RecJPQ: Training Large-Catalogue Sequential Recommenders

Aleksandr V. Petrov
University of Glasgow
United Kingdom
a.petrov.1@research.gla.ac.uk

Craig Macdonald
University of Glasgow
United Kingdom
craig.macdonald@glasgow.ac.uk

ABSTRACT

Sequential recommender systems rank items based on the likelihood of their next appearance in user-item interactions. Current models such as BERT4Rec and SASRec generate sequence embeddings and compute scores for catalogue items, but the increasing catalogue size makes training these models costly. The Joint Product Quantisation (JPQ) method, originally proposed for passage retrieval, markedly reduces the size of the retrieval index with minimal effect on model effectiveness by replacing passage embeddings with a limited number of shared centroid embeddings. This paper introduces RecJPQ, a novel adaptation of JPQ for sequential recommendations. We apply RecJPQ to SASRec, BERT4Rec, and GRU4rec models on three large-scale sequential datasets. Our results showed that RecJPQ can notably reduce model size (e.g., 48x reduction for the Gowalla dataset with no effectiveness degradation). RecJPQ can also improve model performance through a regularisation effect (e.g. +0.96% NDCG@10 improvement on the Booking.com dataset).

ACM Reference Format:
Aleksandr V. Petrov and Craig Macdonald. 2023. RecJPQ: Training Large-Catalogue Sequential Recommenders. In *WSDM '24), March 4–8, 2024, Mérida, Mexico.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Sequential recommender systems are a class of recommendation models that use the sequence of user-item interactions to predict the next item. Most of the state-of-the-art models for sequential recommendation are based on deep neural networks, for example, recurrent neural networks [17, 18], convolutional neural networks [49, 57], and most recently, transformers [25, 37, 38, 48]. All these models use learnable *item embeddings* as an essential component in their model architectures. Figure 1 illustrates item embeddings in a typical neural sequential recommendation model. As the figure shows, item embeddings usually have two roles in the model architecture: (i) to convert the sequence of input item ids to a sequence of item representation vectors and (ii) to convert the sequence embedding produced by the model into the distribution of predicted item scores. In both cases, a recommender system that works with an item set I requires an embedding tensor with $|I| \cdot d$ parameters, where d is the size of each embedding.

When a recommender has many items in the catalogue, various challenges arise in training the neural recommendation model. Firstly, the item embedding tensor may contain more model parameters than the rest of the model. For example, there are more than 800 million videos on YouTube[1]. If the model uses 128-dimensional embeddings, the whole item embeddings tensor will have more than 100 billion parameters, which is comparable with the number of parameters of the largest available machine learning models [2], even without accounting for the parameters of the model's intermediate layers. This is a problem specific to recommender systems. However, in the related area of dense passage retrieval [26, 27], passage embeddings are obtained by encoding passage text using a pretrained language model. In contrast, item side information, such as text, is not necessarily available in a typical recommender systems scenario; therefore, item embeddings should be directly learned from the interactions. Secondly, a large number of such trainable parameters also makes the model prone to overfitting. A third challenge caused by the large catalogue is the size of the output scores tensor (rightmost tensor in Figure 1): for example, in BERT4Rec, it contains a score for each item for every position, for every sequence in the training batch, so training BERT4Rec with more than 1 million items in the catalogue may be prohibitively expensive [37]. This problem is typically solved using negative sampling, whereby instead of computing the full output tensor, the model computes scores for a small proportion of negative items (e.g. SASRec [25] uses one negative per positive). However, negative sampling comes with its own challenges (for example, it usually requires informative negative mining [41]). Nevertheless, negative sampling is an orthogonal research direction, and in this paper, we use SASRec in cases where negative sampling is necessary. To summarise the challenges, a large embedding tensor increases the model size, slows model training down, and can necessitate further modelling tricks such as negative sampling, which bring their own challenges.

