Quantize Sequential Recommenders Without Private Data

Lingfeng Shi, Yuang Liu, Jun Wang, Wei Zhang*
School of Computer Science and Technology, East China Normal University
{lingfengs111,frankliu624,wongjun,zhangwei.thu2011}@gmail.com

ABSTRACT

Deep neural networks have achieved great success in sequential recommendation systems. While maintaining high competence in user modeling and next-item recommendation, these models have long been plagued by the numerous parameters and computation, which inhibit them to be deployed on resource-constrained mobile devices. Model quantization, as one of the main paradigms for compression techniques, converts float parameters to low-bit values to reduce parameter redundancy and accelerate inference. To avoid drastic performance degradation, it usually requests a fine-tuning phase with an original dataset. However, the training set of user-item interactions is not always available due to transmission limits or privacy concerns. In this paper, we propose a novel framework to quantize sequential recommenders without access to any real private data. A generator is employed in the framework to synthesize fake sequence samples to feed the quantized sequential recommendation model and minimize the gap with a full-precision sequential recommendation model. The generator and the quantized model are optimized with a min-max game — alternating discrepancy estimation and knowledge transfer. Moreover, we devise a two-level discrepancy modeling strategy to transfer information between the quantized model and the full-precision model. The extensive experiments of various recommendation networks on three public datasets demonstrate the effectiveness of the proposed framework.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; *Personalization.*

KEYWORDS

Sequential Recommenders, Data-free Quantization, Model Compression

ACM Reference Format:
Lingfeng Shi, Yuang Liu, Jun Wang, Wei Zhang. 2023. Quantize Sequential Recommenders Without Private Data. In *Proceedings of the ACM Web Conference 2023 (WWW '23), April 30–May 04, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3543507.3583351

*Corresponding author. The work was supported in part by NSFC (No. 62072182 and 92270119), Shanghai Institute for AI Education, KLATASDS-MOE, and the special fund of short-term training and international conference for graduate students of East China Normal University.

WWW '23, April 30–May 04, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9416-1/23/04…$15.00
https://doi.org/10.1145/3543507.3583351

1 INTRODUCTION

Deep learning-based recommender systems have flourished for their powerful capacity in learning users' latent interests [46]. However, these large recommendation models inevitably require a lot of electrical and computing power to support their heavy computations, which will bring a lot of carbon emissions [11]. And their ever-growing complexities and spaces inhibit the deployment on resource-constrained devices. Moreover, deep recommendation models are fueled by a vast amount of user behavior data. When providing personalized services to users, these giant recommender systems even require immediate contextual data and side information for real-time inference, raising widespread concerns about personal privacy [15].

To tackle the problem of model redundancy, many methods have been applied [2, 34, 37, 41]. Model quantization, converting high-precision parameters to low-precision ones, becomes one of the main paradigms in model compression and acceleration [4, 40, 43]. It aims to store parameters with fewer bits so that the computation can be executed on integer-arithmetic units rather than on power-hungry floating-point ones [13]. However, one important challenge for quantization-based methods is the drastic reduction of model performance. In order to address this challenge, a series of quantization-aware training approaches have been proposed [28]. The common pipeline is first training a full-precision teacher model and then transferring the teacher's knowledge to a quantized student model using knowledge distillation (KD) [12].

Although these approaches have been proven effective in various recommendation scenarios, they always require full access to the training data of user behavior sequences. However, this kind of user data is not always available due to security concerns or transmission limits. In real situations, the user-item interactions in behavior sequences largely represent interests, tastes, even personalities, and are consequently important for personal privacy. Therefore, in the absence of user interactions and other contextual information (*e.g.*, reviews, pictures) to assist in user modeling [22], the effect of model quantization is still far from satisfactory.

Post-training quantization methods [6] therefore emerge to quantize weights in DNNs and embedding tables through correction strategies, without training on the original data. However, there is a non-negligible gap between the strategies and the goals of target tasks, causing the quantized models to suffer from performance degradation. This issue is even amplified in recommender systems with highly sparse user-item interactions. Another solution is directly using random noise as training sequences, but apparently random noise diverges from real user sequences in nature, especially when it comes to long-term user interests.

To cope with these issues, we propose an adversarial learning framework that provides Private sequential Recommenders Quantization (PRecQ), i.e. quantizing sequential recommenders without private data. Particularly, a generator is introduced in the

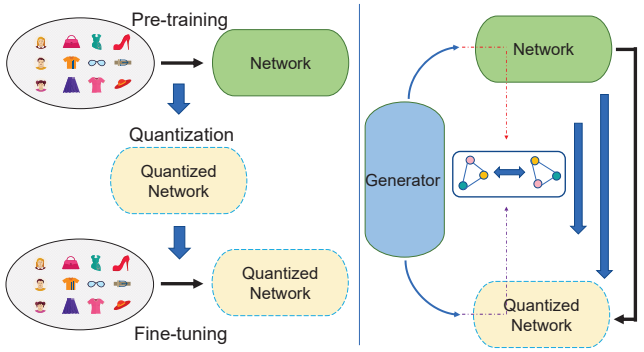


Figure 1: Overview of model quantization. Left: private data is available for model quantization. Right: private data is not accessible for model quantization.

framework to generate user interaction sequences. We ensure the validity and the authenticity of synthesized data, by adopting different sampling strategies and gradient backpropagation tactics. Moreover, we devise a two-level discrepancy modeling method for PRecQ to measure the gap between a quantized model and its corresponding full-precision model. This method fuses not only the discrepancy from models' output layers, but also a new embedding table discrepancy based on item similarity maps. Finally, the generator and quantized model are optimized together in an adversarial learning manner [5] to enable effective discrepancy estimation and knowledge transfer, as depicted in the right part of Figure 1. To sum up, our contributions are as follows:

- We propose a novel framework to effectively quantize sequential recommenders without access to private data. A generator is introduced to synthesize realistic and diverse user interaction sequences, marking the first attempt at data-free quantization in the field of recommender systems.
- A novel two-level discrepancy modeling strategy has been employed to measure the gap between a quantized model and its full-precision model, guiding the training of the quantized model and the generator at the same time.
- Extensive experiments of various recommendation networks on three public datasets have been conducted, demonstrating the superiority of our proposed framework PRecQ, in terms of the validity of data generation and the effectiveness of model quantization.

2 RELATED WORK

In this section, we review the relevant studies from sequential recommendation and model compression for recommendation.

2.1 Sequential Recommendation

Sequential recommendation requires handling user dynamic interests based on the user's historical interactions, which conforms to many practical recommendation scenarios and thus has been extensively studied in the past decade. Early works mainly focus on modeling transition patterns between consecutive items using Markov chains (MCs) [29, 31]. However, the long-range dependencies over the behavior sequences could not be well tackled in these methods. With the success of deep learning, deep sequential models have emerged as the mainstream approaches for sequential recommendation [38]. GRU4Rec [10] and its improved version [9] are the pioneering recurrent neural network-based models in this regard. To empower the capability of comprehensively modeling the correlations between different items in a user interaction sequence, attention-based mechanisms [36] are heavily utilized in sequential recommendation models, such as SASRec [19] and Bert4Rec [33].

