On-Device Next-Item Recommendation with Self-Supervised Knowledge Distillation

Xin Xia  
The University of Queensland  
Brisbane, Australia  
x.xia@uq.edu.au

Hongzhi Yin∗

The University of Queensland  
Brisbane, Australia  
h.yin1@uq.edu.au

Junliang Yu

The University of Queensland  
Brisbane, Australia  
jl.yu@uq.edu.au

Qinyong Wang

Baidu Inc.  
Beijing, China  
wangqinyong@baidu.com

Guandong Xu

University of Technology Sydney  
Sydney, Australia  
Guandong.Xu@uts.edu.au

Quoc Viet Hung Nguyen

Griffith University  
Gold Coast, Australia  
quocviethung1@gmail.com

ABSTRACT

Modern recommender systems operate in a fully server-based fashion. To cater to millions of users, the frequent model maintaining and the high-speed processing for concurrent user requests are required, which comes at the cost of a huge carbon footprint. Meanwhile, users need to upload their behavior data even including the immediate environmental context to the server, raising the public concern about privacy. On-device recommender systems circumvent these two issues with cost-conscious settings and local inference. However, due to the limited memory and computing resources, on-device recommender systems are confronted with two fundamental challenges: (1) how to reduce the size of regular models to fit edge devices? (2) how to retain the original capacity?

Previous research mostly adopts tensor decomposition techniques to compress regular recommendation models with low compression rates so as to avoid drastic performance degradation. In this paper, we explore ultra-compact models for next-item recommendation, by loosing the constraint of dimensionality consistency in tensor decomposition. To compensate for the capacity loss caused by compression, we develop a self-supervised knowledge distillation framework which enables the compressed model (student) to distill the essential information lying in the raw data, and improves the long-tail item recommendation through an embedding-recombination strategy with the original model (teacher). The extensive experiments on two benchmarks demonstrate that, with 30x size reduction, the compressed model almost comes with no accuracy loss, and even outperforms its uncompressed counterpart. The code is released at https://github.com/xiaxin1998/OD-Rec.

CCS CONCEPTS

• Information systems  →  Recommender systems.

∗Corresponding author.

KEYWORDS

Next-item Recommendation, Self-supervised Learning, Knowledge Distillation, On-device Learning

ACM Reference Format:

Xin Xia, Hongzhi Yin, Junliang Yu, Qinyong Wang, Guandong Xu, and Nguyen Quoc Viet Hung. 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 (SIGIR ’22), July 11–15, 2022, Madrid, Spain. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3477495.3531775

1 INTRODUCTION

Benefiting from the huge model size and powerful neural architectures, deep learning-based recommender systems show unparalleled capacity in mining users’ latent interests [52]. However, these systems are fully constructed at the server side and rely on abundant storage, memory and computing resources in the cloud. Despite their effectiveness, running these systems is at the cost of a huge carbon footprint [11]. The frequent training and update of recommendation models and the high-speed processing for millions of concurrent user request are driven by a large number of energy-consuming CPUs/GPUs, requiring a vast amount of electric power. Meanwhile, deep recommendation models are fueled by massive user behavior data. When serving users, these cloud-based recommender systems even ask for the immediate contextual data for real-time inference, raising public concern about privacy [14].

Recently, on-device machine learning [6] has attracted increasing attention, which enables models to run on personal devices. In this learning paradigm, the machine learning model is first trained on the cloud, and then is downloaded and deployed on local energy-saving devices such as smartphones. When using the model, users no longer need to upload the sensitive data to servers, and therefore can enjoy low-latency services that are orders of magnitude faster than server-side models [17]. 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 towards on-device recommendation [1, 4, 8, 26, 39, 46]. However, due to the limited memory and computing power, running a cloud-side recommendation model overloads the edge devices [6]. We have to compress the model to fit resource-constrained environments. Therefore, it is non-trivial to develop practicable...
on-device recommender systems, which are confronted with two fundamental challenges.

The first challenge is how to retain the original capacity? Unlike the vision and language models [9] 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, since the recommender systems need to uniquely identify millions of items and users, the embedding table is the one that accounts for the vast majority of memory use [42]. Classical model compression techniques such as pruning [33], down-sizing [12], and parameter sharing [29] hence are less suited, leading to negligible reduced sizes. Previous practices of on-device recommendation [34, 39] mostly adopt decomposition techniques such as tensor-train decomposition (TTD) [28] to approximate the embedding table with a series of matrix multiplication. However, to avoid drastic performance drop, they only compress the embedding table with a small compression rate (e.g., 4–8x) and hesitate to explore more compact tensor approximations.

The second challenge is how to reduce the size of regular models to adapt to edge devices? Inevitably, after being compressed, the capacity of the lightweight model would seriously degrade. To fix this problem, a typical approach is knowledge distillation [12] whose pipeline is first training a teacher model on the cloud by fully utilizing the abundant resources and then transferring teacher’s knowledge to the light-weight model, which is called student. However, although the existing KD frameworks have been proved beneficial in traditional machine learning problems such as classification and regression, applying it to recommendation is still challenging due to the data sparsity issue [15, 18]. Besides, recent studies find that models with similar structures (e.g., encoder–decoder) are easier to transfer knowledge [2], while the tensor decomposition may widen the structural gap between the teacher and the student because it can be seen as a special linear layer prior to student’s embedding layer.