There are some existing methods [24, 51, 55] for item embedding compression (we discuss these methods in Section 2). However, most of these methods compress the embedding tensor *after* the model is fully trained (including training the full embedding tensor). However, as argued above, training may be prohibitively expensive in large-scale recommender systems. Hence, this paper addresses the problem of a large item embedding tensor in sequential recommendation models *at the training stage.*

To mitigate this problem, we propose a novel RecJPQ technique inspired by the success of a recent Joint Product Quantisation (JPQ) work [58] for text retrieval. JPQ itself is based on Product Quantisation (PQ) [22], a popular method of compressing vectors by splitting them into sub-embeddings and encoding them using a discrete centroids codebook (the codebook maps from item ids to the associated centroids; see Section 3.1 for the details). The main innovation of JPQ compared to the standard PQ method is that it learns the centroids embeddings as part of the overall model training process. In contrast, PQ requires training the model first and only then compressing the embeddings (frequently, this second step uses external tools, such as FAISS [23]). This means that JPQ does not need to keep the embedding matrix in memory during model training. We argue that this innovation is valuable for recommender systems. Indeed, as mentioned above, real-life recommender systems can have hundreds of millions of items in their catalogues and keeping full

[1] https://earthweb.com/how-many-videos-are-on-youtube/

WSDM '24, March 4–8, 2024, Mérida, Mexico

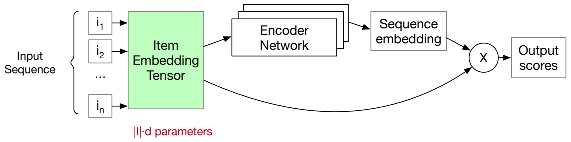


Figure 1: Item embeddings in a typical sequential recommender system. These item embeddings are used in two ways: (i) to obtain sequence representation and (ii) to generate item scores. The embedding tensor requires $|I| \cdot d$ trainable parameters, where $|I|$ is the items catalogue size, and d is the size of an embedding. When catalogue size $|I|$ is large, item embeddings comprise most of the model's parameters.

embeddings tensor in memory may be prohibitively expensive. This is particularly important for deep-learning-based sequential recommender systems because these models require keeping the whole model in GPU (or TPU) memory during training. GPU memory is costly even when compared to regular computer RAM.

Unfortunately, it is hard to adapt JPQ to the recommendation scenario, as it is specific to textual information retrieval. In particular, JPQ assumes the existence of a pre-trained passage retrieval model and index, which it uses to assign items to centroids (see more details in Section 3.2). These pre-trained models rarely exist in item recommendations. Hence, in RecJPQ, we experiment with performing the initial assignment of centroids using three different strategies: (i) discrete truncated SVD (centroids obtained by discretising the item representations obtained by an SVD decomposition of the user-item matrix), (ii) discrete BPR (centroids obtained by discretising the item embeddings obtained from BPR) and (iii) random assignments. We describe these assignment strategies in detail in Section 4.

RecJPQ is a model component that replaces traditional item embeddings in sequential recommender systems. In general, it can be applied to any recommender system based on item embeddings, but in this paper, we focus specifically on sequential models, as in these models, item embeddings comprise the biggest part of the model (e.g. sequential models usually do not have user embeddings). In contrast with existing methods, RecJPQ does not require training full uncompressed embedding and does not modify the original model loss function. Our experimentation on three datasets (see Section 5) demonstrates that RecJPQ can be successfully applied to different models, including SASRec [25], BERT4Rec [48] and GRU [18, 37], achieving a large factor of embeddings compression (e.g. 47.94x compression of SASRec on Gowalla) without any effectiveness degradation. Moreover, on 2 out of our 3 experimental datasets, applying RecJPQ *increases* model performance (e.g. +0.96% NDCG@10 on Booking.com dataset, significant improvement); we attribute these improvements to model regularisation.

In short, the contributions of this paper are as follows:

(1) We propose RecJPQ, a novel technique for reducing the size of sequential recommendation models during training based on Joint Product Quantisation.
(2) We propose three strategies for assigning codes to items, two of which (discrete truncated SVD and discrete BPR) assign similar codes to similar items, and one assigns codes randomly.
(3) We perform an extensive experimental evaluation of RecJPQ on three datasets and show that RecJPQ allows reducing the models' size without hindering the model performance.

The rest of the paper is organised as follows: Section 2 introduces related work on embeddings compression and identifies limitations of existing methods; Section 3 covers Product Quantisation (PQ) and Joint Product Quantisation (JPQ) - the methods, which serve as the basis for our work; Section 4 introduces RecJPQ and covers centroid assignment strategies for RecJPQ; in Section 5 we experimentally evaluate RecJPQ; Section 6 contains final remarks.

Table 1: Existing embedding compression methods. Desired method characteristics are highlighted in bold.

