backward information retention for accurate binary neural networks. In CVPR. 2250–2259.

[43] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In ECCV. Springer, 525–542.

[44] Steffen Rendle and Christoph Freudenthaler. 2014. Improving pairwise learning for item recommendation from implicit feedback. In WSDM. 273–282.

[45] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian personalized ranking from implicit feedback. arXiv.

[46] Zixing Song, Ziqiao Meng, Yifei Zhang, and Irwin King. 2021. Semi-supervised Multi-label Learning for Graph-structured Data. In CIKM. ACM, 1723–1733.

[47] Zixing Song, Xiangli Yang, Zenglin Xu, and Irwin King. 2022. Graph-based semi-supervised learning: A comprehensive review. TNNLS.

[48] Shyam A Tailor, Javier Fernandez-Marques, and Nicholas D Lane. 2021. Degree-quant: Quantization-aware training for graph neural networks. 9th ICLR.

[49] Qiaoyu Tan, Ninghao Liu, Xing Zhao, Hongxia Yang, Jingren Zhou, and Xia Hu. 2020. Learning to hash with GNNs for recommender systems. In WWW. 1988–1998.

[50] Jiaxi Tang and Ke Wang. 2018. Learning compact ranking models with high performance for recommender system. In SIGKDD. 2289–2298.

[51] Amazon user statistics. 2022. https://backlinko.com/amazon-prime-users.

[52] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. ICLR.

[53] Junfu Wang, Yunhong Wang, Zhen Yang, Liang Yang, and Yuanfang Guo. 2021. Bi-gcn: Binary graph convolutional network. In CVPR. 1561–1570.

[54] Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. 2017. A survey on learning to hash. TPAMI 40, 4, 769–790.

[55] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In SIGIR. 165–174.

[56] Xiang Wang, Hongye Jin, An Zhang, Xiangnan He, Tong Xu, and Tat-Seng Chua. 2020. Disentangled graph collaborative filtering. In SIGIR. 1001–1010.

[57] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In ICML. PMLR.

[58] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. IEEE TNNLS 32, 1, 4–24.

[59] Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V Le. 2020. Self-training with noisy student improves imagenet classification. In CVPR. 10687–10698.

[60] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. In ICML. PMLR, 5453–5462.

[61] Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xian-sheng Hua. 2019. Quantization networks. In CVPR. 7308–7316.

[62] Menglin Yang, Min Zhou, Marcus Kalander, Zengfeng Huang, and Irwin King. 2021. Discrete-time Temporal Network Embedding via Implicit Hierarchical Learning in Hyperbolic Space. In SIGKDD. 1975–1985.

[63] Menglin Yang, Min Zhou, Jiahong Liu, Defu Lian, and Irwin King. 2022. HRCF: Enhancing collaborative filtering via hyperbolic geometric regularization. In WebConf. 2462–2471.

[64] 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 SIGKDD. 974–983.

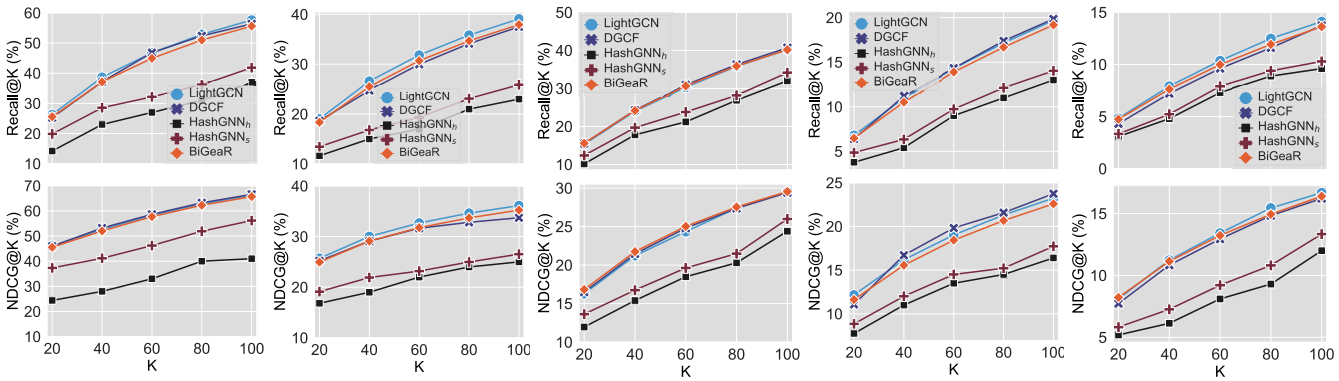
[65] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. 2016. Discrete collaborative filtering. In SIGIR. 325–334.

