Efficient On-Device Session-Based Recommendation

XIN XIA and JUNLIANG YU, The University of Queensland, Australia
QINYONG WANG, Baidu Inc., China
CHAOQUN YANG and NGUYEN QUOC VIEU HUNG, Griffith University, Australia
HONGZHI YIN, The University of Queensland, Australia

On-device session-based recommendation systems have been achieving increasing attention on account of the low energy/resource consumption and privacy protection while providing promising recommendation performance. To fit the powerful neural session-based recommendation models in resource-constrained mobile devices, tensor-train decomposition and its variants have been widely applied to reduce memory footprint by decomposing the embedding table into smaller tensors, showing great potential in compressing recommendation models. However, these model compression techniques significantly increase the local inference time due to the complex process of generating index lists and a series of tensor multiplications to form item embeddings. The resultant on-device recommender fails to provide real-time responses and recommendations. To improve the online recommendation efficiency, we propose to learn compositional encoding-based compact item representations. Specifically, each item is represented by a compositional code that consists of several codewords, and we learn embedding vectors to represent each codeword instead of each item. Then the composition of the codeword embedding vectors from different embedding matrices (i.e., codebooks) forms the item embedding. Since the size of codebooks can be extremely small, the recommender model is thus able to fit in resource-constrained devices and save the codebooks for fast local inference. In addition, to prevent the loss of model capacity caused by compression, we propose a bidirectional self-supervised knowledge distillation framework. Extensive experimental results on two benchmark datasets demonstrate that compared with existing methods, the proposed on-device recommender not only achieves an 8× inference speedup with a large compression ratio but also shows superior recommendation performance. The code is released at https://github.com/xiaxin1998/EODRec.

CCS Concepts: • Information systems → Recommender systems;

Additional Key Words and Phrases: Model compression, on-device learning, next-item recommendation, self-supervised learning, knowledge distillation

ACM Reference format:
Xin Xia, Junliang Yu, Qinyong Wang, Chaoqun Yang, Nguyen Quoc Viet Hung, and Hongzhi Yin. 2023. Efficient On-Device Session-Based Recommendation. *ACM Trans. Inf. Syst.* 41, 4, Article 102 (March 2023), 24 pages.
https://doi.org/10.1145/3580364

This work was supported by the Australian Research Council Future Fellowship (Grant No. FT210100624), the Discovery Project (Grant No. DP190101985), and the Discovery Early Career Research Award (Grant No. DE200101465).

Authors’ addresses: X. Xia, J. Yu, and H. Yin (corresponding author), The University of Queensland, Brisbane, Queensland, Australia, 4072; emails: {x.xia, jl.yu}@uq.edu.au, db.hongzhi@gmail.com; Q. Wang, Baidu Inc., Beijing, China, 100085; email: wangqinyong@baidu.com; C. Yang and N. Q. V. Hung, Griffith University, Gold Coast, Queensland, Australia, 4222; emails: {chaoqun.yang, henry.nguyen}@griffith.edu.au.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
1046-8188/2023/03-ART102 $15.00
https://doi.org/10.1145/3580364

ACM Transactions on Information Systems, Vol. 41, No. 4, Article 102. Publication date: March 2023.
1 INTRODUCTION

Recently, session-based recommender systems [4, 28, 39, 50] are emerging, which aim to recommend a ranked list of items that the target user is interested in based on his/her current session context. Meanwhile, of all the shopping channels available to customers, mobile commerce is taking the lead.1 Currently, most session-based recommendation systems are built upon a cloud-based paradigm, where the session-based recommender models are trained and deployed on powerful cloud servers [9, 10]. When a recommendation request is made by a user via a mobile device, the current session information will be sent to the on-cloud session-based recommender models to generate recommendation results. Then, the recommendation results will be returned to the mobile device for display. However, in reality, this recommendation paradigm heavily relies on high-quality wireless network connectivity and increasingly raises public privacy concerns.

To better protect user privacy and reduce communication costs, on-device machine learning [9] that aims to develop and deploy machine learning models on resource-constrained devices has been attracting more and more attention. In this learning paradigm, the machine learning model is first trained on the cloud and then downloaded and deployed on local devices such as smartphones. When using the model, users no longer need to upload their sensitive data (e.g., various context information) to central servers and, therefore, can enjoy privacy-preserving and low-latency services that are orders of magnitude faster than server-side models [23]. Since the principle of on-device machine learning is well-aligned with the need for low-cost and privacy-preserving recommender systems, there has been a growing trend toward on-device recommendation [1, 5, 12, 33, 48, 56]. However, it is non-trivial to deploy the powerful recommender model trained on the cloud to resource-constrained mobile devices, which are confronted with the following two fundamental challenges.

The first challenge is how to reduce the size of on-cloud trained models to fit in resource-constrained devices without compromising recommendation performance. To address this challenge, a typical approach is to employ knowledge distillation (KD) [15] that first trains a teacher model on the cloud by fully utilizing the abundant resources and then transfers the teacher’s knowledge to a lightweight student model. To retain the original recommendation performance, even the student model cannot be small enough to fit in resource-constrained mobile devices without compression. But the classical model compression techniques such as pruning [42], down-sizing [15], and parameter sharing [37] are not so effective in compressing the recommender models. Unlike the vision and language models [13], which are usually over-parameterized with a very deep network structure, the recommendation model only contains several intermediate layers that account for a small portion of learnable parameters. Instead, the item embedding table is the one that accounts for the vast majority of memory footprint [51]. Recent practices of on-device recommendation [43, 48] mostly adopt decomposition techniques such as tensor-train decomposition (TTD) [35] to reconstruct the embedding table with a series of matrix multiplication. However, to avoid a drastic recommendation performance drop, they can only compress the embedding table with a small compression rate (e.g., 4–8×), which is far from meeting the resource-constrained conditions on mobile devices. In addition, recent studies show that models with similar structures (e.g., encoder-decoder) are easier to transfer knowledge [2], while the tensor decomposition widens the structural gap between the teacher and the student.

The second challenge is how to provide real-time response and recommendation. When a user interacts with new items in the current session, the recommendation system should provide a response by updating recommendations immediately. The response speed to generate new recommendations is crucial for user experience in mobile e-commerce. However, the widely used

1 https://www.shopify.com/au/enterprise/mobile-commerce-future-trends.
recommendation model compression techniques (TTD and its variants) significantly delay the on-device model inference due to their complex process of generating index lists and a series of multiplications to form item embeddings, and the resultant on-device recommenders fail to provide real-time response and recommendation.

In our previous work [54], we proposed semi-tensor decomposition and self-supervised KD techniques to address the first challenge, based on which an ultra-compact and effective on-device recommendation model OD-Rec was developed. Specifically, given the dilemma that the small TT-rank in TTD leads to under-expressive embedding approximations, whereas the larger TT-rank sacrifices the model efficiency [32, 35], we introduced the semi-tensor product (STP) operation [65] to TTD for the extreme compression of the embedding table. This operation endows TTD with the flexibility to perform multiplication between tensors with inconsistent ranks, considering both effectiveness and efficiency. With semi-tensor decomposition to compress the embedding table, we achieved an extremely compact model of 30× smaller size than the original model. In addition, we proposed recombining the embeddings learned by the teacher and the student to perform self-supervised learning, which can further distill the essential information lying in the teacher’s embeddings to compensate for the accuracy loss caused by the extremely high compression. However, the second challenge has not been addressed in [54]. As a variant of TTD, the process of reconstructing item embedding in our semi-tensor decomposition is also very complex, which significantly slows down the online model inference to generate recommendations. An index list that is used to map the rows and columns in the factorized tensors should be first calculated, and it takes more time to do a series of multiplication between the mapped vectors.

In light of this, we extend our previous work [54] by developing a new compositional code-based compression method to replace the previous semi-tensor decomposition. In the new method, we propose representing each item by a unique $M$-dimensional code consisting of $M$ discrete codewords, where $M$ is much smaller than the original embedding dimension. There are $M$ codebooks, each containing $K$ codeword vectors (i.e., basic vectors). We learn embedding vectors for each codeword rather than each item and then simply sum up the corresponding codeword vectors to construct item embedding. For example, given a compositional code $C_i = (2, 3, 5, 1)$ for item $i$, and four codebooks $E_1, E_2, E_3, E_4$, then the item embedding of $i$ is computed as

$$E(C_i) = \sum_{t=1}^{4} E_t(C'_t),$$

(1)

where $E_t(C'_t)$ is the $C'_t$-th codevector in codebook $E_t$. To learn effective item compositional embeddings, we minimize the difference between them and the well-trained uncompressed item embeddings. Meanwhile, the recommendation task is jointly optimized with this regularization term to ensure a smooth knowledge transfer. After convergence, the well-trained codebooks and codes are saved to be used in the on-device recommender. Therefore, on the resource-constrained devices, the total memory footprint for items is of size $|V|M + MKN$, where $N$ denotes the dimension of a codeword vector. We can adjust the value of $M$ to compress the item embedding table to any degree. In addition, our previous KD framework only transfers the teacher’s knowledge to the student based on the assumption that the teacher always outperforms the student. However, our experiments in [54] demonstrated that the lightweight student model achieves better recommendation performance on some metrics, which is consistent with the findings in [22]. Inspired by this observation, we propose a bidirectional KD framework that also enables the knowledge transfer from the student to the teacher. It means that both the teacher and the student are iteratively updated in the framework. Particularly, we find that the student and the teacher trained under this new framework are superior to those trained under our previous framework.
To summarize, our major new contributions are listed as follows:

— We make the exploration of an ultra-compact efficient on-device session-based recommendation by integrating discrete compositional code learning into recommender systems to compress an item embedding table.