Model Agnostic	Method	Backbone Models	Sequential backbone	Trains full embeddings	Compression Unit
No	EODRec [55]	SASRec [25]	**Yes**	Yes	**Item**
	LightRec [31]	DSSM [20]	No	Yes	**Item**
	MDQE [51]	SASRec [25]	**Yes**	Yes	**Item**
Yes	PreHash [47]	BiasedMF [28]; NeuMF [16]	No	**No**	User
	Quotient Remainder [46]	DCN [52]; DLRM [35]	No	**No**	Features
	MGQE [24]	SASRec [25]; NeuMF [16]; GCF [16]	**Yes**	Yes	**Item**
Yes	RecJPQ (ours)	SASRec [25]; BERT4Rec [48]; GRU [18]	**Yes**	**No**	**Item**

2 RELATED WORK

This section covers existing work on compressing and discretising embeddings in recommender systems, identifies the limitations in existing work and positions our contributions in the context of existing methods. Table 1 summarises existing methods and positions RecJPQ, our proposed compression technique. The table highlights with boldface the desirable characteristics necessary for training a large-scale[2] sequential model, specifically: the method can be applied to work with different *backbone* sequential models, and does not require training full embeddings (as we work with the assumption that full embeddings tensor does not fit into GPU memory); and we want the model to focus on item embeddings rather than embeddings of other entities, such as users or features. As illustrated in the table, the methods for compressing the models can be broadly divided into two groups: *model dependent* and *model agnostic*.

In the model-dependent methods [31, 55], embedding compression mechanism is integrated as a component into the recommendation model itself. Hence the training architecture of these methods has to be aware of the compression, and the loss function includes components responsible for the embedding compression. For example, LightRec [31] uses the Deep Semantic Similarity Model (DSSM) [20] as the backbone model and uses an additional knowledge distillation component in the loss, which allows for learning compressed representations of the embeddings. However, DSSM is not a sequential model, and it is unclear whether or not the method can be adapted to the sequential recommendation case. Similarly, EODRec [55] which uses SASRec [25] as its backbone, one of the most popular sequential models based on the Transformer architecture [50]. The loss function of EODRec also consists of four components, some of which are responsible for recommendation, and others are responsible for embedding compression. While SASRec, used by EODRec as its backbone, is an efficient model, in many cases other models such as BERT4Rec show better results [38, 48]. In general, while some model-dependent methods may substantially reduce the size of a trained model, these methods have several limitations, which make them unsuitable for training sequential recommendation models with large catalogues. In particular:

L1 Model-dependent methods are, by their nature, tied to a specific model, making them inflexible when adapting to a specific task. For example, LightRec [31] uses a non-sequential DSSM model as a backbone. The core component of LightRec (Recurrent Composite

[2] For simplicity, we say that a catalogue is "large-scale" if it has more than 1 million, as it becomes challenging to train recommender systems on that scale [37].

Encoding) is tightly integrated into the DSSM architecture, and it is unclear whether or not it can be used outside of DSSM.

L2 Model-dependent methods usually require training (uncompressed) item embeddings and then use knowledge distillation or teacher-student techniques to obtain compressed representations of the embeddings. This approach substantially reduces the final model size, thereby helping inference on smaller devices, but requires a large amount of GPU memory while training, thereby limiting the overall number of items in the catalogue. For this reason, the main positioning for EODRec model [55] is the on-device recommendation: while the final model produced by this method is small, it requires storing full item embeddings while training. Post-training quantisation methods [56], which recently became popular to reduce the size of large language models via quantising their weights into lower-precision numbers (e.g. float16, or int8) also have this limitation – they need to have access to full model before quantising. Similarly, Mixed Precision Training [34] builds a smaller precision model, but it requires keeping full precision weights in memory. Placing the embeddings tensor into Approximate Nearest Neighbours [23] or Hierarchical Navigable Small Worlds [33] indexes also requires access to the full embedding tensor at model training time, and therefore also exhibits this limitation.

L3 Model-dependent methods require multi-component loss functions, some of which are responsible for the recommendation task and others for the model compression. This is a form of multi-objective optimisation, which is a challenging problem [8, 45], as finding the balance between the loss components for different objectives usually requires extensive hyperparameters search.