In this paper, we work towards an ultra-compact and effective on-device recommendation model under a self-supervised knowledge distillation framework. To tackle the first challenge, given the dilemma that the small TT-rank in tensor-train decomposition leads to under-expressive embedding approximations whereas the larger TT-rank sacrifices the model efficiency [37], we introduce the semi-tensor product (STP) operation [5] to tensor-train decomposition for the extreme compression of the embedding table. This operation endows tensor-train decomposition with the flexibility to perform multiplication between tensors with inconsistent ranks, taking into account both effectiveness and efficiency. Meanwhile, to tackle the second challenge, inspired by the genetic recombination [24] in biology, we propose to recombine the embeddings learned by the teacher and the student to perform self-supervised tasks, which can help distill the essential information lying in the teacher’s embeddings, so as to compensate for the accuracy loss caused by the extremely high compression. Particularly, we exchange the long-tail item embeddings learned by the two models to form new embeddings. For those long-tail items with few interactions, this embedding recombination strategy provides them with the opportunities to learn with the direct guidance from the teacher, reducing the impact of the asymmetric teacher-student structure and leading to more expressive embeddings.

To summarize, our contributions are as follows:

- We explore ultra-compact on-device recommendation models by introducing semi-tensor product to tensor-train decomposition for higher compression rates of the embedding table.
- We develop a self-supervised KD framework to compensate for the accuracy loss caused by the high compression rate. For the first time, self-supervised learning is applied to on-device recommendation.
- We validate the effectiveness of the proposed on-device recommendation model in the next-item recommendation scenario. The extensive experiments on two benchmarks demonstrate that, with 30x model size reduction, the compressed model almost comes with no accuracy loss, and even outperforms its uncompressed counterpart in some cases.

2 RELATED WORK

2.1 On-Device Recommendation

On-device machine learning is becoming a trend due to its low latency inference and advantages on privacy protection. This line of research [1, 4, 6, 8, 26] mainly adopts model compressing techniques like pruning [9, 33], quantization [7], and low-rank factorization [25, 28] to compress regular models so as to fit edge devices. Recently, some on-device recommender models emerged, achieving desired performance with a tiny model size. Chen et al. [4] proposed to learn elastic item embeddings to make lightweight models adaptively fit various devices under any memory constraints for on-device recommendation. DeepRec [8] uses model pruning and embedding sparsification techniques to fulfill model training on mobile devices and fine-tunes the model using local user data, reaching comparable performance in sequential recommendation with 10x parameter reduction. Wang et al. [39] proposed LLRec, a lightweight next POI recommendation model based on tensor-train factorization. TT-Rec [46] applies tensor-train decomposition in the embedding layer to compress the model and improves performance with a sampled Gaussian distribution for the weight initialization of the tensor cores.

2.2 Knowledge Distillation for Recommendation

Knowledge Distillation (KD) [12] is a typical approach to transfer knowledge from a well-trained teacher model to a simple student model. In this framework, the student model is optimized towards 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 [15, 18, 35]. The first work is Ranking Distillation [35], which chooses top-K items from 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 [18] proposes 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 [15] 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. There also have been some works [19, 36] that combine self-supervised learning and knowledge distillation. But they are mainly designed for vision tasks and cannot be easily tailored for recommendation.

2.3 Session-Based Next-Item Recommendation

The early session-based recommendation [31, 32, 47] is based on Markov Chain and mainly focus on modeling temporal order between items. In the deep learning era, networks such as RNNs [3, 10, 53] are modified to model session data and handle the temporal shifts within sessions. The follow-up models, NARM [20] and STAMP [21] employ attention mechanism to give items different priorities when profiling user’s main interests. Graph-based methods [30, 41] construct various types of session graphs to model item relations. For example, SR-GNN [41] constructs session graphs for every session and designs a gated graph neural network to aggregate information between items into session representations. GCE-GNN [40] proposes to capture both global-level and session-level interactions and aggregates item information through graph convolution and self-attention mechanism. Xia et al. [44, 45] proposed to integrate self-supervised learning into session-based recommendation to boost recommendation performance. These models are of great capacity in generating accurate recommendations but they all rely on sufficient storage and computing resources in the cloud, which cannot run on edge devices like smartphones.

3 PRELIMINARIES

3.1 Next-Item Recommendation Task

In this paper, we apply our on-device recommendation model to the session-based scenario for next-item recommendation. Let $\mathcal{V} = \{v_1, v_2, v_3, ..., v_{|\mathcal{V}|}\}$ denote item set and $s = [v_{s1}, v_{s2}, ..., v_{sl}]$ denote a session sequence. Every session sequence is composed of interacted items in the chronological order from an anonymous user. The task of session-based recommendation is to predict the next item, namely $v_{s{l+1}}$, for the current session. 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 session-based recommendation model is a ranked list $y = [y_1, y_2, y_3, ..., y_{|\mathcal{V}|}]$ 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 highest probabilities in $y$ will be selected as the recommendations.