[66] Yan Zhang, Defu Lian, and Guowu Yang. 2017. Discrete personalized ranking for fast collaborative filtering from implicit feedback. In AAAI, Vol. 31.

[67] Yifei Zhang and Hao Zhu. 2019. Doc2hash: Learning discrete latent variables for documents retrieval. In ACL. 2235–2240.

[68] Yifei Zhang and Hao Zhu. 2020. Discrete Wasserstein Autoencoders for Document Retrieval. In ICASSP. IEEE, 8159–8163.

[69] Han Zhu, Mingsheng Long, Jianmin Wang, and Yue Cao. 2016. Deep hashing network for efficient similarity retrieval. In AAAI, Vol. 30.



(a) MovieLens. (b) Gowalla. (c) Pinterest. (d) Yelp2018. (e) Amazon-Book.

Figure 1: Top-K recommendation curve.

Table 1: Notations and meanings.

Notation	Explanation
d, L	Embedding dimensions and graph convolution layers.
$\mathcal{U}, \mathcal{I}$	Collection of users and items.
$\mathcal{N}(x)$	Neighbors of node x.
$\boldsymbol{v}_x^{(l)}$	Full-precision embedding of node x at l-th convolution.
$\boldsymbol{q}_x^{(l)}$	Binarized embedding of of node x at l-th quantization.
$\alpha_x^{(l)}$	l-th embedding scaler of node x.
$\mathcal{A}_x$ and $\mathcal{Q}_x$	Binarized embedding table of x learned by BiGeaR.
w_l	l-th weight in predicting matching score.
$y_{u,i}$	A scalar indicates the existence of user-item interaction.
$\widehat{y}_{u,i}^{tch}$	Predicted score based on full-precision embeddings.
$\widehat{y}_{u,i}^{std}$	Predicted score based on binarized embeddings.
$\widehat{\boldsymbol{y}}_u^{tch,(l)}$	Predicted scores of u based on l-th embeddings segments.
$\widehat{\boldsymbol{y}}_u^{std,(l)}$	Predicted scores of u based on l-th quantized segments.
$S_{tch}^{(l)}(u)$	pseudo-positive training samples of u.
w_k	k-th weight in inference distillation loss.
$\mathcal{L}_{BPR}^{tch}, \mathcal{L}_{BPR}^{std}$	BPR loss based on full-precision and binarized scores.
$\mathcal{L}_{ID}$	Inference distillation loss.
$\mathcal{L}$	Objective function of BiGeaR.
$u(\cdot), \delta(\cdot)$	Unit-step function and Dirac delta function.
$\lambda, \lambda_1, \lambda_2, \gamma, \eta$	Hyper-parameters and the learning rate.

Algorithm 1: BiGeaR algorithm.