On the other hand, the existing model-agnostic methods [46, 47] do not depend on the specific model architecture, and likewise do not add extra components to the loss functions. Typically, these methods implement a mechanism that takes the place of the embeddings tensor in the backbone model, and hence can be used with many models. However, on inspection of the relevant work, we identified additional limitations of these methods:

L4 Many methods are not designed for compressing item embeddings. For example, PreHash [47] is a method specific for compressing user embeddings (i.e. it uses the user's history to construct user embeddings). The method uses an attention network over the history of user interactions. Adapting this network structure for items embeddings is a hard task: a user may only interact with a few items; in contrast, a popular item may be interact with by millions of users. The attention mechanism depends quadratically on the sequence length, and therefore applying it to users who interacted with a popular item would be prohibitively expensive.

L5 Finally, many methods lack structure in compressed item representations. This leads to situations where unrelated items have similar representations and, conversely, when similar items have dissimilar representations. Both these cases may limit the models' generalisation ability and hinder the models' performance. One example of such unstructured methods is hashing-based Quotient Remainder [46], which compresses embeddings of categorical features (e.g., genres). Quotient Remainder assigns feature codes based on the quotient and the remainder of the division of the feature id by some number. When applied to item ids (items can also be seen as categorical features), the quotient and the remainder are unrelated to the item characteristics. Hence similar items are unlikely to have similar codes. Nevertheless, Quotient Remainder is one of the few methods that can be used to train a model on a large-scale dataset, and therefore we use this method as a baseline in our experiments.

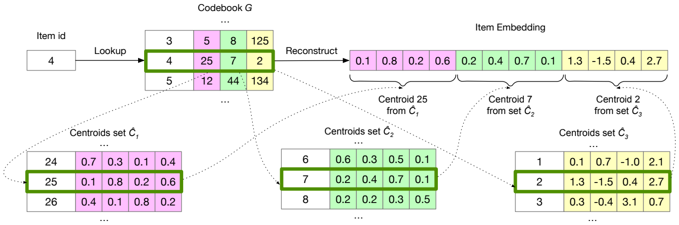


Figure 2: Reconstruction of item embeddings in RecJPQ: Codebook length $m = 3$, item embedding length $d = 12$, number of centroids per split $b = 256$.

Overall, among the related work, we argue that existing methods exhibit a number of Limitations (L1-L5). In summary, the model-dependent methods require training full embeddings first, limiting the maximum number of items that can be considered in the catalogue of the recommender system. On the other hand, methods which do not require training full embeddings first, such as Quotient Remainder, rely on simple heuristics and may result in unrelated items having similar representations. On the surface, the nearest related work to ours is VQ-Rec [19] as it also applies JPQ-style technique to sequential recommendation; however, similarly to JPQ, it relies on the availability of textual features and pre-trained language models. In contrast, our work's main novelty is adapting JPQ to the scenario when (e.g., textual) side information is unavailable. In the next section, we describe JPQ [58], a method of embedding compression for information retrieval, which directly learns embeddings in compressed form, reducing GPU memory requirements. Then, in Section 4, we propose RecJPQ - an adaptation of JPQ to the sequential recommendation task, which successfully addresses Limitations L1-L5.

3 PRODUCT QUANTISATION AND JPQ

We now describe Product Quantisation (PQ) and Joint Product Quantisation (JPQ), two methods which serve as a backbone for our method. Section 3.1 covers PQ, a classic embedding compression technique. Section 3.2 describes JPQ - a recently proposed information retrieval method that learns compressed embeddings directly instead of compressing them after the model training.

3.1 Product Quantisation

Product quantisation [13, 22] is a well-cited method of compressing vectors used by many libraries, such as FAISS [23] & nanopq[3]. Its main idea is to split a collection of d-dimensional vectors, V, into m collections $V_i; i = 1..m$ of smaller vectors of $\frac{d}{m}$ dimensions each. The original vectors can be recovered back via concatenation: $V = concat(V_1, V_2, ...V_m)$. Product quantisation then clusters each V_i into b clusters (e.g. using the k-means algorithm [32]) and replaces each vector v_{ij} with the centroid of the assigned cluster $c_{ij} \approx v_{ij}$. With this replacement, the original vectors collection can be approximated as a concatenation of the centroid matrices $C_1...C_m$, which are constructed from $V_1...V_m$ by replacing each vector v_{ij} by the closest centroid c_{ij}:

$$V \approx concat(C_1, C_2, ...C_m) \tag{1}$$

Note that in each centroid matrix C_i, there are, at most, b different rows, as each row corresponds to one of the centroids of the clusters, so instead of storing full matrix C_i, we can store these unique rows in the separate tensor $\hat{C}_i$, whose elements $\hat{c}_{ij}, j \in \{1..b\}$ correspond