3.2 Tensor-Train Decomposition

Tensor-train decomposition (TTD) is a typical model compressing technique that can compress a large tensor in a tensor-train (TT) format [28] where each element in the tensor can be computed by a sequence of smaller tensor multiplication. Formally, given a $d$-dimensional tensor $\mathcal{A} \in \mathbb{R}^{N_1 \times N_2 \times \cdots \times N_d}$, each entry indexed by $(i_1, i_2, ..., i_d)$ can be represented in the following TT-format (for conciseness, we use $x$ to refer to $\mathcal{A}(i_1, i_2, ..., i_d)$):

$$
\begin{align*}
x &= \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_d=1}^{R_d} G_1(i_1, r_1) G_2(r_1, i_2, r_2) \cdots G_d(r_d-1, i_d) \\
&= G_1[i_1:] G_2[i_2:] \cdots G_{d-1}[i_{d-1}:] G_d[i_d:] .
\end{align*}
$$

(1)

$\{G_k\}_{k=1}^d$ are called TT-cores and $G_k$ is of the size $R_k \times N_k \times R_k$, where $R_k \in [1, N_k]$, and $R_0 = R_d = 1$. The sequence of $\{R_k\}_{k=0}^d$ is called TT-rank. $G_k[\cdot, i_k, \cdot]$ represents a slice of tensor $G_k$.

If $N_k$ can be further factorized into $I_k \times J_k$, then $G_k \in \mathbb{R}^{R_k \times I_k \times J_k \times R_k}$ can be represented as $G_k^\ast \in \mathbb{R}^{R_k \times I_k \times J_k \times R_k}$. Eq(1) hence can be rewritten as:

$$
\begin{align*}
\mathcal{A}((i_1, j_1), \cdots, (i_d, j_d)) &= G_1^\ast(i_1, j_1) G_2^\ast(i_2, j_2) \cdots G_d^\ast(i_d, j_d) \\
&= G_1^\ast[i_1:] G_2^\ast[i_2:] \cdots G_d^\ast[i_d:] ,
\end{align*}
$$

(2)

where $0 \leq i_k < I_k, 0 \leq j_k < J_k (i_k, j_k$ are integers), $\forall k = 1, \ldots, d$. 

---

*Figure 1: An overview of the proposed framework.*
3.3 Semi-Tensor Product

Semi-tensor product (STP) is a generalization of conventional matrix product [5] and the factor matrices can have arbitrary dimensions. In this paper, we focus on the case when the number of columns in the left matrix is proportional to the number of rows in the right matrix, which is called semi-tensor product. Let \( a \in \mathbb{R}^{1 \times np} \) denotes a row vector and \( b \in \mathbb{R}^p \), then \( a \) can be split into \( p \) equal-size blocks as \( a_1, a_2, \ldots, a_p \), the left semi-tensor product denoted by \( \times \) can be defined as:

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

Then, for two matrices \( A \in \mathbb{R}^{H \times np}, B \in \mathbb{R}^{np \times Q} \), 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 \( h \)-th row in \( A \), \( B(:, q) \) represents \( q \)-th column in \( B \) and \( h = 1, 2, \ldots, H \), \( q = 1, 2, \ldots, Q \).

4 ULTRA-COMPACT ON-DEVICE RECOMMENDATION MODEL

4.1 Basic Model Structure

In this paper, we use SASRec [16] as our base model. SASRec is a famous Transformer-based [38] 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 (shown in Fig 1(a)). 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 correlation. Inside the block, residual connections, the neuron dropout, and the layer normalization are sequentially used. The model can be formulated as:

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

where \( X, P \) are the item embeddings and position embeddings, respectively, \( F \) is the session representation learned via self-attention blocks. FNN represents feed forward network layers and \( \Theta \) aggregates embeddings of all items. In the 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 [40] to generate the session representation, which is computed as:

\[
\alpha_t = f^T \sigma(W_{ij} x_i + W_{jk} x_j + c), \ \theta_s = \sum_{t=1}^{I} \alpha_t x_t,
\]

where \( W_{ij} \in \mathbb{R}^{N \times N} \), \( W_{jk} \in \mathbb{R}^{N \times N} \), \( c \in \mathbb{R}^N \) are learnable parameters, \( x_i \) is the embedding of item \( i \) and \( x_j \) is obtained by averaging the embeddings of items within the session \( s \), i.e. \( x_j = \frac{1}{|s|} \sum_{r \in s} x_r \). \( \sigma \) is sigmoid function. Session representation \( \theta_s \) is represented by aggregating item embeddings while considering their corresponding importances.

\[
\text{Figure 2: Given the embedding table of size 12} \times 8, \text{ after applying TTD, it can be approximated by the multiplication of three small tensors. While after applying STTD, the number of the parameters is further compressed.}
\]