— We propose a bidirectional KD framework that enables the teacher and the student to mutually enhance each other during the learning process.

— We conduct extensive experiments to evaluate the online inference efficiency, recommendation accuracy, and model compression ratio. The experimental results on two benchmark datasets demonstrate that the proposed new method completely outperforms the previous ones, showing an 8× speedup in model inference with smaller model size and higher recommendation accuracy Figure 1.

The remainder of the article is organized as follows. In Section 2, the related work is reviewed in detail, followed by the description of the preliminaries in Section 3. In Section 4, the detail of the proposed compression method is presented. Then, we show how to use self-supervised KD to prevent the loss of model capacity in Section 5. Extensive experiments are conducted and analyzed to verify the efficacy of the proposed model in Section 6. At last, we conclude our work and discuss the future work in Section 7.

2 RELATED WORK

In this section, we briefly review the literature on four related topics: on-device recommendation, session-based recommendation, KD for recommendation, and compositional code learning. The former two are the scenarios where our method is applied, and the latter two are techniques that our method involves.

2.1 On-Device Recommendation

On-device machine learning [9] has been drawing increasing attention in many fields due to its advantages in local inference and privacy protection. The fundamental challenge of on-device learning is how to reduce the size of conventional models so as to adapt to resource-constrained devices. The existing research [1, 5, 12, 33] mainly adopts model compressing techniques like pruning [13, 42], quantization [11], and low-rank factorization [32, 35] to achieve this goal. Recently, on-device recommender systems emerged in response to users’ needs for low latency recommendations and have demonstrated desired performance with tiny model sizes. Specifically,
Chen et al. [5] proposed to learn elastic item embeddings to allow recommendation models to be customized for arbitrary device-specific memory constraints without retraining. WIME et al. [1] proposes to use user’s real-time intention to generate on-device recommendations where a sequence and context-aware algorithm was used to embed user intention and a neighborhood searching method followed by a sequence matching algorithm was used to do prediction. DeepRec [12] uses model pruning and embedding sparsification techniques to enable model training with a small memory budget and fine-tunes the model using local user data, reaching comparable performance in sequential recommendation with a 10× parameter reduction. Wang et al. [48] proposed LLRec which is for next POI recommendation model and is based on tensor-train factorization. TT-Rec [56] also applies TTD to the embedding layer for model compression and further improves model performance with a sampled Gaussian distribution for the weight initialization of the tensor cores. Despite the effectiveness of these methods, they are not designed for session-based recommendation, and many of them are still with a large number of parameters that may overload small devices.

### 2.2 Session-Based Recommendation

The early session-based recommendation models [39, 57] are based on Markov Chain and mainly focus on modeling temporal order between items. In the deep learning era, neural networks like RNNs [4, 14, 64] are tailored to model session data and handle the temporal shifts within sessions. The follow-up models such as NARM [25] and STAMP [28] employ attention mechanisms to prioritize items when profiling users’ main interests. Pan et al. [36] proposed to rethink item importances in the session-based recommendation and estimate the importance of items in a session by employing an Importance Extracting Module based on a modified self-attention mechanism. Recently, graph-based methods [38, 46, 50] have demonstrated state-of-the-art performance, which constructs various session graphs to model item transitions. Specifically, SR-GNN [50] constructs session graphs for every session and designs a gated graph neural network to aggregate information between items into session representations. GC-SAN [55] employs a self-attention mechanism to capture item dependencies and user interests via graph information aggregation. GCE-GNN [49] proposes to capture both global-level and session-level interactions and aggregates item information through graph convolution and self-attention mechanisms. Xia et al. [52, 53] proposed integrating self-supervised learning into the session-based recommendation to boost recommendation performance. Transformer-based session-based recommendation models also achieve promising performance by integrating techniques from Transformer into session-based scenarios [7, 8]. A representative method is BERT4SessRec [7], which employs BERT for session-based recommendation by using bidirectional encoder representations from Transformer to capture bidirectional correlations in each session effectively. These models are of great capacity in generating accurate recommendations but they were designed for server-side use, which cannot run on resource-constrained devices like smartphones.

### 2.3 KD for Recommendation

KD [15] is a widely used approach for transferring knowledge from a well-trained large model (teacher) to a simple and lightweight model (student). In this framework, the student model is usually optimized toward two objectives: minimizing the difference between the prediction and the ground truth and fitting the teacher’s label distribution or intermediate layer embeddings. KD was first introduced to the classification problem, and currently, some KD methods have been proposed for recommender systems [19, 24, 44]. The first work is Ranking Distillation [44], which chooses top-K items from the teacher’s recommendation list as the external knowledge to guide the student model to assign higher scores when generating recommendations. However, this ignores...
rich information beyond the recommended top-K items. The follow-up work, CD [24], introduces a rank-aware sampling method to sample items from the teacher model as knowledge for the student model, enlarging the range of information to be transferred. DE-RRD [19] develops an expert selection strategy and relaxes the ranking-based sampling method to transfer knowledge. Though effective, these KD methods can hardly tackle the cold-start problem caused by the long-tail distribution in recommender systems.

2.4 Compositional Code Learning

One-hot encoding is a standard technique in deep representation learning, in which each entity is associated with a continuous embedding vector and usually leads to a large embedding table to be stored. To reduce the memory footprint, some research has explored efficient coding systems, for example, Huffman Code [13, 17] and Hash functions [6, 45]. However, these coding systems often suffer from low accuracy due to limited representation capability of binary codes. Subsequent works [3, 41] then explore the use of additive quantization for source coding, resulting in compositional codes. Specifically, Shu and Nakayama [41] proposed an end-to-end compositional encoding-based neural network to compress the word embeddings in NLP, achieving success in compression ratios and in the meantime being independent of languages. Chen et al. [3] introduced a K-way d-dimensional discrete code for compact embedding representations, which can be generally applied to any differentiable computational graph with an embedding layer. Compositional code has also been explored in recommender systems [20, 27, 40]. Kang et al. [20] proposed dense hash encoding to emb categorical features by using multiple hash functions and transformations, which is very close to compositional encoding. Shi et al. [40] proposed to reduce embedding size by exploiting complementary partitions of the category set to produce a unique embedding for each category at the smallest cost. Liu et al. [27] represented items/users with a set of binary vectors associated with a sparse weight vector, and an integer weight approximation scheme is proposed to accelerate the speed of the method. But their methods either require category or feature information or bear a high computational cost. Li et al. [26] proposed using compositional code to compress an item embedding table in the sequential recommendation. However, the model adopts a quotient-remainder trick to make the codes of items distinct, which increases the time complexity. In addition, its model compression ratio is rather limited.

3 PRELIMINARIES

In this section, we first introduce the session-based recommendation task and then give a brief review of the backbone model and the compression method used in our previous work to help understand the new contributions.

3.1 Session-Based Recommendation Task

Session-based recommender systems rely on the previous clicks in a short period of time to generate the next-item recommendation. Let $\mathcal{V} = \{v_1, v_2, v_3, \ldots, v_{|\mathcal{V}|}\}$ denote the item set and $s = [v_{s,1}, v_{s,2}, \ldots, v_{s,l}]$ denote a session sequence. Every session sequence is composed of interacted items in chronological order from an anonymous user. The task of session-based recommendation is to predict the next item, namely, $v_{s,l+1}$. In this scenario, every item $v \in \mathcal{V}$ is first mapped into the embedding space. The item embedding table is with a huge size and is denoted as $X \in \mathbb{R}^{|\mathcal{V}| \times N}$. Given $\mathcal{V}$ and $s$, the output of the session-based recommendation model is a ranked list $y = [y_1, y_2, y_3, \ldots, y_{|\mathcal{V}|}]$ where $y_v$ ($1 \leq v \leq |\mathcal{V}|$) is the corresponding predicted probability of item $v$. The top-K items ($1 \leq K \leq |\mathcal{V}|$) with the highest probabilities in $y$ will be selected as the recommendations.
3.2 A Brief Review of OD-Rec

In our previous work OD-Rec, we unify the server-side model (teacher) and the on-device model (student) into a KD framework. The teacher and the student adopt the Transformer-based sequential recommendation model SASRec [21] as their backbones to generate a next-item recommendation. To make the student fit in resource-constrained devices, we compress its embedding table through the TTD [35]. Different from other TTD-based models [48, 56], our student model relaxes the constraint of dimensionality consistency in decomposition so as to achieve a higher compression rate. To retain the capacity of the student, we not only employ the traditional soft targets distillation but also devise two self-supervised distillation tasks to transfer the teacher’s knowledge to the student.

3.2.1 Backbone Model Structure. The backbone used in our framework is a famous Transformer-based [47] sequential recommendation model in which several self-attention blocks are stacked, including the embedding layer, the self-attention layer, and the feed-forward network layers. The embedding layer adds position embeddings to the original item embeddings to indicate the temporal and positional information in a sequence. The self-attention mechanism can model the item correlations. Inside the block, residual connections, neuron dropout, and layer normalization are sequentially used. SASRec with a one-layer setting can be formulated as

$$\hat{X} = X + P, \quad F = \text{Attention}(\hat{X}), \quad \Theta = \text{FNN}(F),$$

where $X$, $P$ are the item embeddings and position embeddings, respectively, and $F$ is the session representation learned via self-attention blocks. FNN represents feed-forward network layers and $\Theta$ aggregates embeddings of all items. In origin SASRec, the embedding of the last clicked item in $\Theta$ is chosen as the session representation. However, we think each contained item would contribute information to the session representation for a comprehensive and more accurate understanding of user interests. Instead, we slightly modify the original structure by adopting the soft-attention mechanism [49] to generate the session representation, which is computed as

$$\alpha_t = f^\top \sigma (W_1 x_s^* + W_2 x_t + c), \quad \theta_s = \sum_{t=1}^{l} \alpha_t x_t,$$

where $W_1 \in \mathbb{R}^{N \times N}$, $W_2 \in \mathbb{R}^{N \times N}$, $c, f \in \mathbb{R}^N$ are learnable parameters, $x_t$ is the embedding of item $t$, and $x_s^*$ is obtained by averaging the embeddings of items within the session $s$, i.e., $x_s^* = \frac{1}{l} \sum_{t=1}^{l} x_t$. $\sigma$ is the sigmoid function. Session representation $\theta_s$ is represented by aggregating item embeddings while considering their corresponding importance.