[3] https://github.com/matsui528/nanopq

Table 2: Analysis of PQ's impact on memory requirements for storing item embeddings tensor for selected recommendation datasets, based on 512-dimension float32 vector embeddings. The table compares base memory usage with different code lengths, shown as percentages relative to the base.

Dataset	Num Items	Size of Item Embedding Tensor			
		Base	Code length=2 512 centroids 1.00 MB	Code length=8 2,048 centroids 4.00 MB	Code length=32 8,192 centroids 16.00 MB
MovieLens-1M	3,416	6.67 MB	14.988%	59.953%	239.813%
Booking.com	34,742	67.86 MB	1.474%	5.895%	23.580%
Gowalla	1,280,969	2.44 GB	0.040%	0.160%	0.640%

to the unique centroids. To compress the approximate embedding matrix defined by Equation (1), we need only store the ids of centroids. Overall, there are m centroid sets C_i, so each vector can be encoded using m integer codes, and each code can have b different values; therefore, overall, this scheme can encode up to b^m different vectors. The vector of centroid ids $g_i = \{g_{i1}, g_{i2} ..., g_{im}\}$ associated with the vector $v_i \in V$ is known as the *code* of the vector v_i [22]. The number of centroids associated with an item m is the *length* of the code. The table G of codes associated with each vector from V is also known as a *codebook* [22].

Figure 2 illustrates the vector reconstruction process applied to item embeddings. For each centroid id g_{ij} in the codes vector g_i of an item i, we extract a centroid associated with this centroid id and then concatenate the centroids to obtain reconstructed item embeddings.

The number of splits m is usually considerably smaller than the original vector dimensions d: $m \ll d$ to achieve compression. The number of centroids per split, b, is typically a power k of 256, so that the codes can be stored as k-byte integers. In this paper, for simplicity and following JPQ [58], we fix $k = 1$, so each centroid id can be represented with a single byte; therefore, we only store m bytes for each item. Even with fixed k (and therefore b), we can adjust model capacity by controlling the number of centroids associated to each item, m, and the dimension of the centroid embeddings, $\frac{d}{m}$, by adjusting d. Figure 2 illustrates RecJPQ for $m = 3$, $d = 12$, $b = 256$. Further, to illustrate the achieved compression, if embeddings are stored as 256-dimensional float32 vectors, a full-item embedding requires 1KB of memory. After compression using product quantisation with $m = 8$ splits, we only need to store 8 bytes per item (0.78% of the original memory requirement). While some memory is also required to store the centroids themselves, this is a negligible for large datasets compared to the original vector requirements (see also Table 2 for an analysis of centroids memory requirements).

Product Quantisation is a well-established vectors compression technique, which has been shown to be successful in approximate nearest neighbour methods [10, 22, 23, 30], information retrieval [21, 27, 44] and recommender systems [1, 24, 51]. However, Product Quantisation requires the *full* embeddings tensor to be trained before the embeddings are compressed. Indeed, the quantisation operation is not differentiable, and therefore the model can not be trained end-to-end. Therefore, it requires first training full (non-quantised) model and only after that apply compression. Some recommendation models (e.g. [24]) use differentiable variations of Product Quantisation to allow end-to-end training, but these methods still require training full embeddings alongside the quantised versions.

Overall, Product Quantisation addresses Limitations **L1** (it is model agnostic), **L5** (it is applicable for training item embeddings compression), and **L4** (when centroids assigned using clusterisation, similar items will have similar codes). However, it does not address Limitations **L2** (it requires training full embeddings first),and **L3** (PQ uses a separate loss function for embeddings reconstruction, which is not aligned with the ranking loss). In the next section, we discuss Joint Product Quantisation, a method that can be adapted to addressing these remaining limitations.