Input: Interaction graph; trainable embeddings $\boldsymbol{v}_{\{\cdots\}}$; hyper-parameters: L, $\eta, \lambda, \lambda_1, \lambda_2, \gamma$.
Output: Prediction function $\mathcal{F}(u, i)$

$\mathcal{A}_u \leftarrow \emptyset, \mathcal{A}_i \leftarrow \emptyset, \mathcal{Q}_u \leftarrow \emptyset, \mathcal{Q}_i \leftarrow \emptyset$;
while BiGeaR not converge **do**
  for $l = 1, \cdots, L$ **do**
    $\boldsymbol{v}_u^{(l)} \leftarrow \sum_{i \in \mathcal{N}(u)} \frac{1}{\sqrt{|\mathcal{N}(u)| \cdot |\mathcal{N}(i)|}} \boldsymbol{v}_i^{(l-1)}$,
    $\boldsymbol{v}_i^{(l)} \leftarrow \sum_{u \in \mathcal{N}(i)} \frac{1}{\sqrt{|\mathcal{N}(i)| \cdot |\mathcal{N}(u)|}} \boldsymbol{v}_u^{(l-1)}$.
    if with inference distillation **then**
      $\boldsymbol{q}_u^{(l)} \leftarrow \text{sign}(\boldsymbol{v}_u^{(l)})$, $\boldsymbol{q}_i^{(l)} \leftarrow \text{sign}(\boldsymbol{v}_i^{(l)})$,
      $\alpha_u^{(l)} \leftarrow \frac{\|\boldsymbol{v}_u^{(l)}\|_1}{d}$, $\alpha_i^{(l)} \leftarrow \frac{\|\boldsymbol{v}_i^{(l)}\|_1}{d}$;
      Update $(\mathcal{A}_u, \mathcal{Q}_u), (\mathcal{A}_i, \mathcal{Q}_i)$ with $\alpha_u^{(l)} \boldsymbol{q}_u^{(l)}, \alpha_i^{(l)} \boldsymbol{q}_i^{(l)}$;
  $\widehat{y}_{u,i}^{tch} \leftarrow \left\langle \|_{l=0}^{L} w_l \boldsymbol{v}_u^{(l)}, \|_{l=0}^{L} w_l \boldsymbol{v}_i^{(l)} \right\rangle$.
  if with inference distillation **then**
    $\boldsymbol{q}_u^{(0)} \leftarrow \text{sign}(\boldsymbol{v}_u^{(0)})$, $\boldsymbol{q}_i^{(0)} \leftarrow \text{sign}(\boldsymbol{v}_i^{(0)})$,
    $\alpha_u^{(0)} \leftarrow \frac{\|\boldsymbol{v}_u^{(0)}\|_1}{d}$, $\alpha_i^{(0)} \leftarrow \frac{\|\boldsymbol{v}_i^{(0)}\|_1}{d}$;
    Update $(\mathcal{A}_u, \mathcal{Q}_u), (\mathcal{A}_i, \mathcal{Q}_i)$ with $\alpha_u^{(0)} \boldsymbol{q}_u^{(0)}, \alpha_i^{(0)} \boldsymbol{q}_i^{(0)}$;
    $\widehat{y}_{u,i}^{std} = \langle f(\mathcal{A}_u, \mathcal{Q}_u), f(\mathcal{A}_i, \mathcal{Q}_i) \rangle$;
    $\{\widehat{y}_{u,i}^{tch,(l)}\}_{l=0,1,\cdots,L} \leftarrow$ get score segments from $\widehat{y}_{u,i}^{tch}$;
    $\{\widehat{y}_{u,i}^{std,(l)}\}_{l=0,1,\cdots,L} \leftarrow$ get score segments from $\widehat{y}_{u,i}^{std}$;
    $\mathcal{L}_{ID} \leftarrow$ compute loss with $\{\widehat{y}_{u,i}^{tch,(l)}\}_{l=0,1,\cdots,L}, \{\widehat{y}_{u,i}^{std,(l)}\}_{l=0,1,\cdots,L}$.
    $\mathcal{L} \leftarrow$ compute $\mathcal{L}_{BPR}^{std}$ and $\mathcal{L}_{ID}$.
  else
    $\mathcal{L} \leftarrow$ compute $\mathcal{L}_{BPR}^{tch}$.
  Optimize BiGeaR with regularization;
return $\mathcal{F}$.