Despite their performance improvements, the size and complexity of these models also increase significantly. This makes it difficult for them to be deployed on resource-constrained devices such as smart mobile phones. Although some existing studies (introduced in the next part) have addressed the model efficiency in recommender systems, few of them investigate this in the sequential recommendation scenario. In this paper, we concentrate on compressing deep sequential recommendation models.

2.2 Model Compression for Recommendation

Model compression for recommendation models dates back to the early hashing-based methods [45, 47]. They usually represent users and items using binary representations. Although they are very efficient in both space storage and inference computation, the model expressive capacity is very limited and inflexible, causing poor recommendation performance. Some recent studies investigate recommendation model compression from the aspect of pruning or dimension reduction [3, 7, 18, 24]. The techniques of automated machine learning and sparsity regularization are leveraged in these studies. However, the usage of high-precision real-valued embeddings constrains the memory compression ratios.

Model quantization, which uses low-bit numbers instead of high-precision floating-point values, could reduce a large recommendation model to a much smaller one. Some studies [1, 17, 32] in this regard adopt the product quantization technique [16]. It decomposes the full embedding table matrix into multiple codebooks and codewords. Nevertheless, the embeddings in the codebooks are still high-precision and the precomputation is hard to be realized for the sequential recommendation scenario. By contrast, general quantization techniques [21] could achieve both significant storage compression and high inference speed.

To our knowledge, there are only a very few studies w.r.t.model compression for sequential recommendation [7, 23, 41]. Although these works have shown some promising results, they always assume the existence of the training data for training the compressed models, which is usually not practical in real scenarios due to privacy concerns and transmission limits. In this paper, we focus on leveraging general quantization techniques to compress deep sequential recommendation models in a data-free manner [25], without accessing the training data.

3 PRELIMINARIES

3.1 Sequential Recommendation Task

In this paper, we apply quantized recommendation models to the session-based scenario for next-item recommendation [39]. Let $\mathcal{I} = \{i_1, i_2, i_3, \cdots, i_{|\mathcal{I}|}\}$ denote an item set and $T_u = [t_{u,1}, t_{u,2}, t_{u,3}, \cdots, t_{u,l}]$ denotes a behavior sequence from an anonymous user u, where

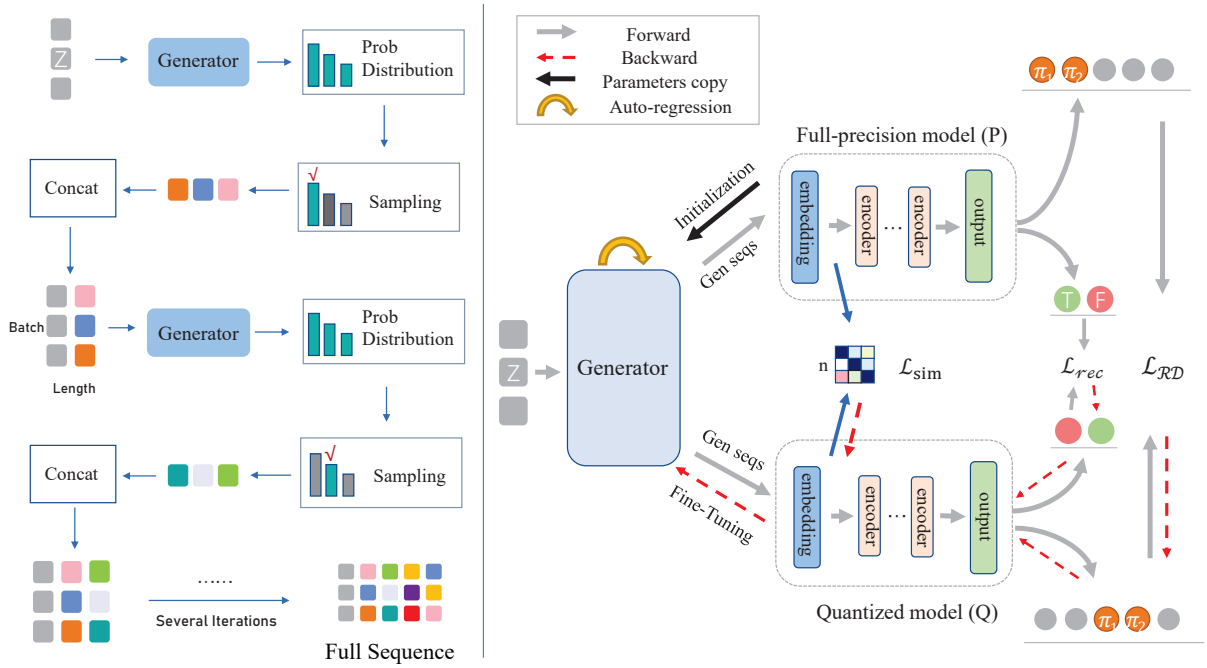


Figure 2: Overview of PRecQ. Left: flow of sequence generation. Right: procedure of discrepancy modeling.

l is the sequence length. Every behavior sequence is composed of interacted items (*i.e.*, $t_{u,j} \in \mathcal{I}$ where $1 \le j \le l$) in chronological order. The task of sequential recommendation is to predict the next item, namely t_{l+1}, for the current sequence. In this scenario, every item is first mapped into the embedding space through an embedding table, which has a huge size and is denoted as $\boldsymbol{W} \in \mathbb{R}^{|\mathcal{I}| \times d}$. Given $\mathcal{I}$ and T_u, the output of sequential recommendation model is a probability distribution $\boldsymbol{p} = [p_1\ p_2\ p_3\ ...\ p_{|\mathcal{I}|}]$, where p_i corresponds to item i $(1 \le i \le |I|)$. Normally, the top-N items with the largest probability values in $\boldsymbol{p}$ will be selected as recommendation results.

3.2 Model Quantization

We use symmetric linear quantization [43] as the quantization scheme for quantizing the weights in neural networks to integers with lower bits:

$$Quantize\,(x \mid S, M) := \text{Clamp}\,(\lfloor x \times S \rceil, -M, M)\,, \tag{1}$$

$$\text{Clamp}(x, a_1, a_2) = \min(\max(x, a_1), a_2)\,, \tag{2}$$

where x denotes a full-precision (float32) value and M is the highest value when performing b-bit quantization, which is expressed as follows:

$$M = 2^{b-1} - 1\,. \tag{3}$$

For example, when quantizing to 8 bits, $M = 127$. S is the quantization scaling factor for input x, which can be calculated based on statistics during training or on some calibration datasets for post-training. Here, the weight scaling factor is calculated by:

$$S_x = \frac{M}{max(|x_f|)}\,, \tag{4}$$

where x_f is any one of float32 numbers.