3.2.2 Model Compression in Our Previous Work. To factorize the embedding table of the student model into smaller tensors, we let the total number of items $|V| = \prod_{k=1}^{d} I_k$ and the embedding dimension $N = \prod_{k=1}^{d} J_k$. For one specific item $v$ indexed by $i$, we map the row index $i$ into $d$-dimensional vectors $i = (i_1, i_2, \ldots, i_d)$ according to [16, 35], and then get the particular slices of the index in TT-cores to perform matrix multiplication on the slices. Benefiting from this decomposition, the model just needs to store the TT-cores to fulfill an approximation of the entry in the embedding table, i.e., a sequence of small tensor multiplication. The TTD can be formed as

$$X = G_1 \times G_2 \times \cdots \times G_d,$$

where $X$ is a $d$-dimensional embedding table and $|G_k|_{k=1}^{d}$ are called TT-cores, $G_k$ is of the size $\mathbb{R}^{R_{k-1} \times I_k \times J_k \times R_k}$, and the sequence of $R_k$ is called TT-rank where $R_0 = R_d = 1$ and we set $R_k = R_{k-1} = R$. Each entry in $X$ indexed by $(i_1, i_2, \ldots, i_d)$ can be represented in the following
TT-format:

\[
x = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \ldots \sum_{r_{d-1}=1}^{R_{d-1}} G_1 (i_1, r_1) G_2 (r_1, i_2, r_2) \ldots G_d (r_{d-1}, i_d)
\]

\[= G_1 \left[ i_1, \ldots, i_{d-1} \right] G_2 \left[ i_1, i_2, \ldots, i_{d-1} \right] \ldots G_{d-1} \left[ i_1, i_2, \ldots, i_{d-1} \right] G_d \left[ i_1, \ldots, i_{d-1} \right].
\]

However, conventional TTD requires strict dimensionality consistency between factors and probably leads to dimension redundancy. For example, if we decompose the item embedding matrix into two smaller tensors, i.e., \( X = G_1 G_2 \), to get each item’s embedding, the number of columns of \( G_1 \) should be the same with the number of rows of \( G_2 \). This consistency constraint is too strict for flexible and efficient tensor decomposition, posing an obstacle to further compression of the embedding table. Therefore, inspired by [65], we propose to integrate STP with TTD where the dimension of TT-cores can be arbitrarily adjusted. Let \( a \in \mathbb{R}^{1 \times np} \) denote a row vector and \( b \in \mathbb{R}^p \), then \( a \) can be split into \( p \) equal-size blocks as \( a^1, a^2, \ldots, a^p \), and the left STP denoted by \( \times \) can be defined as

\[
\begin{align*}
\{ a \times b &= \sum_{i=1}^{p} a^i b_i \in \mathbb{R}^{1 \times n} \\
 b^T \times a^T &= \sum_{i=1}^{p} b_i (a^i)^T \in \mathbb{R}^n.
\end{align*}
\]

Then, for two matrices \( A \in \mathbb{R}^{H \times np} \), \( B \in \mathbb{R}^{P \times Q} \), STP is defined as

\[ C = A \times B, \]

and \( C \) consists of \( H \times Q \) blocks and each block \( C^{hq} \in \mathbb{R}^{1 \times n} \) can be calculated as

\[ C^{hq} = A(h, :) \times B(:, q), \]

where \( A(h, :) \) represents the \( h \)-th row in \( A \), \( B(:, q) \) represents the \( q \)-th column in \( B \), and \( h = 1, 2, \ldots, H \), \( q = 1, 2, \ldots, Q \). With the above definitions, the conventional tensor product between TT-cores can be replaced with the STP as

\[ X = \hat{G}_1 \times \hat{G}_2 \times \cdots \times \hat{G}_d, \]

where \( \{ \hat{G}_k \}_{k=1}^d \) are the core tensors after applying semi-tensor product based tensor-train decomposition (STTD) and \( \hat{G}_1 \in \mathbb{R}^{I_1 J_1 \times R}, \{ \hat{G}_k \}_{k=2}^d \in \mathbb{R}^\frac{E}{n} \times \frac{I_k J_k}{n} \times R, \hat{G}_d \in \mathbb{R}^\frac{E}{n} \times \frac{I_d J_d}{n} \). STP loosens the strict dimensionality consistency by enabling the factorized tensors to do the product as long as their corresponding dimensions are proportional. The compression rate is

\[
\text{rate} = \frac{\sum_{k=1}^{d} I_k J_k}{I_1 J_1 R + \sum_{k=1}^{d-1} I_k J_k \frac{E}{n} + I_d J_d \frac{E}{n}}.
\]

We can adjust the TT-rank, namely, the values of the hyperparameters \( R \) and \( n \), and the length \( k \) of the tensor chain to flexibly compress the model to any degree.

### 4 COMPOSITIONAL ENCODING FOR MODEL COMPRESSION

Although the TTD empowered by the STP can reduce a substantial memory footprint, the multiplication of tensors also greatly increases the inference time. In this section, we present our new compression method which is based on the compositional code learning.
4.1 Discrete Code Embedding Framework

We propose to compress the item embedding table by enabling items to share embedding vectors through a discrete code learning mechanism. Specifically, each item $v$ will be represented by a unique code $C_v = (C_{1v}, C_{2v}, \ldots, C_{Mv}) \in \mathcal{B}^M$ where each component in the code is a discrete number ranged in $[1, K]$ and $\mathcal{B}$ is the set of code bits with cardinality of $M$. There is a code allocation function $\Phi(\cdot) : \mathcal{V} \rightarrow \mathcal{B}^M$ that maps each item with its discrete code. Since the discrete code has $M$ values, we cannot directly use the code to look up the item vector in the embedding table; instead, we expect to learn a code composition function $\Theta$ that takes a discrete code of item $v$ as input and generates the corresponding continuous embedding vector as output, i.e., $\Theta : \mathcal{B}^M \rightarrow \mathbb{R}^N$. Then the discrete code framework can be formed as

$$\Gamma = \Phi \cdot \Theta.$$  \hfill (11)

Given learned $\Phi$ and item $v$, we can get its code, i.e., $C_v = \Phi(v) = (C_{1v}, C_{2v}, \ldots, C_{Mv})$. In order to get the composite embedding, we adopt a code composition function $e_v = \Theta(C_v)$. We first embed $C_v$ to a sequence of embedding vectors $(E_{1v}^{C_1}, E_{2v}^{C_2}, \ldots, E_{iv}^{C_i})$. And then we apply embedding transformation to generate $e_v$. Here we create $M$ codebooks $E_1, E_2, \ldots, E_M$, each containing $K$ vectors with the dimensionality of $N$. The final embedding of item $v$ is

$$e_v = \sum_{i=1}^{M} E_i \left( C_{iv} \right),$$ \hfill (12)

where $E_i(C_{iv})$ is the $C_{iv}$-th vector in codebook $E_i$. Therefore, in this case, we just need to store the integer codes of all items, which is called code matrix $C$ and the $M$ codebooks. We present the processes of one-hot encoding and compositional encoding in Figure 2. For one-hot encoding, the number of vectors used in representing items is identical to the number of items. In contrast, the number of vectors for compositional approach required to construct the embeddings is $M \times K$. To store each code, $M \log_2 K$ bits are required, so $K$ is selected to be a number of a multiple of 2 for the convenience of storage. In this case, the compression ratio can be calculated as

$$\text{rate} = \frac{|\mathcal{V}| \cdot N}{MN + M \cdot |\mathcal{V}|} = \frac{N}{\frac{MN}{|\mathcal{V}|} K + M}. \hfill (13)$$

We can adjust the value of $M$ and $K$ to flexibly compress the model to any degree. As shown in the above equation, the upper bound of the compression ratio is $N/M$. Therefore, when $N$ is fixed, we should seek small $M$ to have a larger compression ratio. Here we show the varying model compression ability with different values of $M$ and $K$ in Table 1. We assume that there are 20,000 items and the dimension of each embedding vector is 100. Then we calculate the model size before and after model compression and present them in Table 1. It is clear that the value of $M$ affects the compression ratio more than $K$ because $MN \ll |\mathcal{V}|$.

4.2 Compositional Code Learning

Although the embedding size can be greatly reduced after using discrete code, we expect to prevent serious performance degradation and maintain model capacity in next-item recommendation. Namely, given baseline embedding matrix $X$ (here the baseline embedding is the well-trained item embedding matrix in the teacher model because we have a KD framework), we want to find a set of codes $C$ and codebooks $E$ that can generate embeddings with the same effectiveness as $X$. A straightforward way is to minimize the squared distance between the original embedding and the
Fig. 2. Difference between one-hot encoding and compositional encoding.