4.2 Model Compression

It is obvious that the item embedding table accounts for the large majority of total learnable parameters. Compressing the embedding table is the key to employ on-device recommendation. Therefore, we first adopt the tensor-train decomposition (TTD) [13, 25] to shrink the size of item embeddings. To factorize the embedding table 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 \( i = (i_1, ..., i_d) \) according to [13], and then get the particular slices of the index in TT-cores to perform matrix multiplication on the slices (Eq. (1)). Benefitting from this decomposition, the model does not need to store the embedding table which consumes large memory. Instead we just store the TT-cores to fulfill an approximation of the entry in the embedding table, i.e., a sequence of small tensor multiplication. Given \( \{G_k\}_{k=1}^{d} \) of size \( \sum_{k=1}^{d} R_{k-1} I_k J_k R_k \), the compression rate is:

\[
\text{rate} = \frac{\prod_{k=1}^{d} I_k J_k}{\sum_{k=1}^{d} R_{k-1} I_k J_k R_k} = \frac{I_1 J_1 R + \sum_{k=2}^{d-1} I_k J_k R_k^2 + I_d J_d R}{R_d R_{d-1}}, \quad (8)
\]

where \( R_0 = R_d = 1 \). For simplicity, we assign the same rank \( R \) to the intermediate TT-cores, namely, \( \{R_k\}_{k=1}^{d-1} = R \) in this paper.

However, conventional tensor-train decomposition 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 [54], we propose to integrate semi-tensor product with tensor-train decomposition where the dimension of TT-cores can be arbitrarily adjusted. The conventional tensor product between TT-cores is replaced with the semi-tensor product as:

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

\[
\text{549}
\]
where \( \{ \hat{G}_k \}_{k=1}^d \) are the core tensors after applying semi-tensor product based tensor-train decomposition (STTD) and \( \hat{G}_1 \in \mathbb{R}^{I_1 \times J_1 \times R} \), \( \{ \hat{G}_k \}_{k=2}^{d-1} \in \mathbb{R}^{I_{k+1} \times J_k \times R} \), \( \hat{G}_d \in \mathbb{R}^{I_1 \times J_1 \times R} \). The compression rate is:

\[
\text{rate} = \frac{\prod_{k=1}^{d} I_k J_k}{I_1 J_1 + \sum_{k=1}^{d-1} I_k J_k R^2 + I_d J_d R^2}.
\]

(10)

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 in any large degree. In Fig.2 we illustrate TTD and STTD to facilitate understanding.

Putting Eq. (8) and Eq. (10) together, we can easily find that when \( k \) is large enough, the compression rate calculated by Eq. (10) approaches \( n^2 \) times the rate derived from Eq. (8), towards an ultra-compact model. We hereby show the model compression ability with different TT-ranks by the example in Table 1. We assume there are 20,000 items and this number is factorized into \( 10 \times 10 \times 25 \times 8 \). The original item embedding dimension is 128 which is factorized into \( 4 \times 4 \times 4 \times 2 \). We calculate the model size before and after applying TTD and STTD with different TT-ranks and \( n = 2 \). As shown in the table, it is clear that, with the TT-rank getting smaller, the number of the parameters is dramatically compressed by several orders of magnitude. Particularly, with the same TT-rank, STTD further shrinks the model to about 1/3~1/4 the size compressed by TTD. Meanwhile, our experiments in section 6 demonstrate that, the further compression supported by STTD will not incur performance drop on the one compressed by TTD.

| Original Size | TT-rank | TTD size rate | STTD size rate |
|--------------|---------|--------------|---------------|
| 2,560,000    | 4       | 2464 1039    | 736 3478      |
|              | 8       | 9408 272     | 2592 988      |
|              | 16      | 36736 70     | 9664 265      |

Table 1: Model Size Analytics with Different TT-ranks in TTD and STTD where \( n = 2 \).

5 SELF-SUPERVISED KNOWLEDGE DISTILLATION

A highly compressed model brings efficiency but inevitably comes at the cost of capacity loss compared with the original. Therefore, in this section we propose a novel self-supervised knowledge distillation framework to restore the capacity. In this framework, the compressed model plays the role of student and is trained with the help of its pre-trained uncompressed counterpart, namely, the teacher. Note that the only difference between the teacher and the student model is that the embedding table of the student is compressed by STTD.

5.1 Architecture

To the best of our knowledge, we are the first to apply self-supervised knowledge distillation to recommendation. Therefore, we first define a general framework of self-supervised knowledge distillation in the scenario of recommendation. Let \( D \) denote the item interaction data, \( D_{soft} \) represent the soft targets from the teacher and \( D_{sdt} \) denote the augmented data in 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:

\[
f \left( M_T, M_S, D, D_{soft}, D_{sdt} \right) \rightarrow M'_S.
\]

(11)

which means that by jointly supervising the student model with the historic interactions \( D \), the teacher’s soft targets \( D_{soft} \), and the self-supervised signals extracted from \( D_{sdt} \), we can finally obtain an improved student model \( M'_S \) that retains the capacity of the regular model. We define the learning objective of the student as:

\[
\mathcal{L} = \alpha \mathcal{L}_{rec} + \beta \mathcal{L}_{kd}. \quad \mathcal{L}_{kd} = \{ L_{kd}^1, L_{kd}^2, \ldots \}.
\]

(12)

\( \mathcal{L}_{rec} \) is the recommendation loss, \( \mathcal{L}_{kd} \) is the KD loss which is composed of multiple tasks. \( \alpha \) and \( \beta \) are hyperparameters to control the magnitudes of different tasks. Architecturally, the proposed framework is model-agnostic so as to boost recommendation models with arbitrary structures. Note that, the teacher model has been well-trained before and will not be updated in this framework. We design three types of distillation tasks: traditional distillation task based on soft targets, contrastive self-supervised distillation task and predictive self-supervised distillation task.