4 PROPOSED METHOD

4.1 Framework Overview

Figure 2 gives the overview of the proposed framework PRecQ. As described in the left part, taking initial noise z as input, the generator outputs a series of probability distributions. Given sequences produced so far, the next items in the sequences could be decided based on these distributions. Then they are concatenated with the sequences to make them become longer. The above process is repeated until the pre-set maximum length $\mathcal{T}$ is reached and consequently we get the full sequences. The right part contains pre-trained full-precision model P, quantized model Q, and generator G. The sequences produced by G are used to compute the discrepancy between P and its quantized model Q. The discrepancy function is composed of L_{rec} and L_{RD} in the output level, and items similarity loss L_{sim} in the intermediate level. As a result, the quantized model and the generator are optimized through a minimax game, where adversarial training is conducted. In what follows, we first introduce the rationale of sequence generation. Then we elaborate on PRecQ, including sampling strategies, discrepancy modeling, and knowledge transfer.

4.2 Sequence Generation

Since the training data is unavailable, the potential of the full-precision model P, as the only source of the original data, should be fully exploited. Inspired by autoregressive language models where generated sentences are similar to a 'real' data distribution, and the finding that sequential recommenders are often trained in an autoregressive manner [9, 19, 33], we directly use a copy of the full-precision model P as the initial Generator to synthesize user

behavior sequences autoregressively:

$$\begin{aligned} T_{u,l+1} &= [t_{u,1}, t_{u,2}, ..., t_{u,l}, t_{u,l+1}] \\ &= \textbf{Concat}(T_{u,l},\ \textbf{Model}(T_{u,l}))\,. \end{aligned} \tag{5}$$

To start the generation of one behavior sequence, a randomly sampled item is regarded as the first item in the sequence, *i.e.*, $t_{u,1}$. Afterwards, we feed it to a sequential recommendation model to have the probability distribution $\boldsymbol{p}$. And the generation of the next item depends on $\boldsymbol{p}$. This step can be repeated autoregressively to make the sequence longer until reaching the pre-set maximum length $\mathcal{T}$.

Once the generator is selected, the sampling strategy to determine the next item based on the distribution p becomes extremely important. This directly determines the quality and authenticity of the data generation. In what follows, we detail the strategies and the rationales behind them.

4.3 Sampling Strategies

4.3.1 Sampling strategy I. The simplest and most intuitive selection strategy is to choose the top-1 item with the maximum probability in the distribution $\boldsymbol{p}$ at each time step. This can be regarded as a greedy strategy. However, it easily leads to a high degree of similarity between the generated items in a sequence. A common paradigm of introducing randomness into this sampling strategy is the epsilon-greedy strategy:

$$\text{Next\_item\_index}(t) = \begin{cases} \arg\max_i(\boldsymbol{p}), & \text{with prob}(1-\varepsilon) \\ \text{any item}(i), & \text{with prob}(\varepsilon)\,, \end{cases} \tag{6}$$

where ε controls the magnitude of the introduced random variables. It represents purely a greedy algorithm strategy when ε tends to be 0, and obtains purely random sequences when ε tends to be 1.

For ease of representation and subsequent introduction, we often digitize the item index identification from an integer to a one-hot encoding, which is given by:

$$\bar{\boldsymbol{t}} = one\_hot\ (\text{Next\_item\_index}\,(t))\ . \tag{7}$$

The above sampling strategy has a certain effect. But one problem it encounters is that when choosing based on predicted probability values, only the item with the largest probability can be chosen, and when choosing based on random sampling, it can only be based on a discrete uniform distribution. Therefore, we introduce the Gumbel-Max trick [14] to fuse the above two choices: sampling according to the predicted probability distribution and introducing certain randomness at the same time.

4.3.2 Sampling strategy II. The Gumbel-Max trick provides a simple and efficient way to draw samples from a categorical distribution with class probability distribution $\boldsymbol{p}$:

$$\bar{\boldsymbol{t}} = one\_hot\left(\arg\max_i\left[g_i + \log p_i\right]\right), \tag{8}$$

where g_i belongs to i.i.d samples drawn from Gumbel(0,1)[1]. Then we use the softmax function as a continuous and differentiable approximation to argmax, which is named as Gumbel-Softmax distribution. Based on this, we generate k-dimensional sample vectors $\boldsymbol{y} = [y_1\ y_2\ y_3\ \cdots\ y_K] \in \mathbb{R}^K$ as follows:

$$y_i = \frac{\exp\left(\left(\log\left(p_i\right) + g_i\right)/\tau\right)}{\sum_{j=1}^{K}\exp\left(\left(\log\left(p_j\right) + g_j\right)/\tau\right)}, i \in \{1, \ldots, k\}\,. \tag{9}$$

where K corresponds to the K top-ranked items in the recommendation candidate list, meaning that we only consider the first K items given by the generator. Within this range, we apply the calculation of the Gumbel-Max trick, and the rest of the items are not considered. The rationale behind this choice is due to the commonly known deviation of the exposure of items in the recommendation system, the model may be more confident about the items with the highest prediction probabilities, compared to the items in the tail of the recommendation ranking list.

The Gumbel-Softmax distribution is smooth when $\tau > 0$, and therefore has a well-defined gradient that can be computed. As such, by replacing epsilon-greedy samples with Gumbel-Softmax samples we can further use backpropagation to compute gradients. This procedure of replacing non-differentiable sampling with a differentiable approximation during training is called Gumbel-Softmax estimator.

Based on the Gumbel-Softmax estimator, we can obtain the sample t as the output of G with parameters θ and random noise z. This is represented as $t = G(\theta, z)$. The path-wise gradients from quantized model Q to θ can therefore be computed without encountering any stochastic nodes as follows:

$$\frac{\partial}{\partial\theta}\mathbb{E}_{t\sim p_\theta}\left[\mathrm{Q}(t)\right] = \frac{\partial}{\partial\theta}\mathbb{E}_z\left[\mathrm{Q}(\mathrm{G}(\theta, z))\right] = \mathbb{E}_{z\sim p_z}\left[\frac{\partial \mathrm{Q}}{\partial \mathrm{G}}\frac{\partial \mathrm{G}}{\partial\theta}\right], \tag{10}$$

where p_θ refers to the distribution of G. In practice, we cut off the gradients between items in the same sequence, i.e. treating items in a single sequence as every independent prediction and computing their gradients respectively.

4.4 Two-level Discrepancy Modeling

As aforementioned, the framework PRecQ develops a novel two-level discrepancy modeling approach to characterize the discrepancy between full-precision and quantized models from the output level and the intermediate level.