Table 1. Model Size Analytics with Different M and K

| Original size of item embeddings | Codebook number M | Vector number K | Size after compression | Compression rate |
|----------------------------------|------------------|-----------------|------------------------|-----------------|
| 2,000,000                        | 2                | 8               | 41,600                 | 48              |
|                                  |                  | 128             | 65,600                 | 30              |
|                                  |                  | 512             | 142,400                | 14              |
| 2,000,000                        | 4                | 8               | 83,200                 | 24              |
|                                  |                  | 128             | 131,200                | 15              |
|                                  |                  | 512             | 284,800                | 7               |
| 2,000,000                        | 8                | 8               | 185,600                | 11              |
|                                  |                  | 128             | 262,400                | 8               |
|                                  |                  | 512             | 569,600                | 3               |

composite embedding:

$$L_{mse} = \arg \min_{C,E} \frac{1}{|V|} \sum_{v \in V} \| E(C_v) - X_v \|^2$$

$$= \arg \min_{C,E} \frac{1}{|V|} \sum_{v \in V} \left( \sum_{i=1}^{M} E_i(C^i_v) - X_v \right)^2.$$  \hspace{1cm} (14)

4.2.1 End-To-End Learning with Gumbel-Softmax. To minimize the MSE loss means we should optimize the item-to-code mapping function $\Phi$ and the code-to-embedding composition function $\Theta$. However, each code is discrete so the learning is not differentiable. In order to enable an end-to-end learning, we consider that each code $C_v$ can be seen as a concatenation of $M$ one-hot vectors, i.e., $C_v = (O^1_v, O^2_v, \ldots, O^M_v)$, where $\forall i, O^i_v \in [0, 1]^K$ and $\sum_k O^i_k = 1$, and $O^i_k$ is the $k$-th component of $O^i_v$. Let $O^1, O^2, \ldots, O^M$ represent $M$ code matrices where each matrix is with the size of $|V| \times K$ and each row in each matrix is a $K$-dimensional one-hot vector. Then the generation of the item’s embedding can be reformulated to be

$$e_v = \sum_{i=0}^{M} E_i^T O^i_v,$$  \hspace{1cm} (15)

where $O^i_v$ represents the one-hot vector corresponding to the code component $C^i_v$ of item $v$. Therefore, the optimization of the item-to-code-embedding function becomes to find an optimal set of one-hot code matrices $O^1, O^2, \ldots, O^M$ and basis codebooks $E_1, E_2, \ldots, E_M$, that minimize the reconstruction loss $L_{mse}$ mentioned above. Inspired by [41], we then adopt Gumbel-Softmax [18] to make the continuous vector $O^i_v$ approximate the one-hot vector. The $k$-th element in $O^i_v$ is
computed as

\[
    (O^t_v) = \text{softmax} \left( \log \alpha^t_v + G \right)
\]

\[
    = \frac{\exp \left( \left( \log \alpha^t_v + G \right) / \epsilon \right)}{\sum_{k=1}^{K} \exp \left( \left( \log \alpha^t_v + G^t_k \right) / \epsilon \right)}
\]

where \( G_k \) is a noise term that is sampled from the Gumbel distribution \(-\log(\log(\text{Uniform}(0, 1)))\), and \( \epsilon \) is the temperature in softmax. As \( \epsilon \to 0 \), the softmax computation smoothly approaches the argmax. (In our setting, we empirically set \( \epsilon \) to 0.3, which is reported as a good choice in many previous studies). And \( \alpha^t_v \) is computed by a two-layer MLP:

\[
    \alpha^t_v = \text{softmax} \left( \text{softplus}(\theta^i_v h^i_v + b^i) \right)
\]

\[
    h^i_v = \tanh \left( \theta^i X^i_v + b^i \right)
\]

where \( X^i_v \) is the well-trained embedding of item \( i_v \), \( \theta^i \in \mathbb{R}^{N \times (MK/2)} \), \( \theta^i_v \in \mathbb{R}^{(MK/2) \times MK} \), \( b^i \in \mathbb{R}^{MK/2} \), and \( b \in \mathbb{R}^{MK} \). By using this reparameterization trick, the model can circumvent the indifferentiable look-up operation and allows the gradients from the MSE loss to be delivered to the codebooks.

### 4.2.2 Code Learning with Guidances.

Although compositional encoding can compress the embedding table, it will definitely degrade recommendation performance. In order to retain the model capacity and guide the compositional code learning, we propose embedding mixup in the process of code learning. Specifically, during training, instead of solely using the generated embedding from Equation (15), we use an interpolation of the composite embedding \( e^i_v \) and the uncompressed embedding \( X^i_v \):

\[
    e^i_v = \eta \odot X^i_v + (1 - \eta) \odot e^i_v,
\]

where \( \eta \in (0, 1) \) is a hyperparameter. We can tune it to achieve the best performance. Note that we only use the embedding mixup in training. In the test/inference phase, only the compositional embedding is used for prediction.

### 4.2.3 Time Complexity Analysis.

In this section, we compare the time complexity of the proposed new compression method and the previous TTD. The discussion focuses on the process of reconstructing item embeddings, which is critical for fast local inference. Recall that, in TTD, for each item there is an index list to calculate, followed by a series of tensor multiplications between the vectors looked up from factorized tensors with the index list. In our new method, to generate item embeddings, we just need the matrix multiplication between the code matrix and codebooks. Here we let \( |V| \) be the number of items, \( N \) is the embedding dimension, \( R \) is the value of TT-rank in TTD, \( M \) is the number of codebooks, and \( K \) is the number of vectors in each codebook. In TTD, the time complexity of the index list calculation for \( n \) items is \( O(|V|^2 + |V|) \); as for the multiplication between indexed embedding vectors, its time complexity is \( O(|V|R) \). Then the total time complexity of TTD is \( O(|V|^2 + |V| + |V|R) \). When using compositional codes to represent items, we first pre-train the code matrix and codebooks and then they are saved to form item embeddings. Therefore, there is only one-time matrix multiplication between the code matrix and the codebooks, and the time complexity is \( O(|V|MKN) \). Since \( M \) and \( K \) are small, the new compression method is theoretically much more efficient than the old one.

### 5 BIDIRECTIONAL SELF-SUPERVISED KD

In our previous article, we proposed a self-supervised KD framework to improve the capacity of the on-device model in a teacher-student fashion. In this article, we advance it by proposing bidirectional KD.
5.1 Framework Formulation

Since we are the first to combine self-supervised learning and KD for recommendation, we define the general framework of self-supervised KD for recommendation as follows. Let \( D \) denote the item interaction data, \( D_{\text{soft}} \) represent the soft targets from the teacher, and \( D_{\text{aug}} \) denote the augmented data under the self-supervised setting. The teacher model and the student model are denoted by \( M_T \) and \( M_S \), respectively. Then the framework is formulated as follows:

\[
f \left( M_T(D), M_S(D, D_{\text{soft}}, D_{\text{aug}}) \right) \rightarrow M^*_S,
\]

which means that by jointly supervising the student model with the historic interactions \( D \), the teacher’s soft targets \( D_{\text{soft}} \) and the self-supervised signals extracted from \( D_{\text{aug}} \), we can finally obtain an improved student model \( M^*_S \) that retains the capacity of the regular model.

5.2 Data Augmentation

In this framework, two types of distillation tasks, traditional distillation task based on soft targets and contrastive self-supervised distillation task, are integrated. The essential idea of self-supervised learning \([29, 59, 60]\) is to learn with the supervisory signals which are extracted from augmentations of the raw data. However, in recommender systems, interactions follow a long-tail distribution. For the long-tail items with few interactions, generating informative augmentations is often difficult \([62]\). Inspired by the genetic recombination \([31]\) and the preference editing in \([30]\), we come up with the idea to exchange embedding segments from the student and the teacher and recombine them to distill knowledge. Such recombinations enable the direct information transfer between the teacher and the student, and meanwhile create representation-level augmentations which inherit characteristics from both sides. Under the direct guidance from the teacher, we expect that the student model can learn more from distillation tasks. We term this method embedding recombination.

To implement it, we divide items into two types: hot and cold by their popularity. The top 20% popular items are considered as hot items and the rest are cold items. For each session, we split it into two sub-sessions, cold session and hot session, in which only the cold items or hot items are contained. Then we learn two corresponding session representations, respectively: hot session representation and cold session representation. The hot session representation derives from representations of hot items in that session based on the soft attention mechanism in Equation (3), while the cold session representation is analogously learned from representations of cold items. They are formulated as follows:

\[
\theta^\text{hot}_s = \sum_{t=1}^{l_h} \alpha^\text{hot}_t x^\text{hot}_t, \quad \theta^\text{cold}_s = \sum_{t=1}^{l_c} \alpha^\text{cold}_t x^\text{cold}_t,
\]

where \( l_c \) and \( l_h \) are lengths of the cold session and the hot session, \( x^\text{hot}_t, x^\text{cold}_t \) represents the representation of the \( t \)-th hot item or cold item in the session \( s \), \( \alpha^\text{hot}_t (\alpha^\text{cold}_t) \) is the learned attention coefficient of the \( t \)-th hot or cold item in the session, and \( \theta^\text{hot} (\theta^\text{cold}) \) is the learned hot or cold session representation. In both the student and the teacher models, we learn such session embeddings. Then for the same session, we swap their cold session representations as follows to generate new embeddings:

\[
z^\text{tea}_s = [\theta^\text{hot}_s, \theta^\text{cold}_s], \quad z^\text{stu}_s = [\theta^\text{hot}_s, \theta^\text{cold}_s].
\]

As for those sessions which only contain hot items or cold items, we generate the corresponding type of session representations and swap them.
5.3 KD Tasks

5.3.1 Contrastive Distillation. Since the generated new embeddings are profiling the same session, we consider that there is shared learnable invariance lying in these embeddings. Therefore, we propose to contrast recombined embeddings so as to learn discriminative representations for the student model, which can mitigate the data sparsity issue to some degree. Meanwhile, this contrastive task also helps to transfer the information in the teacher’s representation, rather than only transferring the output probabilities. We follow [58, 61] to use InfoNCE [34] to maximize the mutual information between teacher’s and student’s session representations. For any session \(s\), its recombined representations are positive samples of each other, while the recombined representations of other sessions are its negative samples. We conduct negative sampling in the current batch \(\mathcal{B}\). The loss of the contrastive task is defined as follows:

\[
L_{cl} = -\sum_{s \in \mathcal{B}} \log \frac{\psi(W_t z_s^{tea}, W_s z_s^{stu})}{\psi(W_t z_s^{tea}, W_s z_s^{stu}) + \sum_{j \in \mathcal{B}/\{s\}} \psi(W_t z_s^{tea}, W_s z_j^{stu})},
\]

where \(\psi(W_t z_s^{tea}, W_s z_s^{stu}) = \exp(\phi(W_t z_s^{tea}, W_s z_s^{stu})/\tau)\), \(\phi(\cdot)\) is the cosine function, \(\tau\) is temperature (0.2 in our method), and \(W_t\) and \(W_s \in \mathbb{R}^{N \times 2N}\) are projection matrices.