5.2 Embedding Recombination

The essential idea of self-supervised learning [22, 49, 50] 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 [48]. For the long-tail items with few interactions, generating informative augmentations is often intractable. Inspired by the genetic recombination [24] and the preference editing in [23], we come up with the idea to exchange embedding segments from the student and the teacher and recombine them to distill knowledge. Such recombinations fulfil 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. Since the parameters of the teacher are frozen, this strategy will not impact the teacher model. 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 part is 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 separately: 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 Eq. (7), while the cold session representation is analogously learned from representations of cold items. They are formulated as follows:

\[
g_{s, \text{hot}} = \left( \sum_{t=1}^{l_c} \alpha_{t}^{\text{hot}} \hat{x}_{t}^{\text{hot}} \right) \cdot \hat{x}_{t}^{\text{cold}}, \quad g_{s, \text{cold}} = \left( \sum_{t=1}^{l_c} \alpha_{t}^{\text{cold}} \hat{x}_{t}^{\text{cold}} \right).
\]

(13)

where \( l_c \) and \( l_h \) are lengths of the cold session and the hot session. \( \hat{x}_{t}^{\text{hot}} (\hat{x}_{t}^{\text{cold}}) \) represents representation of the t-th hot item or cold item in the session. \( \alpha_{t}^{\text{hot}} (\alpha_{t}^{\text{cold}}) \) is the learned attention coefficient of the t-th hot or cold item in the session, and \( \hat{g}_{s, \text{hot}} (\hat{g}_{s, \text{cold}}) \) 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:

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

(14)

As for those sessions which only contain hot items or cold items, we generate the corresponding type of session representations and swap them. The embedding recombination is illustrated in Fig. 1.(c).

5.3 Contrastive Self-Supervised Task

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 TT-core 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 \cite{51} to use InfoNCE [27] 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 \(B\). The loss of the contrastive task is defined as follows:

\[
L_{\text{cl}} = - \sum_{s \in B} \log \frac{\psi(W_{z^\text{tea}_s} \cdot W_{z^\text{stu}_s})}{\sum_{j \in B/\{s\}} \psi(W_{z^\text{tea}_j} \cdot W_{z^\text{stu}_j})},
\]

(15)

where \(\psi(W_{z^\text{tea}_s} \cdot W_{z^\text{stu}_s}) = \exp(\phi(W_{z^\text{tea}_s} \cdot W_{z^\text{stu}_s}) / \tau)\), \(\phi(\cdot)\) 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.

5.4 Predictive Self-Supervised Task

The contrastive task directly distills the recombined embeddings whilst ignoring the ground truth. We consider that under the direct guidance of the session embedding from the teacher model, the swapped session embedding from the student can improve its ability to predict the ground truth. We then let the teacher and student learn from the ground-truth using their recombined session representations, which also makes the KD framework compatible with the main recommendation task. The predictive loss is defined as follows:

\[
L_{\text{pred}} = - \left( \sum_{v=1}^{|V|} y_v \log \left( \hat{y}_v^{\text{tea}} \right) + (1 - y_v) \log \left( 1 - \hat{y}_v^{\text{tea}} \right) \right) - \left( \sum_{v=1}^{|V|} y_v \log \left( \hat{y}_v^{\text{stu}} \right) + (1 - y_v) \log \left( 1 - \hat{y}_v^{\text{stu}} \right) \right).
\]

(16)

where \(y\) is one-hot encoding vector of ground-truth, \(\hat{y}_v^{\text{tea}}\) represents the predicted score of the teacher on item \(v\) for each session \(s\), and \(\hat{y}_v^{\text{stu}} = \text{softmax}(W_{\text{tea}} \cdot z^\text{stu}_s)\), and \(W_{\text{tea}} \in \mathbb{R}^{N \times 2N}\). \(\hat{y}_v^{\text{stu}}\) is computed analogously.