4.4.1 Discrepancy modeling of output level. To effectively model the discrepancy between P and Q, we utilize the binary cross entropy (BCE) loss which is a commonly-used point-wise loss in sequential-based scenarios for next-item recommendation:

$$\mathcal{L}^{+}_{rec} = -\sum_{T\in\mathcal{G}}\sum_{j=1}^{\mathcal{T}}\left(r_{j,P}\cdot\log r_{j,Q} + \left(1-r_{j,P}\right)\cdot\log\left(1-r_{j,Q}\right)\right), \tag{11}$$

where $r_{j,P}$ and $r_{j,Q}$ are from the full-precision model P and the quantized model Q, respectively. They represent the results of the logits after sigmoid functions at the position j. It is worth mentioning that one of the two log terms above would be mathematically zero in conventional settings because the ground truth is either hard 1 or 0 in real user-item interactions. However, in our settings, $r_{t,P} \in (0, 1)$, so we keep both log terms. $\mathcal{G}$ represents the data set produced by the generator G. T is the generated training sequence.

[1]The Gumbel(0,1) distribution can be sampled by using inverse transform sampling, *i.e.*, drawing $u \sim$ Uniform(0, 1) and computing $g = -log(-log(u))$.

We view the sequences sampled from the generator as positive input sequences. In addition to these sequences, we also randomly sample some items which are aside from the positive ones. Then we have the following loss:

$$\mathcal{L}_{rec}^{-} = -\sum_{T \notin \mathcal{G}} \sum_{j=1}^{\mathcal{T}} \left(r_{j,P} \cdot \log r_{j,Q} + (1 - r_{j,P}) \cdot \log (1 - r_{j,Q}) \right). \quad (12)$$

To further exploit the output information of the full-precision model P, we hope that the quantized model Q not only fits the full-precision model's predictions of certain items, just as the BCE losses shown in Equation 11 and Equation 12, but also learns positional information from the top-K predicted items given by the full-precision model. Based on these considerations, we introduce a commonly-used ranking loss *RD* [35] in recommendation:

$$\begin{aligned} \mathcal{L}_{RD}(\pi_{1..K}, \hat{r}) &= -\sum_{m=1}^{K} \frac{1}{m} \cdot \log \left(\text{Prob} \left(rel = 1 \mid \hat{r}_{\pi_m} \right) \right) \\ &= -\sum_{m=1}^{K} \frac{1}{m} \cdot \log \left(\sigma \left(\hat{r}_{\pi_m} \right) \right), \end{aligned} \quad (13)$$

where $\pi_{1...K}$ is the top-K item list predicted by P, rather than the top-K list given by G as described in Section 4.3.2. σ is the sigmoid function and $\hat{r}_{\pi_m}$ represents the predicted score of the quantized model at the m-th position. We use the simple weight $1/m$ to give penalties inversely proportional to the rank. In practice, we calculate $\mathcal{L}_{RD}$ every s items in a single behavior sequence T, where $s \leq \mathcal{T}$.

4.4.2 Discrepancy modeling of intermediate level. A conventional manner to measure the discrepancy of intermediate layers relies on correlating feature maps, just as what knowledge distillation (KD) commonly does [30, 44]. However, the numerical spans of P and Q are very different because of precision settings. As a result, the gap between feature maps in the two models is relatively large. Considering the fact that the parameters of the embedding table occupy most of the parameters of the entire recommenders, we take into account the similarity between stored items in the embedding tables, i.e., the similarity between items should remain unchanged after quantization. Therefore, we introduce a loss to calculate the similarity between items in the embedding tables of the full-precision and quantized models as follows:

$$\mathcal{L}_{Sim} = \left\| \boldsymbol{R}_P(n) - \boldsymbol{R}_Q(n) \right\|_2^2, \quad (14)$$

where $\boldsymbol{R}_P$ and $\boldsymbol{R}_Q$ are the similarity matrices of the full-precision model and quantized model respectively. n is an item list randomly sampled from the whole item set. Here we use the L2 norm to calculate the distance between the two models. Particularly, we select the cosine similarity to compute the matrix $\boldsymbol{R}$:

$$\boldsymbol{R}_{ij} = \frac{< \boldsymbol{W}_{i'}, \boldsymbol{W}_{i''} >}{\|\boldsymbol{W}_{i'}\|_2 \cdot \|\boldsymbol{W}_{i''}\|_2}, \quad (15)$$

where $\boldsymbol{W}_{i'}$ and $\boldsymbol{W}_{i''}$ represent the embeddings of the two items i' and i'' in the embedding table, and $\boldsymbol{R}_{ij}$ represents their cosine similarity score.

Table 1: Dataset statistics.

Dataset	User	Item	Pair	Avglen	Density
Foursquare-TKY	2293	61857	573703	248.20	0.405%
Amazon Beauty	52204	57289	394908	5.63	0.013%
Steam	334730	13047	3686172	9.02	0.084%

4.5 Adversarial Knowledge Transfer

The proposed PRecQ framework trains the quantized model Q and the generator G in an adversarial minimax game, which contains the discrepancy estimation and knowledge transfer stages. In the knowledge transfer stage, the quantized model Q is optimized to minimize the two-level discrepancy to approximate the full-precision model P, which is defined as follows:

$$\mathcal{L}_{KT} = \mathcal{L}_{rec}^{+} + \alpha \mathcal{L}_{rec}^{-} + \beta \mathcal{L}_{RD} + \gamma \mathcal{L}_{Sim}, \quad (16)$$

where α, β, and γ are the hyperparameters to balance different loss terms.

In the discrepancy estimation stage, the generator G aims at maximizing the two-level discrepancy between Q and P to search for an effective discrepancy space, so as to generate more informative examples to guide the training of the quantized model Q. The loss is defined as follows:

$$\mathcal{L}_{DE} = -\mathcal{L}_{rec}^{+} - \alpha \mathcal{L}_{rec}^{-} - \beta \mathcal{L}_{RD}. \quad (17)$$

As a consequence, the knowledge is transferred from P to Q progressively without any real private data.

5 EXPERIMENTS

5.1 Experimental Setup

Datasets. We evaluate our approach on three public datasets:

- **Amazon Beauty** [27] is a dataset corresponding to the category 'Beauty', which is crawled from <Amazon.com> and contains abundant user profiles and behavior sequences.
- **Steam** [26] is a dataset collected from Steam, a large online video game distribution platform. It consists of 334,730 users, 13,047 games, and 3,686,172 English reviews from October 2010 to January 2018.
- **Foursquare-Tokyo** [42] consists of check-ins collected in Tokyo (marked as 'TKY') for about 10 months, which contains 2293 users, 61857 places, and 573,703 check-ins.

We follow the same preprocessing procedure as the previous tudies [8, 19]. Particularly, we treat the presence of a rating as a type of implicit feedback and discard users and items with fewer than 5 related interactions. For the Foursquare dataset, we only use the most recent 50 implicit feedbacks of each user. We follow the study [19] to have the data partition.