5.3.2 Soft Targets Distillation. The self-supervised KD tasks rely on data augmentations. Though effective, there may be divergence between original representations and recombined representations. We finally distill the soft targets (i.e., teacher’s prediction probabilities on items). Following the convention, we adopt KL Divergence to make the student generate probabilities similar to the teacher’s. Let \(\text{prob}^{tea}\) and \(\text{prob}^{stu}\) represent the predicted probability distributions of the teacher and the student, respectively. Then for each session \(s\), we have

\[
\text{prob}^{tea} = \text{softmax}\left(\theta^{tea}_s^\top X\right), \quad \text{prob}^{stu} = \text{softmax}\left(\theta^{stu}_s^\top X\right); \quad (23)
\]

\(\theta^{tea}_s\) and \(\theta^{stu}_s\) are representations of session \(s\) from teacher and student. Then to maximize the agreement between them, we define the loss of distilling soft targets as

\[
L_{soft} = -\sum_{v=1}^{|V|} \text{prob}^{tea}_v \ln \frac{\text{prob}^{stu}_v}{\text{prob}^{tea}_v}. \quad (24)
\]

5.4 Bidirectional KD

The self-supervised KD framework has been demonstrated to be effective in our previous work. However, the framework also has limitations. To be specific, the framework is based on the assumption that the teacher is superior to the student so the teacher can transfer its knowledge to help the student. But in practice, the teacher is not always superior to the student and sometimes the student can even achieve better performance, which has been reported in the results in our previous work. Such a phenomenon is also observed in [22]. The possible reason is that the teacher may mistakenly rank some items which should be recommended low whereas the student can correctly rank them high. Therefore, in this work we conduct bidirectional KD which allows the reverse transfer knowledge from the student to the teacher. In this way, the two models can recursively learn from each other and get improved. Since both the teacher and the student are trainable, the upgraded framework is reformulated as

\[
f \left(M_T(D, D_{soft}, D_{aug}), M_S(D, D_{soft}, D_{aug})\right) \rightarrow M^*_S, M^*_T. \quad (25)
\]
ALGORITHM 1: The training process of the proposed method.

**Input**: session data $D$  
**Output**: Recommendation lists

1. Train the teacher model $M_T$ on the cloud and get the well-trained item embedding matrix $X$;  
2. Pre-train the student with Equation (24) to get the codebooks $E_1, E_2, \ldots, E_M$ and code matrix $C$ for items;  
3. **for each iteration** do  
   4. Form item embedding through Equations (13)–(16);  
   5. Compute $L_{mse}$ through Equation (12);  
   6. Form session representations through Equations (18) and (19);  
   7. Compute the recommendation loss by following Equations (25) and (26);  
   8. Compute the KD loss by following Equations (20)–(22);  
   9. Jointly optimize the model using Equation (27) to update both $M_T$ and $M_S$.  
4. end

5.5 Training Scheme and Model Optimization

We first train the teacher model in the cloud where the embedding table is not compressed. Then, the student model is trained under the help of the teacher in the cloud. Finally, the well-trained lightweight student model is downloaded and saved on the device for local inference. Before training with the KD framework, we first pre-train the student with the recommendation task to get the well-trained code matrix and codebooks. The pre-training objective can be formed as

$$L_{pre} = L_{rec} + L_{mse}.$$  

(26)

As for the recommendation task, we use the inner product of the candidate item embeddings $E'$ (obtained through embedding composition in Equation (10) and embedding mixup in Equation (16)) and the given session $s$ to denote the scores of items, then we apply softmax to compute the probabilities of items to be recommended:

$$\hat{y} = \text{softmax} (\theta^T s E'),$$

(27)

where $\theta_s$ is computed through Equation (3). The cross-entropy is used as the loss function of the recommendation task:

$$L_{rec} = - \sum_{v=1}^{V} y_v \log (\hat{y}_v) + (1 - y_v) \log (1 - \hat{y}_v),$$

(28)

where $y$ is the one-hot encoding vector of the ground truth. For simplicity, we leave out the $L_2$ regularization terms. After this warm-up, we unify the teacher and the student into the bidirectional self-supervised KD framework. Then the total loss is

$$L = L_{rec} + L_{mse} + L_{kd} = L_{rec} + L_{mse} + \beta L_{con} + \gamma L_{soft},$$

(29)

where $\beta, \gamma$ are coefficients that control the magnitudes of two KD tasks. The whole training process is presented in Algorithm 1.

6 EXPERIMENTS

In this section, we first describe the experimental settings, including datasets, baselines, and hyper-parameter settings in Section 6.1. Then we evaluate recommendation performances of all baseline methods in Section 6.2. In Section 6.3, we explore the effectiveness of the compressed model by using different values of key hyperparameters. To investigate the on-device inference efficiency
of the proposed model, we compare it with other baseline methods in Section 6.4. We also design several variants of the proposed method to verify the contributions of each part in the model in Section 6.5 and investigate the sensitivity of three hyperparameters in the model in Section 6.6. Finally, we analyze the distribution of compositional codes to evaluate if a good code matrix is learned.

6.1 Experimental Settings

6.1.1 Datasets. We evaluate our model on two real-world benchmark datasets: Tmall\(^2\) and RetailRocket\(^3\). Tmall is from IJCAI-15 competition and contains anonymized users’ shopping logs on Tmall shopping platform. RetailRocket is a dataset on a Kaggle contest published by an E-commerce company, including the user’s browsing activities within 6 months. For convenient comparison, we duplicate the experimental environment in [49, 50]. Specifically, we filter out all sessions whose length is 1 and items appearing less than five times. The latest interacted item of each session is assigned to the test set and the previous data is used for training. The validation set is randomly sampled from the training set and makes up 10% of the training set. Then, we augment and label the training and test datasets by using a sequence splitting method, which generates multiple labeled sequences with the corresponding labels \((v_{s,1}, v_{s,2}), (v_{s,1}, v_{s,2}, v_{s,3}), \ldots, (v_{s,1}, v_{s,2}, \ldots, v_{s,l-1}, v_{s,l})\) for every session \(s = [v_{s,1}, v_{s,2}, v_{s,3}, \ldots, v_{s,l}]\). Note that the label of each sequence is the last consumed item in it. The statistics of used datasets are presented in Table 2.

| Dataset      | #Training sessions | #Test sessions | #Items | Avg. Length |
|--------------|--------------------|----------------|--------|-------------|
| Tmall        | 351,268            | 25,898         | 40,728 | 6.69        |
| RetailRocket | 433,643            | 15,132         | 36,968 | 5.43        |

6.1.2 Baseline Methods. We compare our method with the following representative session-based recommendation methods (since our work is the first on-device session-based recommendation model, we only compare the new method with its predecessor OD-Rec at the on-device level):

- **GRU4Rec** [14] is a GRU-based session-based recommendation model which also utilizes a session-parallel mini-batch training process and adopts ranking-based loss functions to model user sequences.
- **NARM** [25] is an RNN-based model which employs an attention mechanism to capture users’ main purpose and combines it with the temporal information to generate recommendations.
- **STAMP** [28] adopts attention layers to replace all RNN encoders and employs the self-attention mechanism [47] to model long-term and short-term user interests.
- **SR-GNN** [50] proposes a gated graph neural network to refine item embeddings and also employs a soft-attention mechanism to compute the session embeddings.
- **SASRec** [21] is a Transformer-based sequential recommendation model to capture long-term semantics and predict based on relatively few actions.
- **OD-Rec** [54] is our previous work that proposes a session-based recommendation model operating on resource-constrained devices where a TTD method is used to compress the deep model and a self-supervised KD framework is proposed to retain model capacity.

\(^2\)https://tianchi.aliyun.com/dataset/dataDetail?dataId=42.
\(^3\)https://www.kaggle.com/retailrocket/ecommerce-dataset.
Table 3. Performance Comparison for All the Methods

| Method    | Tmall Prec@5 | Tmall NDCG@5 | RetailRocket Prec@5 | RetailRocket NDCG@5 |
|-----------|--------------|--------------|---------------------|---------------------|
| GRU4REC   | 16.48        | 9.25         | 32.37               | 19.36               |
| NARM      | 16.84        | 10.96        | 32.94               | 20.07               |
| STAMP     | 17.45        | 12.92        | 33.57               | 21.53               |
| SR-GNN    | 19.39        | 14.29        | 35.68               | 27.19               |
| SASRec    | 20.23        | 15.11        | 36.37               | 27.54               |
| Teacher   | 22.47        | 18.05        | 36.51               | 27.77               |
| OD-Rec    | 23.68        | 18.86        | 35.17               | 28.45               |
| Student   | 23.66        | 18.49        | 37.72               | 28.60               |

We use P@K (Precision) and NDCG@K (Normalized Discounted Cumulative Gain) to evaluate the recommendation results where K is 5 or 10. Precision measures the ratio of hit items and NDCG assesses the ranking quality of the recommendation list.