5.5 Distilling Soft Targets

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 \(\text{prob}^{\text{tea}}\) and \(\text{prob}^{\text{stu}}\) represent the predicted probability distributions of the teacher and the student, respectively. Then for each session \(s\), we have:

\[
\begin{align*}
\text{prob}^{\text{tea}} &= \text{softmax}(\theta^\text{tea}_s^T X), \\
\text{prob}^{\text{stu}} &= \text{softmax}(\theta^\text{stu}_s^T X), \\
\end{align*}
\]

\[
\begin{align*}
\theta^\text{tea}_s \text{ and } \theta^\text{stu}_s \text{ are representations of session } s \text{ from teacher and student. Then to maximize the agreement between them, we define the loss of distilling soft targets as:}
\end{align*}
\]

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

(18)

5.6 Model Optimization

After training using knowledge distillation, we get refined item and session representations. The inner product of one candidate item \(v \in V\) and the given session \(s\) is the score of item \(v\) to be recommended to this session. We apply softmax to compute the probability:

\[
\hat{y} = \text{softmax}(\theta^s \cdot x_v).
\]

(19)

We then use the cross entropy loss to be the learning objective:

\[
L_{\text{rec}} = - \sum_{v=1}^{|V|} y_v \log (\hat{y}_v) + (1 - y_v) \log (1 - \hat{y}_v).
\]

(20)

\(y\) is the one-hot encoding vector of the ground truth. For simplicity, we leave out the \(L_2\) regularization terms. We jointly learn the recommendation task with the proposed self-supervised knowledge distillation framework through Eq. (12) which is:

\[
L = (1 - \beta_3)L_{\text{rec}} + \beta_1 L_{\text{cl}} + \beta_2 L_{\text{pred}} + \beta_3 L_{\text{soft}}.
\]

(21)

In the recommendation loss, we use a coefficient \(\beta_3\) to balance the recommendation task and the task of distilling soft targets.

6 EXPERIMENTS

6.1 Experimental Settings

6.1.1 Datasets. We evaluate our model using two real-world benchmark datasets: Tmall\(^1\) and RetailRocket\(^2\). Tmall is from IJCAI-15 competition and contains anonymized user’s 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 six months. For convenient comparison, we duplicate the experimental environment in \cite{40,41}. Specifically, we filter out all sessions whose length is 1 and items appearing less than 5 times. The latest one interaction of each session is set to be the test set and the previous data is used for training. The validation set is randomly sampled from the training set

\(^1\)https://tianchi.aliyun.com/dataset/dataDetail?dataId=42

\(^2\)https://www.kaggle.com/retailrocket/ecommerce-dataset
and makes up 10% of it. 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 \([v_{s,1}, v_{s,2}], (v_{s,1}, v_{s,2}, v_{s,3}), \ldots, (v_{s,1}, v_{s,2}, \ldots, v_{s,t-1}, v_{s,t})\) for every session \(s = [v_{s,1}, v_{s,2}, \ldots, v_{s,t}]\). Note that the label of each sequence is the last consumed item in it. The statistics of datasets are presented in Table 2.

### Table 2: Dataset Statistics

| 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

- **GRU4Rec** [10]: 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** [20]: An RNN-based model which employs attention mechanism to capture user’s main purpose and combines it with the temporal information to generate recommendations.
- **STAMP** [21]: Adopts attention layers to replace all RNN encoders and employs the self-attention mechanism [38] to model long-term and short-term user interests.
- **SR-GNN** [41]: Proposes a gated graph neural network to refine item embeddings and also employs a soft-attention mechanism to compute the session embeddings.

Following [41], we use P@K (Precision) and MRR@K (Mean Reciprocal Rank) to evaluate the recommendation results where K is 5 or 10. Precision measures the ratio of hit items while MRR measures the item ranking in the recommendation list.

#### 6.1.3 Hyperparameter Settings

As for the setting about 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 \(U(-0.1, 0.1)\). And in our base model, the number of attention layer and attention head are 1 and 2 for RetailRocket; 1 and 1 for Tmall. Dropout rate is 0.5 for Tmall and 0.2 for RetailRocket. We use Adam with the learning rate of 0.001 to optimize the model. For all baselines, we report their best performances with the same general experimental settings.

### 6.2 Capacity of Teacher Model

To validate if our base model is qualified to be a good teacher model, we compare it with all the baselines on the two datasets and present results in Table 3. It is clearly shown that our teacher model, which employs Transformer-based structure to learn item representations, outperforms other baselines on the two datasets. These results demonstrate that this teacher model is with the adequate capacity that can teach the highly compressed student model.

| Method     | Tmall   | RetailRocket |
|------------|---------|--------------|
|            | P@5    | M@5         | P@10    | M@10   | P@5    | M@5   | P@10    | M@10   |
| GRU4Rec    |        |             |         |        |
| NARM       |        |             |         |        |
| STAMP      |        |             |         |        |
| SR-GNN     |        |             |         |        |
| Teacher    |        |             |         |        |

#### 6.3 Effectiveness of Model Compressing

To evaluate the effectiveness and the generalization ability of the proposed method, we conduct a comprehensive study by investigating different compression rates. There are three factors in the student that can influence the compression rate: the shape of the factorized tensors, TT-rank \(R\) and hyperparameter \(n\) in semi-tensor product. To investigate the influence of factorization shape, we design four different student models using different tensor shapes, which are denoted as Stu-1, Stu-2, Stu-3 and Stu-4. And in those four models, the item embedding table for Tmall is decomposed into tensors with these shapes:

- **Stu-1**: \([169 \times 18 \times 79] \approx [169 \times 13 \times 241]\) and hyperparameter \(n = 8\)
- **Stu-2**: \([169 \times 8 \times 169] \approx [169 \times 13 \times 241]\) and hyperparameter \(n = 8\)
- **Stu-3**: \([169 \times 8 \times 169] \approx [169 \times 13 \times 241]\) and hyperparameter \(n = 16\)
- **Stu-4**: \([169 \times 8 \times 169] \approx [169 \times 13 \times 241]\) and hyperparameter \(n = 24\)

And for RetailRocket, the core tensor shapes of the four models are:

- **Stu-1**: \([117 \times 16 \times 316] \approx [117 \times 16 \times 316]\) and hyperparameter \(n = 8\)
- **Stu-2**: \([117 \times 16 \times 316] \approx [117 \times 16 \times 316]\) and hyperparameter \(n = 16\)
- **Stu-3**: \([117 \times 16 \times 316] \approx [117 \times 16 \times 316]\) and hyperparameter \(n = 32\)
- **Stu-4**: \([117 \times 16 \times 316] \approx [117 \times 16 \times 316]\) and hyperparameter \(n = 48\)

We report the compression rates (CR) and corresponding performances of the above four variants in Table 4 where we use \(R = 60\), \(n = 2\) for Tmall and \(R = 100\), \(n = 2\) for RetailRocket (Stu-1-60 means model Stu-1 with \(R = 60\)). It is apparent that the factorization shape has a great influence on the recommendation performance. To our surprise, on the two datasets, by comparing Stu-1 with Stu-2 and comparing Stu-3 with Stu-4, we find that the models which have smaller compression rates do not show better performance on most metrics. Such a finding is quite counter-intuitive. Though the reason is not clear, we suggest that, for a better performance, it is necessary to try different factorization shapes instead of empirically choosing a larger one.

To explore the relation between TT-rank and recommendation performance, we further experiment with different \(R\) values on Stu-1 and Stu-3 with same \(n\) value (\(n = 2\)). For Tmall, we select 20,
6.4 Ablation Study

To investigate the effectiveness of the STTD and the self-supervised knowledge distillation framework, we design five variants: Stu-base, Stu-TTD, Stu-c, Stu-p, and Stu-s. Stu-base means that the self-supervised knowledge distillation is disabled; Stu-TTD refers to the case where the conventional tensor-train decomposition is used; Stu-c, Stu-p, and Stu-s correspond to the models in which the contrastive task, the predictive task and the task of distilling soft targets are respectively detached. We report their performances on the two datasets in Figure 3. Here we adopt the settings of Stu-1 in Table 4 for its best performance on the two datasets.

By associating Table 3 and Fig. 3, we can observe that, the student model almost outperforms teacher model on the two datasets, showing the great effectiveness of our proposed framework, which is out of our expectation. Actually, similar results are also reported in other related research [43]. It indicates there is much parameter redundancy in Transformer-based recommendation models. In addition, though with a tiny size, Stu-base even shows better performance than STAMP and SR-GNN in most cases, implying that our model is an ideal solution for on-device recommendation. By comparing Stu-base with Stu-TTD, we can draw a conclusion that the semi-tensor product can further compress the model without any accuracy sacrifice. Comparing Stu-base with the full version of the student, KD improves the model by 16.86% on P@5, 12.31% on M@5, 16.35% on P@10, and 12.34% on M@10 on Tmall and improves the model by 9.53% on P@5, 3.21% on M@5, 11.24% on P@10 and 3.58% on M@10 on RetailRocket. It is concluded that the proposed self-supervised KD framework creates effective self-supervised signals and successfully transfers rich knowledge from the teacher to the student. As for the specific distillation tasks, we can see that all the three tasks are helpful. The task of distilling soft targets contributes the most among the three, while the contributions of the contrastive and predictive tasks are negligible.

| 
| 
| 
| 

| time | GRU4Rec | NARM | STAMP | SR-GNN | Student |
| Tmall | 0.417 | 0.264 | 0.175 | 0.310 | 0.169 |
| RetailRocket | 0.093 | 0.096 | 0.094 | 0.192 | 0.081 |

Table 7: Prediction time (s) on device.

6.5 Efficiency Comparison

The low latency is a prominent advantage of on-device models. We also study the efficiency of the student model on resource-constrained devices. We first train the student model on GPU with the help of the teacher, and then the well-trained student is encapsulated by PyTorch Mobile. We use Android Studio 11.0.11 to simulate virtual device environment and deploy the student on this virtual environment to do the inference. The selected device system is Google Pixel 2 API 29. We compare the prediction time for every 100 sessions of each model on the two datasets. For a fair comparison, these baselines are also deployed on the same virtual environment. The results are reported in Table 7. Obviously, our student model is faster than all the baselines with a much tinier model size on the two datasets.

Figure 3: Performances of all the variants of the student.