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, 49] is a widely adopted benchmark for movie recommendation. Similar to the setting in [9, 24, 49], $y_{u,i} = 1$ if user u has an explicit rating score towards item i, otherwise $y_{u,i} = 0$. In this paper, we use the MovieLens-1M data split.
- **Gowalla** [21, 49, 55, 56] 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, 49, 55, 56].
- **Pinterest** [15, 49] 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, 55, 56] 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, 55, 56].
- **Amazon-Book** [21, 55, 56] is organized from the book collection of Amazon-review for product recommendation [20]. Similarly to [21, 55, 56], 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 [49] 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 [49] 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, 67]. GumbelRec utilizes the Gumbel-softmax trick to replace sign$(\cdot)$ function for embedding binarization.
- **HashGNN** [49] is the state-of-the-art end-to-end 1-bit quantization recommender system. **HashGNN**$_h$ denotes its vanilla *hard encoding* version; and **HashGNN**$_s$ 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** [55] is a state-of-the-art graph-based collaborative filtering model that largely follows the standard GCN [30].
- **DGCF** [56] is one of the latest graph-based model 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^{-5}	1×10^{-4}	1×10^{-4}	1×10^{-6}
λ_1	1	1	1	1	1
λ_2	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 $\alpha^{(l)}$

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.

Table 3: Implementation of Embedding Scaler.

	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
LB	23.07	41.42	17.01	23.11	14.19	15.29	6.05	10.80	4.52	7.85
	-9.78%	*-9.09%*	*-7.35%*	*-7.41%*	*-8.86%*	*-9.15%*	*-6.49%*	*-6.90%*	*-3.42%*	*-3.33%*
BiGeaR	**25.57**	**45.56**	**18.36**	**24.96**	**15.57**	**16.83**	**6.47**	**11.60**	**4.68**	**8.12**

F.3 Implementation of w_l.

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

(1) $w_l = \frac{1}{L+1}$ equally contributes for all embedding segments.
(2) $w_l = \frac{1}{L+1-l}$ is positively correlated to the l value, so as to highlight higher-order structures of the interaction graph.
(3) $w_l = 2^{-(L+1-l)}$ 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., $w_l \propto l$, which however is simpler and effective with better recommendation accuracy.

Table 4: Implementation of w_l.

	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
(1)	22.75	41.13	16.15	21.82	14.16	15.48	5.88	10.32	4.46	7.63
(2)	25.07	44.64	17.81	24.46	15.26	16.57	6.40	11.38	4.58	7.96
(3)	21.23	37.81	15.24	20.71	12.93	14.28	5.24	9.51	3.74	64.98
Best	**25.57**	**45.56**	**18.36**	**24.96**	**15.57**	**16.83**	**6.47**	**11.60**	**4.68**	**8.12**

F.4 Implementation of w_k.

We further evaluate different w_k:

(1) $w_k = \frac{R-k}{R}$ is negatively correlated to the ranking position k.
(2) $w_k = \frac{1}{k}$ is inversely proportional to position k.
(3) $w_k = 2^{-k}$ 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 [44]. Moreover, Equation (12) introduces hyper-parameters to provide the flexibility of adjusting the function properties for different datasets.

Table 5: Implementation of w_k.

	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
(1)	24.97	44.33	17.96	24.87	15.11	16.20	6.28	11.21	4.43	7.78
(2)	25.08	45.19	17.95	24.95	15.18	16.34	6.27	11.25	4.48	7.92
(3)	25.16	44.92	18.32	24.81	15.26	16.65	6.33	11.36	4.53	8.06
Best	**25.57**	**45.56**	**18.36**	**24.96**	**15.57**	**16.83**	**6.47**	**11.60**	**4.68**	**8.12**