6.1.3 Hyperparameter Settings. As for the setting of the general hyperparameters, we set the mini-batch size to 100, the $L_2$ regularization to $10^{-5}$, and the embedding dimension to 128 for Tmall and 256 for RetailRocket. All the learnable parameters are initialized with the Uniform Distribution $U(-0.1, 0.1)$. For the backbone, the best performance is achieved when the number of attention layers and attention heads are 1 and 2 for RetailRocket and 1 and 1 for Tmall. The dropout rate is 0.5 for Tmall and 0.2 for RetailRocket. We use Adam with learning rate 0.001 to optimize the model. We empirically set the temperature in Gumbel-Softmax to 0.3. For all baselines, we report their best performances with the same general experimental settings.

6.2 Model Performance

We first report the performances of all methods in Table 3. We use Teacher to denote the separately trained server-side model and use Student to denote the on-device model trained with the bidirectional self-supervised KD framework. OD-Rec is the predecessor of the student model proposed in our previous work. The performance of the on-device model varies drastically with different hyperparameters. We present the results of OD-Rec and the student when they are compressed to the same degree (compression ratio is 30). According to the table, we can draw the following conclusions:

- Transformer-based models (SASRec, the teacher model, and two student models) outperform all other deep models (RNN-based methods and graph-based methods), showing Transformer-based architecture’s superiority in modeling session-based data. In addition, the teacher model and two student models have higher performances than SASRec, verifying that the soft attention mechanism we use in the base model helps predict accurate recommendation lists.
- OD-Rec can outperform the teacher on Prec@5 on Tmall and NDCG@5 on two datasets with 30× smaller size. The new student model is further strengthened so that it can even outperform the teacher on almost all the metrics and datasets. It also significantly beats OD-Rec especially on Prec@10 on RetailRocket, which demonstrates the effectiveness of the proposed compositional encoding and bidirectional self-supervised KD framework.

6.3 Effectiveness of Compressed Model

In our method, the model compression rate is influenced by two factors, i.e., the number of codebooks $M$ and the number of vectors in each codebook $K$. According to the compression ratio
Table 4. Performance Comparison between Different $M$ and $K$ on Tmall

| $M$ | $K$ | CR   | Prec@5 | NDCG@5 | Prec@10 | NDCG@10 |
|-----|-----|------|--------|--------|---------|---------|
| 1   | 32  | 116  | 21.10  | 15.58  | 25.48   | 17.01   |
| 1   | 128 | 22.01| 15.48  | 26.22  | 17.87   |         |
| 1   | 256 | 22.77| 17.13  | 27.02  | 18.16   |         |
| 1   | 512 | 23.10| 17.76  | 27.38  | 18.22   |         |
| 2   | 32  | 58   | 23.01  | 17.58  | 27.37   | 19.00   |
| 2   | 128 | 23.18| 17.63  | 27.57  | 19.05   |         |
| 2   | 256 | 23.21| 18.70  | 27.53  | 19.12   |         |
| 2   | 512 | 23.45| 18.65  | 27.92  | 19.79   |         |
| 3   | 32  | 39   | 23.05  | 17.67  | 27.36   | 19.07   |
| 3   | 128 | 23.44| 18.43  | 27.42  | 19.66   |         |
| 3   | 256 | 23.75| 18.63  | 27.66  | 19.75   |         |
| 3   | 512 | 23.82| 18.89  | 28.00  | 19.84   |         |
| 4   | 32  | 29   | 23.66  | 18.49  | 27.76   | 19.67   |
| 4   | 128 | 23.88| 18.45  | 27.33  | 19.30   |         |
| 4   | 256 | 23.82| 18.50  | 27.86  | 19.98   |         |
| 4   | 512 | 23.89| 18.61  | 28.04  | 19.64   |         |
| 5   | 32  | 23   | 23.57  | 18.42  | 27.49   | 19.07   |
| 5   | 128 | 23.67| 18.53  | 27.41  | 18.95   |         |
| 5   | 256 | 23.88| 18.67  | 27.44  | 19.06   |         |
| 5   | 512 | 23.81| 18.89  | 28.13  | 19.70   |         |

From the results, we can observe that when the value of $M$ increases, the compression ratio tends to decrease, and the performances are gradually becoming better in most cases. Similarly, when $M$ is fixed, with the increase of $K$, there is a trend toward better recommendation accuracy. For the two datasets, both $M$ and $K$ are vital to the model’s capacity. There is a size-accuracy tradeoff when choosing different values of $M$ and $K$.

We highlight the performance in bold when the compression ratio is 29 on Tmall where $M$ is 4 and $K$ is 32. Compared with the performance of OD-Rec where the compression ratio is 27, the highlighted performance is higher on Prec@10 and NDCG@10 and comparable on Prec@5 and NDCG@5. On the dataset of RetailRocket, we highlight the results in bold when the compression rate is 31 where $M$ = 3 and $K$ = 256. Compared with the results of OD-Rec in Table 3 on RetailRocket which are from the model with 30× smaller size than the uncompressed one, the highlighted performance of the new method is superior on all metrics. These observations demonstrate that when the previous method OD-Rec and the new method are compressed to the same degree, the new method has distinct advantages in recommendation accuracy, which also corroborates the effectiveness of the technical changes we have made on OD-Rec.

6.4 Efficiency Analysis

The fast local inference is an essential feature of on-device models. To demonstrate the efficiency superiority of the proposed method, we test the inference time of the new method and all the baselines including its predecessor OD-Rec on the same resource-constrained device. We first calculate the equation (Equation (11)), it is obvious that the value of $M$ has a larger influence than $K$ in theory. To investigate the effectiveness of compressed models with different compression ratios and different $M$, $K$, we conduct a comprehensive study by selecting some representative values for the two factors. To cover a wide range of compression ratios, we select $\{1, 2, 3, 4, 5\}$ for $M$ and $\{32, 128, 256, 512\}$ for $K$. We report their corresponding compression ratios and performances in Table 4. From the results, we can observe that when the value of $M$ increases, the compression ratio tends to decrease, and the performances are gradually becoming better in most cases. Similarly, when $M$ is fixed, with the increase of $K$, there is a trend toward better recommendation accuracy. For the two datasets, both $M$ and $K$ are vital to the model’s capacity. There is a size-accuracy tradeoff when choosing different values of $M$ and $K$.

We highlight the performance in bold when the compression ratio is 29 on Tmall where $M$ is 4 and $K$ is 32. Compared with the performance of OD-Rec where the compression ratio is 27, the highlighted performance is higher on Prec@10 and NDCG@10 and comparable on Prec@5 and NDCG@5. On the dataset of RetailRocket, we highlight the results in bold when the compression rate is 31 where $M$ = 3 and $K$ = 256. Compared with the results of OD-Rec in Table 3 on RetailRocket which are from the model with 30× smaller size than the uncompressed one, the highlighted performance of the new method is superior on all metrics. These observations demonstrate that when the previous method OD-Rec and the new method are compressed to the same degree, the new method has distinct advantages in recommendation accuracy, which also corroborates the effectiveness of the technical changes we have made on OD-Rec.
Table 5. Prediction Time (s) on Device

|          | Time (s) GRU4Rec | NARM | STAMP | SR-GNN | OD-Rec | New Student |
|----------|------------------|------|-------|--------|--------|-------------|
| Tmall    | 0.417            | 0.264| 0.175 | 0.310  | 0.169  | 0.021       |
| RetailRocket | 0.093          | 0.096| 0.094 | 0.192  | 0.081  | 0.012       |

Fig. 3. Ablation study.

Train all the models on GPU, and then they are encapsulated by PyTorch Mobile. We use Android Studio 11.0.11 to simulate a virtual device environment and deploy all the models under this virtual environment to do the inference. The selected device system is Google Pixel 2 API 29. We record the prediction time for every 100 sessions of each model on the two datasets and then report the average time in Table 5. Obviously, our student model is much faster than all the baselines on the two datasets. RNN-like units in GRU4Rec and NARM and the graph construction in SR-GNN leads to their prolonged inference. In particular, comparing with OD-Rec, our student model is 8× faster on Tmall and 6× faster on RetailRocket. We attribute this success to the fact that the code matrix of items can be directly saved on the device. When doing inference, the new student model just needs to multiply it with the codebooks to get item embeddings. The code matrix consumes very limited memory because $M$ is rather small. In comparison, when using TTD in OD-Rec, even if the index lists can be saved, there is still a series of multiplication between indexed vectors. This efficiency analysis proves that the integration of compositional code can successfully address the computation bottleneck and reduce inference time.