The data statistics are shown in Table 1. The 'Avglen' column refers to the average length of user interaction sequences. We can see that the Amazon Beauty dataset has the fewest interactions per user (on average) and is also the most sparse one. Steam has a larger average number of interactions per user and Foursquare-Tokyo is the densest dataset.

Sequential recommendation models We consider the following three deep sequential recommendation networks for validating the effectiveness of the proposed framework.

Table 2: Metric scores of the compared models on the three adopted datasets.

				Real data		Post quant		Noise		Ours	
Dataset	Model	Full	bits	Ndcg@10	Hit@10	Ndcg@10	Hit@10	Ndcg@10	Hit@10	Ndcg@10	Hit@10
Beauty	GRU4Rec	N:0.2252 H:0.3996	8	0.2200	0.3901	0.0434	0.0981	0.1990	0.3597	0.2168	0.3874
			4	0.2179	0.3870	0.0419	0.0933	0.1983	0.3555	0.2128	0.3830
	SASRec	N:0.2338 H:0.4104	8	0.2278	0.4096	0.0461	0.1017	0.2094	0.3793	0.2241	0.4038
			4	0.2272	0.3989	0.0430	0.0948	0.2074	0.3744	0.2176	0.3930
			2	0.2251	0.3933	0.0420	0.0936	0.2043	0.3684	0.2131	0.3851
Steam	SASRec	N:0.5186 H:0.7814	6	0.5027	0.7776	0.0501	0.1253	0.3807	0.7089	0.4122	0.7189
			4	0.4980	0.7654	0.0496	0.1071	0.3776	0.7027	0.4097	0.7127
	Bert4Rec	N:0.4927 H:0.7635	8	0.4850	0.7567	0.0500	0.1078	0.2298	0.4555	0.4179	0.6522
			4	0.4762	0.7498	0.0472	0.1023	0.2237	0.4455	0.3844	0.6458
			2	0.4689	0.7322	0.0425	0.0985	0.2098	0.4048	0.3324	0.6373
TKY	GRU4Rec	N:0.5550 H:0.7017	6	0.5056	0.6860	0.0447	0.0984	0.3820	0.6013	0.4301	0.6214
			4	0.4999	0.6772	0.0442	0.0975	0.3777	0.5935	0.4271	0.6151
	SASRec	H:0.6057 H:0.7348	8	0.6001	0.7331	0.1398	0.1901	0.4249	0.6171	0.4444	0.6441
			2	0.5893	0.7248	0.1307	0.1762	0.4092	0.5927	0.4284	0.6380
	Bert4Rec	N:0.6026 H:0.7401	4	0.3951	0.6270	0.1027	0.1453	0.3060	0.5237	0.3970	0.5742
			2	0.3875	0.6001	0.0962	0.1227	0.2963	0.5185	0.3846	0.5612

- **GRU4Rec [9]**: A seminal method that uses RNNs to model user behavior sequences for session-based recommendation. We treat the feedback sequence of each user as a session.
- **SASRec [19]**: SASRec consists of an embedding layer that includes both the item embedding and the positional embedding (timeline mask) of an input sequence, as well as a stack of one-directional transformer (TRM) layers. Each transformer layer contains a multi-head self-attention module and a position-wise feed-forward network.
- **Bert4Rec [33]** : Bert4Rec has an architecture similar to SASRec, but uses a bidirectional transformer and an auto-encoding (masked language modeling) task for training. It also includes item embedding and positional embedding.

Implementation details. We implement all networks and quantization methods using Pytorch. Regarding to the network architectures, we use two self-attention blocks with one head for SASRec and Bert4Rec. For GRU4Rec, we use one GRUCell only. We set the dimension to 128 for embedding layers and hidden layers of all models. For the time information, we directly use the timeline matrix to mask unrelated items, except for Bert4Rec, on which we use a positional embedding to add on item embeddings. Adam [20] is adopted as the optimizer and the learning rates of quantized models and generators are initialized to 1×10^{-4} and 1×10^{-5}, respectively. We use multi-step learning rates on most tasks. The dropout rate of turning-off neurons is set to 0.2 for Foursquare-TKY and 0.1 for the other two datasets due to their sparsity. The maximum sequence length of T_u for all users is set to 50 for Foursquare-TKY, 25 for Beauty, and 20 for Steam. More detailed implementations are illustrated in the following parts. To facilitate relevant studies, the code of the proposed PRecQ is available at https://github.com/Sinp17/PRecQ.

In our framework, the generator G and the full-precision model P have the same model structure, and the generator directly copies the parameters from P as initialization. As for the sampling strategy, we only sample the first top-K items, and these items contribute to the gradient back-propagation. K is chosen from {2000,5000,10000} or full item length. During training, we cut off the gradient between items in the same sequence. One choice of gradient computation is to fix the embedding table of the generator to maintain originality from the full-precision model's parameters. The detailed comparison can be found in the component ablation study later.

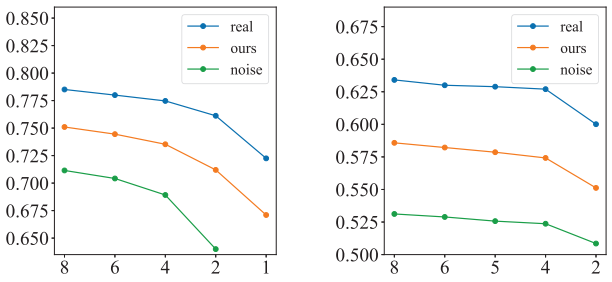
Evaluation metrics. To evaluate the attribute-based recommendation performance, we use two standard ranking metrics, i.e., Normalized Discounted Cumulative Gain at rank N (Ndcg@N) and Hit Rate (Hit@N). Specifically, we set Hit@10 and Ndcg@10 in most tests. Hit@5 and Ndcg@1 are added to the ablation study. All the reported results are averaged.

In the testing stage, to avoid heavy computation on all user-item pairs, we follow the strategy [19]. For each user u, we randomly sample 100 negative items and rank these items with the ground-truth item. Based on the rankings of these total 101 items, Hit@10 and Ndcg@10 can be evaluated.

5.2 Overview of Method Performance

In order to give a comprehensive view of the experimental results, we select two sequential recommendation models on the Amazon Beauty and Steam datasets and all the three models on Foursquare-TKY to conduct our experiments. For each model, we select several important bits for showing the results, which are shown in Table 2. 'Post quant' refers to the method of directly quantizing the teacher model without fine-tuning. And 'noise' means training on random item sequences, where items are sampled based on discrete uniform distribution. Note that the 'Full' column shows the performance in terms of Ndcg@10 and Hit@10 of the full-precision models.