3.2 Joint Product Quantisation

Zhan et al. recently proposed *Joint Product Quantisation (JPQ)* [58], a Product Quantisation-based method developed for dense information retrieval. The main difference with the classic product quantisation is that JPQ generates item codes *before* training the model. As code assignment is the only non-differentiable operation in Product Quantisation, therefore when the codes are assigned before training the model, the model can be trained end-to-end without training full item embeddings – JPQ essentially replaces the embeddings tensor in the model, where item embeddings are constructed via centroid concatenation, as illustrated in Figure 2. Assuming that the codebook in the figure is a constant, all other parameters can be learned using standard gradient descent. In particular, centroid embeddings in JPQ are learnable model parameters, in the same way that item embeddings are learnable parameters in a conventional learned recommendation model. This means that JPQ does not require special loss function components to learn centroids, as they are learned as part of the overall model training process.

Compared to the original Product Quantisation method, JPQ addresses Limitations **L2** (it does not require training full item embeddings) and **L3** (it does not require a specific loss function). However, in contrast to plain PQ, JPQ does not provide a mechanism to assign similar embeddings to similar items and therefore does not address Limitation **L4**. The centroid assignments method proposed in the original JPQ paper is specific for text retrieval (as it relies on the existence of a pre-built index for a text document collection generated using the STAR model [59]). In the next section, we introduce RecJPQ, an adaptation of JPQ to the sequential recommendation scenario, which does not rely on text-specific datasets and models. Our adaptation of JPQ to sequential recommendation requires careful design of novel centroid assignment strategies. Indeed, to the best of our knowledge, this is the first adaptation of the JPQ method to sequential recommendation.

4 RECJPQ

RecJPQ is a Joint Product Quantisation-based method for training recommendation models with a large catalogue of items. As we discuss in Section 3.2, the method used by JPQ for initial centroids assignments relies on the existence of a pre-built index of documents and, therefore, can not be directly used for recommendation scenarios. Hence, the main difference between RecJPQ and the original JPQ is centroid assignment strategies. In general, RecJPQ can be described as follows:

(1) Build the item-code mapping matrix (codebook) using one of the centroid assignment strategies (described in Section 4.1).
(2) Initialise the centroid embeddings randomly.
(3) Replace the item embedding tensor with the concatenation of centroids associated with each item (as illustrated in Figure 2).
(4) Train the model end-to-end using the model's original training task and loss function (this process also trains the centroid embeddings, so they do not need to be trained separately).

The way RecJPQ builds the codebook before training the main model is also similar to how language models (such as BERT [9]) train a tokenisation algorithm before training the main model. Language models also learn embeddings of sub-word tokens instead of learning embeddings of full words to reduce the model's size; similarly, RecJPQ learns sub-item centroid embeddings instead of learning embeddings of full items. Below, we describe centroid assignment strategies used by RecJPQ.

4.1 Centroid Assignment Strategies

4.1.1 Random Centroid Assignments. In the most simple scenario, we can assign items to centroids randomly. We compose the item code out of m random integers in this case. RecJPQ with random centroids assignments strategy does not address Limitation **L4** (similar items do not have similar codes). Indeed, with random centroid assignments, RecJPQ becomes similar to other "random" embeddings compression methods, such as the hashing trick [53] or Quotient Reminder method [46]. The main problem with these methods is that unrelated methods are forced to share parts of their representation, which limits the generalisation ability of the models. However, as we show in Section 5, sometimes random assignments may be beneficial, as the random assignments strategy acts as a form of regularisation. Nevertheless, as the random centroid assignments strategy does not address Limitation **L4** (similar items should have similar representations), we introduce further centroid assignment strategies that can address this limitation in the next sections.

4.1.2 Discrete Truncated SVD. As discussed in Section 3.2, the only limitation not addressed by JPQ is Limitation **L4** (similar items should have similar codes). Random centroid assignments discussed in the previous section do not address this limitation either. Hence, in this section, we design a centroids assignments method that addresses this remaining limitation and assigns similar codes to similar items. Some approaches have used item side information, such as textual data for item representations [40]; however, we address the more generic classic sequential recommendation scenario (which does not rely on item side information) and hence must infer item similarities from the user's sequences. To achieve this, we employ the SVD algorithm, which has been shown to achieve good results in learning item representations [61] for recommender systems.