6.5 Ablation Study

The superior performance of the new on-device model has demonstrated the effectiveness of the overall method design. To investigate the contribution of each part of the proposed method, we devise five variants, i.e., Stu-base, Stu-w/o-c, Stu-w/o-b, Stu-w/o-s, and Stu-w/o-m. Stu-base represents the compressed model without any KD; Stu-w/o-c represents the one where the contrastive learning task is detached; Stu-w/o-b means the knowledge transfer from the student to the teacher is disabled, where the parameters of the teacher model are frozen; and Stu-w/o-s means we dispense with the soft targets distillation task; Stu-w/o-m denotes the version where the embedding mixup proposed in Equation (16) is disabled. We present their results in Figure 3. Comparing Stu-base with the full version of the student, we find that the bidirectional self-supervised KD framework improves the model by 10.55% on Pre@10 and 10.20% on NDCG@10 on Tmall and 8.29% on Pre@10 and 10.90% on NDCG@10 on RetailRocket. The two distillation tasks and the bidirectional transfer mechanism in the KD framework all play a part in these performance gains. In addition, it should be noted that the embedding mixup is critical. Without it, the performance drastically drops, which demonstrates that the direct knowledge transfer at the embedding level is the most effective.
6.6 Hyperparameter Analysis

In our method, there are three important hyperparameters: $\beta$, $\gamma$, and $\eta$. The first two control the effect of the two distillation tasks where $\beta$ is for the contrastive distillation task and $\gamma$ is for the soft target distillation task. $\eta \in (0, 1)$ determines how much information from the teacher is used in the embedding mixup (Equation (16)).

We first investigate the model’s sensitivity to $\beta$ and $\gamma$ on two datasets. A large number of hyperparameter combinations are tested and we choose to report some representative ones of the two hyperparameters; they are \{0.1, 0.3, 0.5, 0.7, 0.9\} for $\gamma$ and \{0.005, 0.01, 0.05, 0.1, 0.2, 0.5\} for $\beta$. The corresponding results are presented in Figure 4. When investigating the former two hyperparameters, we fix $\eta$. As can be observed from Figure 4, although each curve can have more than one rise and fall when tuning $\gamma$, it shows a relatively stable tendency: first it goes up to the top and then gradually declines. Specifically, $\gamma = 0.3$ is the best choice for the model on both the datasets. Meanwhile, the best performance is reached when $\beta = 0.01$ on Tmall and $\beta = 0.5$ on RetailRocket. We also notice that the curves of Precision and NDCG are not always consistent in where the best performance is reached.

In Figure 5, we report the results brought by different $\eta$, ranging from 0.1 to 0.9 with the step 0.1. When investigating $\eta$, we fix $\beta$ and $\gamma$. According to Figure 5, a larger $\eta$ can lead to better performance on both the datasets ($\eta = 0.8$ on Tmall and $\eta = 0.7$ on RetailRocket), which means a great deal of embedding information transfer from the teacher is beneficial. But when $\eta$ is 0.9, the model performance declines drastically. We conjecture that when $\eta$ is overlarge, the gradients for updating the student’s parameters could be too small, resulting in underfitted parameters.

6.7 Analysis on Distribution of Compositional Codes

The item embedding in our on-device model is the sum of vectors looked up from $M$ codebooks by using the compositional code. To investigate the distribution of compositional codes of all items, i.e., how many times each code vector is indexed or mapped, we record the code matrix generated by the Gumbel-Softmax operation when the model reaches its best performance on two datasets. We then calculate the total times that each vector is hit and display these numbers with heatmaps.
Fig. 6. Heatmap of codebook vectors on two datasets.

Table 6. Performances of Different Baseline Models

| Datasets       | Model   | Prec@5 | NDCG@5 | Prec@10 | NDCG@10 |
|----------------|---------|--------|--------|---------|---------|
| Tmall          | NARM    | 16.84  | 10.96  | 19.68   | 11.66   |
|                | NARM†   | 21.19  | 16.79  | 24.68   | 17.92   |
|                | SR-GNN  | 19.39  | 14.29  | 23.79   | 15.81   |
|                | SR-GNN† | 21.61  | 17.14  | 24.86   | 18.20   |
| RetailRocket   | NARM    | 32.94  | 20.07  | 39.18   | 21.53   |
|                | NARM†   | 34.93  | 27.39  | 41.43   | 29.52   |
|                | SR-GNN  | 35.68  | 27.19  | 43.31   | 29.67   |
|                | SR-GNN† | 36.50  | 27.87  | 43.99   | 30.29   |

in Figure 6, where each grid represents a single vector and each row represents vectors from the same codebook. For a clear and convenient display, we choose to present the distribution when \( M = 4 \) and \( K = 32 \) for both datasets. As shown in the figure, we can see that the distribution of compositional codes of items is uniformly relatively. The numbers fall in the intervals of \([1090,1480]\) on Tmall and \([997,1312]\) on RetailRocket. These distributions make sense because items have different characteristics and a good encoding module should map them to different codes in order to reflect the difference. Meanwhile, some items have common characteristics and their codes should have an overlap which results in some lighter grids. Note that the values in the heatmap of Tmall are larger than those in that of RetailRocket. This is because the dataset of Tmall contains more items. The item number equals to a quarter \( \left( \frac{1}{4} \right) \) of the sum of the values in the heatmap.

6.8 Generalization Ability Analysis

In this article, our proposed model compression method and bidirectional self-supervised KD framework are orthogonal to session-based recommendation models. To investigate the generalization ability, we conduct experiments on two other representative baseline models, NARM [25] and SR-GNN [50], which are based on RNN and GNN, respectively. We present their results where we set \( M = 4, K = 32 \) for Tmall and \( M = 3, K = 256 \) for RetailRocket. NARM† and SR-GNN† represents the corresponding new models. Results are shown in Table 6. Compared with the results in Table 3, we can directly know that NARM and SR-GNN are both enhanced, showing that our proposed method can consistently improve a wide spectrum of backbones for session-based recommendation.
7 FUTURE WORK AND CONCLUSION

In recent years, on-device session-based recommendation systems have been achieving increasing attention on account of the low energy/resource consumption and privacy protection. The constrained memory and computation resource in mobile devices has spurred a series of research for making state-of-the-art deep recommendation models fit in resource-constrained devices. Techniques such as quantization, low-rank decomposition, and network pruning have been utilized to compress the regular models and achieved promising performance. However, the high compression rate often comes at the cost of slow local inference because of the cumbersome operations to recover the parameters for prediction. In this article, we propose a new model compression method based on compositional encoding to accelerate the embedding reconstruction for faster local inference. Experimental results show that the new method can achieve $6 \times -8 \times$ faster inference and meanwhile exhibits better recommendation performance with a large compression ratio.

In addition to reducing the model size to fit in mobile devices, how to regularly update the on-device model is another problem. Retraining from scratch can be costly and energy-consuming. Recently, a few studies [63] propose the module grafting, which allows the original parameters to remain unchanged and inserts retrained small blocks with a small number of parameters into the original model to finish model updates. This could be a promising approach for conveniently updating on-device recommendation models. However, the major parameter to be updated in our situation is the item embedding table, which means this technique cannot be directly applied. Therefore, in our future work, we expect to target efficient model updates for on-device recommendation models.

REFERENCES

[1] Benu Madhab Changmai, Divija Nagaraju, Debi Prasanna Mohanty, Kriti Singh, Kunal Bansal, and Sukumar Moharana. 2019. On-device user intent prediction for context and sequence aware recommendation. arXiv:1909.12756.

[2] Liyang Chen, Yongquan Chen, Junfong Xi, and Xinyi Le. 2021. Knowledge from the original network: Restore a better pruned network with knowledge distillation. Complex & Intelligent Systems 8 (2022), 709–718.

[3] Ting Chen, Martin Renqiang Min, and Yizhou Sun. 2018. Learning k-way d-dimensional discrete codes for compact embedding representations. In Proceedings of the International Conference on Machine Learning. PMLR, 854–863.

[4] Tong Chen, Hongzhi Yin, Quoc Viet Hung Nguyen, Wen-Chih Peng, Xue Li, and Xiaofang Zhou. 2020. Sequence-aware factorization machines for temporal predictive analytics. In Proceedings of the 2020 IEEE 56th International Conference on Data Engineering (ICDE’20). IEEE, 1405–1416.

[5] Tong Chen, Hongzhi Yin, Yujia Zheng, Zhi Huang, Yang Wang, and Meng Wang. 2021. Learning elastic embeddings for customizing on-device recommenders. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. 138–147.

[6] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. 2015. Compressing neural networks with the hashing trick. In Proceedings of the International Conference on Machine Learning. PMLR, 2285–2294.

[7] Xusong Chen, Dong Liu, Chenyi Lei, Rui Li, Zheng-Jun Zha, and Zhiwei Xiong. 2019. BERT4SessRec: Content-based video relevance prediction with bidirectional encoder representations from transformer. In Proceedings of the 27th ACM International Conference on Multimedia. 2597–2601.

[8] Gabriel de Souza Pereira Moreira, Sara Rabhi, Jeong Min Lee, Ronay Ak, and Even Oldridge. 2021. Transformers4Rec: Bridging the gap between NLP and sequential/session-based recommendation. In Proceedings of the 15th ACM Conference on Recommender Systems. 143–153.

[9] Sauptik Dhar, Junyao Guo, Jiayi Liu, Samarth Tripathi, Unmesh Kurup, and Mohak Shah. 2021. A survey of on-device machine learning: An algorithms and learning theory perspective. ACM Transactions on Internet of Things 2, 3 (2021), 1–49.

[10] Simone Disabato and Manuel Roveri. 2020. Incremental on-device tiny machine learning. In Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things. 7–13.

[11] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization. arXiv:1412.6115.

[12] Jiali Zhang Han, Yun Ma, Qiaozhu Mei, and Xuanzhe Liu. 2021. DeepRec: On-device deep learning for privacy-preserving sequential recommendation in mobile commerce. In Proceedings of the Web Conference 2021. 900–911.
[13] Song Han, Huizi Mao, and William J. Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman compression. arXiv:1510.00149.

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

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

[16] Oleksii Hrinchuk, Valentin Khrukov, Leyla Mirvakhabova, Elena Orlova, and Ivan Oseledets. 2019. Tensorized embedding layers for efficient model compression. arXiv:1901.10787.