To begin with, we can find that our method has improvements of about 1.5% ∼ 8% of Hit@10 and 4% ∼ 13% of Ndcg@10 on SASRec and GRU4Rec compared to the noise training. On average, we can



(a) GRU4Rec on steam (b) Bert4Rec on TKY

Figure 3: Performance change versus different quantization precisions in terms of the metric Hit@10.

achieve 92.8% of Hit@10 and 87.3% of Ndcg@10 of the training performance on the real data sets.

Furthermore, we find that the post-quantization method behaves poorly on the three datasets. This is probably because the process of quantization can severely disrupt the stored structure of the parameters in the embedding layer and other layers. Moreover, the effect of training with random sequences is surprisingly good. We speculate reasons accounting for this: (1) In the real sequence of user behavior, the short-term user sequence or the most recent interaction has a very large impact on the final decision, while the noise may be closer to the real data situation in terms of reconstructing the short-term sequences. (2) The effect of knowledge distillation is significant, and it can pass on the confidence of the full-precision model to the next click prediction to the quantized model, so the teacher model can give a reasonable explanation of most kinds of input sequences.

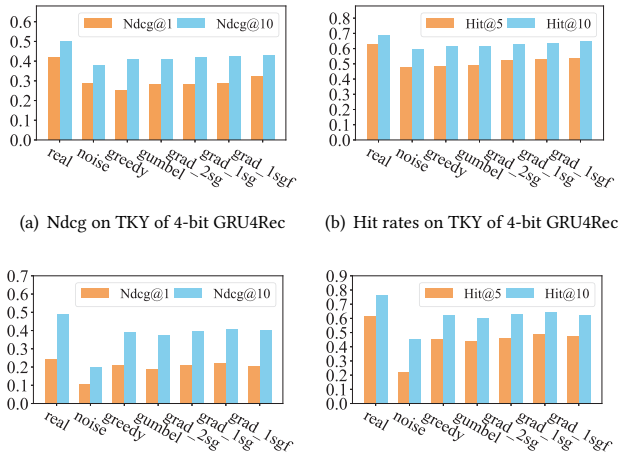
In addition, we also find that the results of Bert4Rec using our framework are much better than training on noise, compared with the results of SASRec and GRU4Rec. This is probably because, in practice, we choose to use the timeline matrix to mask items for SASRec and GRU4Rec, but for Bert4Rec, due to bidirectional modeling, we leverage a positional embedding and add it on token embeddings, and the representation of positional embedding can also be quantized. This makes Bert4Rec appear more sensitive to the authenticity of the sequence, resulting in a significant improvement of Bert4Rec with our method compared to training on noise.

5.3 Quantization Performance of Different Bits

To better illustrate how performance changes with different bits, we present two line charts, as shown in Figure 3. The curves of different quantization methods in the figures reflect that PRecQ is consistently better in the ultra-low precision situation. Particularly, in the range from 8 bits to 4 bits, shown in Figure 3(b), the descent effect of the three methods is not very obvious. However, when it comes to 2 bits or lower, shown in Figure 3(a), the decline is relatively significant. In this case, our method could achieve a relatively lower rate of decline compared to noise training.

5.4 Ablation Study

Component ablation. In our framework, the performance using different training data sources is shown in Figure 4. Apart from real sequences and noise sequences, the other five methods all introduce



(a) Ndcg on TKY of 4-bit GRU4Rec (b) Hit rates on TKY of 4-bit GRU4Rec

(c) Ndcg on Steam of 6-bit Bert4Rec (d) Hit rates on Steam of 6-bit Bert4Rec

Figure 4: Performance change with different data sources.

generators to produce training data. The difference is that these five methods employ different sampling strategies and gradient calculation schemes. All the generators copy the parameters of the corresponding full-precision model as initialization, while the quantized model is initialized randomly.

Particularly, 'greedy' refers to the epsilon-greedy strategy introduced in Section 4.3.1 with ϵ chosen from {0.1, 0.2, 0.3}. 'gumbel' is the Gumbel-Max trick in Section 4.3.2 with τ chosen from {1, 1/16, 1/64}. The latter three methods are with gradient computation and backpropagation based on the Gumbel-Max trick. Among them, considering that there might be a certain gap between the capacity of the generator and the quantized model when initialization [48], we devise a two-stage training method ('grad_2sg' in the figure), where we adopt the Gumbel-Softmax estimator without gradients (same as gumbel in the Figure) until the quantized model fits properly, then the generator and the quantized model are trained together based on a minimax game. While 'Grad_1sg' refers to the method where we directly train the generator and the quantized model together after initialization. 'Grad_1sgf' is similar to 'Grad_1sg', except for fixing the embedding layer of the generator after initialization, so as to maintain the originality of full-precision embedding parameters.

From Figure 4, we can see that the overall performance of methods with gradient backpropagation is better than the ones without gradients. For GRU4Rec on TKY, the methods with gradients are 1-2 points (percent) higher on Hit scores and about 1.5 points higher on Ndcg scores than those without gradients, while for Bert4Rec on Steam, there are 1-2 points higher on the Hit metric and about 1-3 points on the Ndcg metric. In the methods of not calculating the gradients, greedy and Gumbel strategies behave similarly on both metrics. In the methods of calculating gradients, one-stage training is more effective in generator training, and therefore the performance is better than two-stage training. The effect of fixed embedding depends on models and datasets.

Loss ablation. To ensure the effectiveness of knowledge transfer, we design multiple losses. For the loss L_{rec}, we treat the samples produced by the generator as positive samples, and samples that

Table 3: Ablation study of different losses.

Method	full-SASRec		4-bit GRU4Rec	
	N@10	H@10	N@10	H@10
Ours	0.4616	0.6497	0.4151	0.7353
w/o Sim	0.4587	0.6245	0.3966	0.7122
w/o RD	0.4273	0.6354	0.4132	0.7266
w/o Sim & RD	0.4191	0.6201	0.3870	0.7044

are in the itemset $\mathcal{I}$ but not in positive samples as negative samples. For the loss $\mathcal{L}_{Sim}$, we randomly select a list of items with the size of n from the item set and calculate the similarity matrix between them. In practice, we set n=2000 for TKY and Steam and n=1000 for Amazon Beauty. For the ranking loss $\mathcal{L}_{RD}$ in Equation 13, we set $\pi_{1...100}$ for all datasets, $s = 3$ for TKY and Steam, and $s = 5$ for Amazon Beauty. For hyperparameters in Equation 16 and 17, the default setting is $\alpha = 1, \beta = 1, \gamma = 1$ for GRU4Rec and $\alpha = 1, \beta = 0.25, \gamma = 1$ for SASRec and Bert4Rec. Small adjustments might be made depending on datasets.