40, and 60 as the value of R for Stu-1 and 60, 80, 100 for Stu-3. For RetailRocket, we choose 40, 60, and 80 for both models. We report the results in Table 5. From the table, it is discovered that when using smaller TT-rank in Stu-1 or Stu-3, the model size decreases and the performance will drop on four measures on the two datasets.

To investigate the effects of hyperparameter n in STTD, we choose different n values for Stu-1 with two TT-ranks on each dataset because Stu-1 shows the best performance in previous experiments. The TT-rank and n value are denoted in the form of R-n. The results shown in Table 6 indicate that, compared with the factorization shape and TT-rank, the performance is more sensitive to the value of n. Increasing its value leads to a drastic performance drop. Obviously, there is a trade-off between performance and model size for on-device recommendation.
6.6 Performances on Long-Tail Items
To relieve the data sparsity problem, we propose to categorize the learned representations into hot and cold and then swap and recombine their session representations. To verify whether the proposed KD can address this issue, we split test set into two subsets based on the popularity of the ground-truth items. 95% items with the fewest clicks are labeled ‘long-tail’, and the left 5% items with the most clicks are labeled ‘Popular’. Then we compare the proportion that each category of items contributes to Pre@5 of the student model with or without self-supervised knowledge distillation framework on the two datasets. The results are presented in Table 8. On both datasets, we observe that after applying our proposed self-supervised knowledge distillation, there are growths on Pre@5 and the proportion contributed by the long-tail items significantly increases. It implies that the proposed KD framework can effectively relieve the data sparsity problem and the embedding recombination strategy makes student learn more expressive representations under the direct help of the teacher.

![Figure 4: Hyperparameter Analysis.](image)

Table 8: Performance on long-tail items.

| Model       | Tmall     | RetailRocket |
|-------------|-----------|--------------|
|             | Popular   | Long-tail    | Popular | Long-tail |
| Stu-base    | 5.30      | 14.93        | 7.50    | 24.61     |
| Student     | 6.03      | 17.43        | 7.72    | 27.45     |

6.7 Hyperparameter Analysis
We explore the sensitiveness of the three coefficients used in Eq. (21) on the two datasets. A large number of values are searched and we choose to report some representative values of the three parameters; they are \([0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5]\) for \(\beta_1\), \([0.001, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2]\) for \(\beta_2\) and \([0.2, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]\) for \(\beta_3\). When investigating one hyperparameter of them, we fix the rest at the best values of them. For Tmall, the best group is \((0.1, 0.001, 0.8)\). For RetailRocket, the best combo is \((0.005, 0.1, 0.8)\). The Pre@5 and Pre@10 results are presented in Figure 4. Obviously, for most cases on both datasets, with the increase of each parameter, the performance firstly drops, then starts to increase until reaching the peak, and then gradually decreases. Tmall has a little different curves towards \(\beta_2\), where the best performance is achieved at the starting point. The curves of \(\beta_3\) on RetailRocket also show a different pattern, where with the increase of \(\beta_3\), the precision values start to grow steadily until reaching the peak and then decline. Besides, for both datasets, it should be noted that the best value of \(\beta_3\) is larger than \(\beta_1\) and \(\beta_2\), indicating that the soft targets distillation plays a more important role in improving the student model, which is in line with the results in Figure 3.

7 CONCLUSION
Current recommender systems work in a cloud-based paradigm where models are trained and deployed in the server and use abundant memory storage and computing resources to generate recommendations. However, this scheme raises the privacy concern among users and it is not energy-saving. On-device recommendation addresses these issues by deploying lightweight models on edge devices and perform local inference. In this paper, we propose a novel on-device recommendation model for next-item recommendation in the session-based scenario. We loosen the dimension consistency constraint in tensor decomposition for an ultra-compact model and propose a self-supervised knowledge distillation framework to compensate for the capacity loss caused by the compression. Extensive experimental results demonstrate that, with a 30x size reduction, our proposed model can achieve comparable and even higher performances compared with its regular counterpart.

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

REFERENCES
[1] Ben Madhav Changmai, Divija Nagaraju, Debi Prasanna Mohanty, Kriti Singh, Kunal Banal, and Sukumar Miharan. 2019. On-device User Intent Prediction for Context and Sequence Aware Recommendation. arXiv preprint arXiv:1909.12756 (2019).
[2] Liyong Chen, Yongquan Chen, Jun tong Xi, and Xinyi Le. 2021. Knowledge from the original network: restore a better pruned network with knowledge distillation. Complex & Intelligent Systems (2021), 1–10.
[3] 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 2020 IEEE 36th International Conference on Data Engineering (ICDE). IEEE, 1405–1416.
[4] Tong Chen, Hongzhi Yin, Yu jia Zheng, Zi 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.
[5] Daizhan Cheng, Hong sheng Qi, and Ancheng Xue. 2007. A survey on semi-tensor product of matrices. Journal of Systems Science and Complexity 20, 2 (2007), 304–322.
[6] Sauptik Dhar, Junyao Guo, Jiayi Liu, Samarth Tripathi, Umesh 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.
[7] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization. arXiv preprint arXiv:1412.4115 (2014).