[17] David A. Huffman. 1952. A method for the construction of minimum-redundancy codes. Proceedings of the IRE 40, 9 (1952), 1098–1101.

[18] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical reparameterization with Gumbel-softmax. In Conference Track Proceedings of the 5th International Conference on Learning Representations (ICLR’17).

[19] SeongKu Kang, Junyoung Hwang, Wonbin Kweon, and Hwanjo Yu. 2020. DE-RRD: A knowledge distillation framework for recommender system. In Proceedings of the 29th ACM International Conference on Information & Knowledge Management. 605–614.

[20] Wang-Cheng Kang, Derek Zhiyuan Cheng, Tiansheng Yao, Xinyang Yi, Ting Chen, Lichan Hong, and Ed H. Chi. 2020. Learning to embed categorical features without embedding tables for recommendation. arXiv:2010.10784.

[21] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentional sequential recommendation. In Proceedings of the 2018 IEEE International Conference on Data Mining (ICDM’18). IEEE, 197–206.

[22] Wonbin Kweon, SeongKu Kang, and Hwanjo Yu. 2021. Bidirectional distillation for top-K recommender system. In Proceedings of the Web Conference 2021. 3861–3871.

[23] Juhyun Lee, Nikolay Chirkov, Ekaterina Ignasheva, Yury Pisarchyk, Mogan Schieh, Fabio Riccardi, Raman Sarokin, Andrei Kulik, and Matthias Grundmann. 2019. On-device neural net inference with mobile GPUs. arXiv:1907.01989.

[24] Jae-woong Lee, Minjin Choi, Jongwuk Lee, and Hyunjung Shim. 2019. Collaborative distillation for top-N recommendation. In Proceedings of the 2019 IEEE International Conference on Data Mining (ICDM’19). IEEE, 369–378.

[25] Jing Li, Pengjie Ren, Zhumin Chen, Zhaocun Ren, Tao Lian, and Jun Ma. 2017. Neural attentive session-based recommendation. In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. 1419–1428.

[26] Yang Li, Tong Chen, Peng-Fei Zhang, and Hongzhi Yin. 2021. Lightweight self-attentional sequential recommendation. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management. 967–977.

[27] Chenghao Liu, Tao Lu, Xin Wang, Zhiyong Cheng, Jianling Sun, and Steven C. H. Hoi. 2019. Compositional coding for collaborative filtering. In Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval. 145–154.

[28] Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. 2018. STAMP: Short-term attention/memory priority model for session-based recommendation. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 1831–1839.

[29] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Zhaoyu Wang, Li Mian, Jing Zhang, and Jie Tang. 2020. Self-supervised learning: Generative or contrastive. IEEE Transactions on Knowledge and Data Engineering 35, 1 (2023), 857–876.

[30] Muyang Ma, Pengjie Ren, Zhumin Chen, Zhaocun Ren, Huasheng Liang, Jun Ma, and Maarten de Rijke. 2021. Improving transformer-based sequential recommenders through preference editing. arXiv:2106.12120.

[31] Matthew S. Meselson and Charles M. Radding. 1975. A general model for genetic recombination. Proceedings of the National Academy of Sciences U S A 72, 1 (1975), 358–361.

[32] Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. 2015. Tensorizing neural networks. arXiv:1509.06569.

[33] Keiichi Ochiai, Kohei Senkawa, Naoki Yamamoto, Yuya Tanaka, and Yusupe Fukazawa. 2019. Real-time on-device troubleshooting recommendation for smartphones. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2783–2791.

[34] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. arXiv:1807.03748.

[35] Ivan V. Oseledets. 2011. Tensor-train decomposition. SIAM Journal on Scientific Computing 33, 5 (2011), 2295–2317.

[36] Zhiqiang Pan, Fei Cai, Yanxiang Ling, and Maarten de Rijke. 2020. Rethinking item importance in session-based recommendation. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 1837–1840.

[37] Bryan A. Plummer, Nikoli Dryden, Julius Frost, Torsten Hoefler, and Kate Saenko. 2020. Shapeshifter networks: Cross-layer parameter sharing for scalable and effective deep learning. arXiv–2006.

[38] Ruihong Qiu, Hongzhi Yin, Zi Huang, and Tong Chen. 2020. Gag: Global attributed graph neural network for streaming session-based recommendation. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 669–678.
[39] Guy Shani, David Heckerman, and Ronen I. Brafman. 2005. An MDP-based recommender system. *Journal of Machine Learning Research* 6, (Sept. 2005), 1265–1295.

[40] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. 2020. Compositional embeddings using complementary partitions for memory-efficient recommender systems. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 165–175.

[41] Raphael Shu and Hideki Nakayama. 2017. Compressing word embeddings via deep compositional code learning. arXiv:1711.01068.

[42] Suraj Srinivas and R. Venkatesh Babu. 2015. Data-free parameter pruning for deep neural networks. arXiv:1507.06149.

[43] Yang Sun, Fajie Yuan, Min Yang, Guoao Wei, Zhou Zhao, and Duo Liu. 2020. A generic network compression framework for sequential recommender systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1299–1308.

[44] Jiaxi Tang and Ke Wang. 2018. Ranking distillation: Learning compact ranking models with high performance for recommender system. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2289–2298.

[45] Dan Tito Svenstrup, Jonas Hansen, and Ole Winther. 2017. Hash embeddings for efficient word representations. *Advances in Neural Information Processing Systems* 30 (2017).

[46] Huynh Thanh Trung, Tong Van Vinh, Nguyen Thanh Tam, Hongzhi Yin, Matthias Weidlich, and Nguyen Quoc Viet Hung. 2020. Adaptive network alignment with unsupervised and multi-order convolutional networks. In *Proceedings of the 2020 IEEE 36th International Conference on Data Engineering (ICDE’20)*. 85–96.

[47] 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 *Advances in Neural Information Processing Systems*. 5998–6008.

[48] 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 *Proceedings of the Web Conference 2020*. 906–916.

[49] Ziyang Wang, Wei Wei, Gao Cong, Xiao-Li Li, Xian-Ling Mao, and Minghui Qiu. 2020. Global context enhanced graph neural networks for session-based recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 169–178.

[50] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. 2019. Session-based recommendation with graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 346–353.

[51] Xiaorui Wu, Hong Xu, Honglin Zhang, Huaming Chen, and Jian Wang. 2020. Saec: Similarity-aware embedding compression in recommendation systems. In *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems*. 82–89.

[52] Xin Xia, Hongzhi Yin, Junliang Yu, Yingxia Shao, and Lizhen Cui. 2021. Self-supervised graph co-training for session-based recommendation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2180–2190.

[53] Xin Xia, Hongzhi Yin, Junliang Yu, Qinyong Wang, Lizhen Cui, and Xiangliang Zhang. 2021. Self-supervised hypergraph convolutional networks for session-based recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 4503–4511.

[54] 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 *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 546–555.

[55] Chengfeng Xu, Pengpeng Zhao, Yanchi Liu, Victor S. Sheng, Jiajie Xu, Fuqin Zhuang, Junhua Fang, and Xiaofang Zhou. 2019. Graph contextualized self-attention network for session-based recommendation. In *IJCAI*, Vol. 19. 3940–3946.

[56] Chunxing Yin, Bilge Acun, Carole-Jean Wu, and Xing Liu. 2021. TT-Rec: Tensor train compression for deep learning recommendation models. *Proceedings of Machine Learning and Systems* 3 (2021).

[57] Hongzhi Yin and Bin Cui. 2016. *Spatio-Temporal Recommendation in Social Media*. Springer.

[58] Junliang Yu, Xin Xia, Tong Chen, Lizhen Cui, Nguyen Quoc Viet Hung, and Hongzhi Yin. 2022. XSimGCL: Towards extremely simple graph contrastive learning for recommendation. arXiv:2209.02544.

[59] Junliang Yu, Hongzhi Yin, Min Gao, Xin Xia, Xiangliang Zhang, and Nguyen Quoc Viet Hung. 2021. Socially-aware self-supervised tri-training for recommendation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2084–2092.

[60] Junliang Yu, Hongzhi Yin, Jundong Li, Qinyong Wang, Nguyen Quoc Viet Hung, and Xiangliang Zhang. 2021. Self-supervised multi-channel hypergraph convolutional network for social recommendation. In *Proceedings of the Web Conference 2021*. 413–424.

[61] Junliang Yu, Hongzhi Yin, Xin Xia, Tong Chen, Lizhen Cui, and Quoc Viet Hung Nguyen. 2022. Are graph augmentations necessary? Simple graph contrastive learning for recommendation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1294–1303.
[62] Junliang Yu, Hongzhi Yin, Xin Xia, Tong Chen, Jundong Li, and Zi Huang. 2022. Self-supervised learning for recommender systems: A survey. arXiv:2203.15876.

[63] Fajie Yuan, Xiangnan He, Alexandros Karatzoglou, and Liguang Zhang. 2020. Parameter-efficient transfer from sequential behaviors for user modeling and recommendation. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 1469–1478.

[64] Yan Zhang, Hongzhi Yin, Zi Huang, Xingzhong Du, Guowu Yang, and Defu Lian. 2018. Discrete deep learning for fast content-aware recommendation. In Proceedings of the 11th ACM International Conference on Web Search and Data Mining. 717–726.

[65] Hengling Zhao, Yipeng Liu, Xiaolin Huang, and Ce Zhu. 2021. Semi-tensor product-based TensorDecomposition for neural network compression. arXiv:2109.15200.

Received 2 October 2022; revised 8 December 2022; accepted 5 January 2023