To further investigate the effectiveness of different loss terms, we devise the ablation study shown in Table 3. We consider quantized models in two scenarios, full-precision SASRec on Foursquare TKY and 4-bit GRU4Rec on Steam, and choose two metrics to conduct the experiments. By comparing the scores in the first scenario of TKY dataset, we can see that ranking loss contributes to the Ndcg scores more, while the similarity loss helps Hit rate more. Whereas in the sparse dataset Steam, the benefit of the similarity loss is more significant than the ranking loss, making it an indispensable part of the proposed framework.

5.5 Case Study

5.5.1 Generated sequence visualization. To demonstrate the effectiveness of data generation, we conduct an experiment to calculate the item frequency ratios appearing in different data sources and sampling strategies. As illustrated in Figure 5, the x-axis represents the item frequency number in log form, whereas the y-axis represents its ratio in each data source. Particularly, we select 6000 user sequences and consider the frequency number from 1 to 150 for Foursquare-TKY. We can see that the purple line representing grad_1sg is closer to the real data set in terms of the frequency of spawning items. There are both sparse items and more densely distributed items in the real dataset. In contrast, the greedy method is more inclined to produce high-frequency items. The frequency of the noisy items differs the most from the real data situation.

5.5.2 Embedding visualization. Given those embedding parameters accounting for the vast majority of parameters in the sequential recommendation models, we choose to visualize the embedding table before and after quantization. The embedding table is chosen from 4bit-SASRec on Foursquare-TKY and we use Google's open-source visualization tool 'Embedding Projector'[2] for presentation. Figure 6 is visualized using Principal Component Analysis (PCA) based dimension reduction. The three dimensions with the greatest influence are selected as the xyz axes.

[2]http://projector.tensorflow.org/

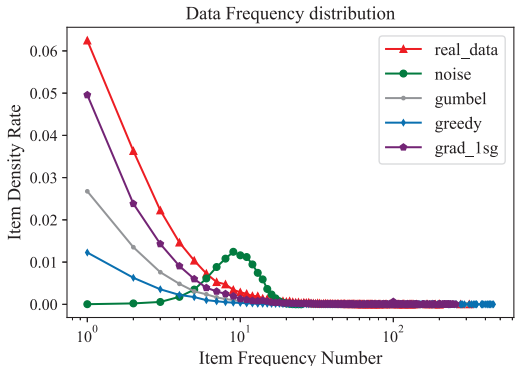
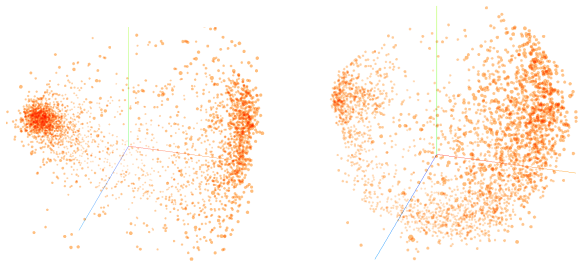


Figure 5: Data distributions for different sampling strategies.



(a) Full-precision embedding (b) Quantized embedding

Figure 6: Visualization of embedding space.

In Figure 6(a), we can see that full-precision embeddings points converge to the left and right parts of the figure. Considering the popularity properties of different items in real situations, high-exposure items should be learned better than low-exposure ones, leading to a bipolar trend in the embedding visualization figure. Affected by model quantization, the embedding image of the quantized model is different from that of the full-precision model, as is shown in Figure 6(b). It is more like a sphere in general. Whereas the points in the right figure also demonstrate a tendency to concentrate on the left and right poles. We attribute this to the high quality of our synthesized data, as is shown in Figure 5, and the effectiveness of knowledge distillation.

6 CONCLUSION

In this paper, we propose a novel framework to effectively quantize sequential recommenders without the requirement of original training data. Our contributions lie in employing a generator to synthesize high-quality user behavior sequences, by adopting different sampling strategies and gradient backpropagation tactics. Our framework is also favorable for the effectiveness of modeling the discrepancy between full-precision and quantized models from the intermediate embedding level and the output level. By jointly optimizing the generator and the quantized model, extensive experiments on various deep neural models demonstrate the superiority of PRecQ. In future work, we might consider applying the proposed method to mixed precision quantization.

REFERENCES

[1] Jin Chen, Defu Lian, Binbin Jin, Xu Huang, Kai Zheng, and Enhong Chen. 2022. Fast Variational AutoEncoder with Inverted Multi-Index for Collaborative Filtering. In *WWW*. 1944–1954.

[2] Tong Chen, Hongzhi Yin, Yujia Zheng, Zi Huang, Yang Wang, and Meng Wang. 2021. Learning elastic embeddings for customizing on-device recommenders. In *KDD*. 138–147.

[3] Wei Deng, Junwei Pan, Tian Zhou, Deguang Kong, Aaron Flores, and Guang Lin. 2021. DeepLight: Deep lightweight feature interactions for accelerating CTR predictions in ad serving. In *WSDM*. 922–930.

[4] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115* (2014).

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative adversarial networks. *Commun. ACM* 63, 11 (2020), 139–144.

[6] Hui Guan, Andrey Malevich, Jiyan Yang, Jongsoo Park, and Hector Yuen. 2019. Post-training 4-bit quantization on embedding tables. *arXiv preprint arXiv:1911.02079* (2019).

[7] Jialiang Han, Yun Ma, Qiaozhu Mei, and Xuanzhe Liu. 2021. DeepRec: On-device Deep Learning for Privacy-Preserving Sequential Recommendation in Mobile Commerce. In *WWW*. 900–911.

[8] Ruining He and Julian McAuley. 2016. Fusing similarity models with markov chains for sparse sequential recommendation. In *ICDM*. 191–200.

[9] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent neural networks with top-k gains for session-based recommendations. In *CIKM*. 843–852.

[10] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).

[11] Yassine Himeur, Abdullah Alsalemi, Ayman Al-Kababji, Faycal Bensaali, Abbes Amira, Christos Sardianos, George Dimitrakopoulos, and Iraklis Varlamis. 2021. A survey of recommender systems for energy efficiency in buildings: Principles, challenges and prospects. *Information Fusion* 72 (2021), 1–21.

[12] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* 2, 7 (2015).

[13] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*. 2704–2713.

[14] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* (2016).

[15] Arjan JP Jeckmans, Michael Beye, Zekeriya Erkin, Pieter Hartel, Reginald L Lagendijk, and Qiang Tang. 2013. Privacy in recommender systems. In *Social Media Retrieval*. Springer, 263–281.

[16] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.

[17] Gangwei Jiang, Hao Wang, Jin Chen, Haoyu Wang, Defu Lian, and Enhong Chen. 2021. xLightFM: Extremely Memory-Efficient Factorization Machine. In *SIGIR*. 337–346.

[18] Manas R Joglekar, Cong Li, Mei Chen, Taibai Xu, Xiaoming Wang, Jay K Adams, Pranav Khaitan, Jiahui Liu, and Quoc V Le. 2020. Neural input search for large scale recommendation models. In *KDD*. 2387–2397.

[19] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *ICDM*. IEEE, 197–206.

[20] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[21] Hao Li, Soham De, Zheng Xu, Christoph Studer, Hanan Samet, and Tom Goldstein. 2017. Training quantized nets: A deeper understanding. *Advances in Neural Information Processing Systems* 30 (2017).

[22] Jingjing Li, Mengmeng Jing, Ke Lu, Lei Zhu, Yang Yang, and Zi Huang. 2019. From zero-shot learning to cold-start recommendation. In *AAAI*, Vol. 33. 4189–4196.

[23] Yang Li, Tong Chen, Peng-Fei Zhang, and Hongzhi Yin. 2021. Lightweight self-attentive sequential recommendation. In *CIKM*. 967–977.

[24] Siyi Liu, Chen Gao, Yihong Chen, Depeng Jin, and Yong Li. 2021. Learnable embedding sizes for recommender systems. *arXiv preprint arXiv:2101.07577* (2021).

[25] Yuang Liu, Wei Zhang, Jun Wang, and Jianyong Wang. 2021. Data-free knowledge transfer: A survey. *arXiv preprint arXiv:2112.15278* (2021).

[26] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. 2015. Image-based recommendations on styles and substitutes. In *SIGIR*. 43–52.

[27] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *EMNLP-IJCNLP*. 188–197.

[28] Antonio Polino, Razvan Pascanu, and Dan Alistarh. 2018. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668* (2018).

[29] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *WWW*. 811–820.

[30] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2014. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550* (2014).

[31] Guy Shani, David Heckerman, Ronen I Brafman, and Craig Boutilier. 2005. An MDP-based recommender system. *JMLR* 6, 9 (2005).

[32] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. 2020. Compositional embeddings using complementary partitions for memory-efficient recommendation systems. In *KDD*. 165–175.

[33] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *CIKM*. 1441–1450.

[34] Yang Sun, Fajie Yuan, Min Yang, Guoao Wei, Zhou Zhao, and Duo Liu. 2020. A generic network compression framework for sequential recommender systems. In *SIGIR*. 1299–1308.

[35] Jiaxi Tang and Ke Wang. 2018. Ranking distillation: Learning compact ranking models with high performance for recommender system. In *KDD*. 2289–2298.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*, Vol. 30.

[37] Qinyong Wang, Hongzhi Yin, Tong Chen, Zi Huang, Hao Wang, Yanchang Zhao, and Nguyen Quoc Viet Hung. 2020. Next point-of-interest recommendation on resource-constrained mobile devices. In *WWW*. 906–916.

[38] Shoujin Wang, Longbing Cao, Yan Wang, Quan Z Sheng, Mehmet A Orgun, and Defu Lian. 2021. A survey on session-based recommender systems. *Comput. Surveys* 54, 7 (2021), 1–38.

[39] Wen Wang, Wei Zhang, Shukai Liu, Qi Liu, Bo Zhang, Leyu Lin, and Hongyuan Zha. 2020. Beyond Clicks: Modeling Multi-Relational Item Graph for Session-Based Target Behavior Prediction. In *WWW*. 3056–3062.

[40] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. 2016. Quantized convolutional neural networks for mobile devices. In *CVPR*. 4820–4828.

[41] Xin Xia, Hongzhi Yin, Junliang Yu, Qinyong Wang, Guandong Xu, and Quoc Viet Hung Nguyen. 2022. On-Device Next-Item Recommendation with Self-Supervised Knowledge Distillation. In *SIGIR*. 546–555.

[42] Dingqi Yang, Daqing Zhang, Vincent. W. Zheng, and Zhiyong Yu. 2015. Modeling User Activity Preference by Leveraging User Spatial Temporal Characteristics in LBSNs. *TSMC* 45, 1 (2015), 129–142.

[43] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. In *EMC2-NeurIPS*. IEEE, 36–39.

[44] Sergey Zagoruyko and Nikos Komodakis. 2016. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *arXiv preprint arXiv:1612.03928* (2016).

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

[46] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. *Comput. Surveys* 52, 1 (2019), 1–38.

[47] Yan Zhang, Hongzhi Yin, Zi Huang, Xingzhong Du, Guowu Yang, and Defu Lian. 2018. Discrete deep learning for fast content-aware recommendation. In *WSDM*. 717–726.

[48] Yichen Zhu and Yi Wang. 2021. Student customized knowledge distillation: Bridging the gap between student and teacher. In *ICCV*. 5057–5066.

Table 6: Quantization settings comparison.

Method	Real	Post quant	Noise	ours
log	0.475/0.767	0.043/0.094	0.372/0.703	0.411/0.729
min-max	0.479/0.775	0.043/0.096	0.389/0.711	0.415/0.735

APPENDIX

A RUNNING COST ANALYSIS

The linear quantization method used in PRecQ could achieve a 4 to 16 compression rate of model size or memory bandwidth, and speed up the sequential recommendation model 1x to 4x depending on hardware. We test the inference latency of 8-bit-GRU4Rec on the Beauty and TKY datasets using Intel CPU E5-2680 and the results are shown in Table 4.

Table 4: Running cost comparison.

Dataset	mode	Size	Latency
Beauty	FP32	59.28MB	9.32ms
	Qint8	14.82MB	5.49ms
TKY	FP32	64MB	21.13ms
	Qint8	16MB	10.62ms

B COMPRESSION METHOD COMPARISION

Different compression techniques vary in nature, making it hard to be compared. Here we take the tensor-train decomposition (TTD) [41] method as an example. Given SASRec tested on the Beauty dataset, the original embedding size is (57289×128) and could be compressed to two TT-cores of $(59 \times 971) + (16 \times 8)$, with TT-rank of 60. As shown in Table 5, though having a better compression rate, the performance of the TTD method is slightly worse than the linear quantization method of the 4-bit model on NDCG@10 and HR@10. Moreover, due to complex lookup mechanisms, the overall CPU time of the TTD method is much longer than that of linear quantization in the inference stage. For a fair comparison, no cache-related optimization is implemented and latency is tested within each batch, where the batch size is 40 and the sequence length is 25.

Table 5: Compression method comparison.

Method	Ndcg@10	Hit@10	comp-rate	Latency
TTD	0.2201	0.3877	120.6x	583ms
Linear Quant	0.2272	0.3989	8x	11.2ms

More importantly, our PRecQ is a general quantization framework that can be combined with other compression techniques, thus the comparison of different compression techniques (e.g. weight-decomposition, pruning) is not the key of this work.

C QUANTIZATION SETTINGS

Our framework is robust to different quantization settings. In this part, we conduct experiments with another non-uniform quantization setting, i.e. log. The performance of 4-bit-GRU4Rec on Steam (NDCG@10/Hit@10) is shown in Table 6.