We first compute the matrix of sequence-item interactions M, where rows correspond to sequences (users), and columns correspond to items. This matrix's elements m_{ij} are either 1 if i^{th} contains interactions with item j and 0 otherwise. We then compute the truncated SVD decomposition of matrix M with m latent components: $M \approx U \times \Sigma \times V^T$, where U is the matrix of user embeddings, V is the matrix of item embeddings, and Σ is the diagonal matrix of largest singular values.

Our initial experiments showed that some items have equal embeddings after performing truncated SVD decomposition. This happens when two items interacted with exactly the same set of users. To ensure that all items have different embeddings, we normalise V using the min-max normalisation range and add a small amount of Gaussian noise: $\hat{v}_{ab} = \frac{v_{ab} - \min\limits_k v_{ak}}{\max\limits_k v_{ak} - \min\limits_k v_{ak}} + \mathcal{N}(0, 10^{-5}); \forall v_{ab} \in V$

The variance of the noise (10^{-5}) is negligible compared to the range of possible values of normalised embeddings ([0..1] after min-max normalisation). Therefore it has a very small influence on the position of the items in the embeddings space. However, if two items have exactly the same embeddings after decomposition (e.g. this can happen if two items appear in exactly the same set of sequences), the noise allows us to distinguish these two embeddings. Lastly, the assignment of centroids involves discretising each dimension of the normalised item embeddings into b quantiles so that each quantile contains an approximately equal number of items. We use these bins as centroid assignments for the items. Note that although this method requires computing an m-dimensional item embeddings tensor (and there can be hundreds of millions of items), it does not require computing them as part of a deep learning model training on a GPU. Indeed, truncated SVD is a well-studied problem. There are effective algorithms for performing it that do not require modern GPUs [14]. Moreover, as the method only requires computing m-dimensionsional embeddings, the table will be many times smaller than full d-dimensional embeddings, so the method requires $\frac{d}{m}$ times less memory to store embeddings. Finally, performing truncated SVD is possible in a distributed manner[4], which makes it is possible to perform truncated SVD even on very large datasets.

In summary, discrete truncated SVD allows assigning similar codes to similar items; it does not require a GPU for intermediate computations and can be easily performed for very large datasets.

4.1.3 Discrete BPR. Truncated SVD is not the only Matrix Factorisation method that can be used for initial centroid assignments. In particular, we also use the classic BPR approach [42] to obtain coarse item embeddings. The method also learns user embeddings (or, in our case, sequence embeddings) U and item embeddings V. The estimate of the relevance of an item i for user j is defined as the dot product of user and item embeddings: $r = u_j \cdot v_i$. In contrast with truncated SVD, BPR does not directly approximate the user-item interaction matrix. Instead, BPR optimises a pairwise loss function that aims to ensure that positive items are scored higher than negative items: $\mathcal{L}_{BPR} = -\log(\sigma(u_i \cdot v_{j^+} - u_i \cdot v_{j^-}))$, where v_{j^+} is the embedding of a positive item for the user u, v_{j^-} is the embedding of a randomly sampled negative item, and σ is the logistic sigmoid function.

BPR is a very successful and one of the most cited methods in recommender systems, and therefore we use BPR as an alternative strategy for coarse item embedding learning. The rest of the discrete BPR strategy is the same as in the truncated SVD: we also normalise the learned embeddings using min-max normalisation and add a small amount of Gaussian noise to ensure different embeddings for different items. Similar to truncated SVD, BPR does not require learning on a GPU, and there exist distributed implementations[5] that allow for learning item embeddings on very large datasets, so overall it can be used as an alternative to truncated SVD for centroids assignments in RecJPQ.

This concludes the description of the centroid assignment strategies for RecJPQ. We now discuss why RecJPQ may act as a regularisation mechanism and improve the performance on the datasets with many long-tail items.

4.2 RecJPQ as a Regularisation Mechanism

The interactions with items in recommender systems typically have long tail distribution [36], meaning that few popular items have the most interactions. In contrast, most items comprise the "long tail" with few interactions. As the training data for these long-tail items is limited, models suffer from overfitting on long-tail items [60], which causes overall performance degradation.

Goodfellow et al. [12, Chapter 7.9] argued that one of the most powerful techniques for preventing overfitting is *parameters sharing*

[4] For example, using Apache Spark
https://spark.apache.org/docs/latest/mllib-dimensionality-reduction

[5] https://github.com/alfredolainez/bpr-spark

Table 3: Salient characteristics of experimental datasets. Long tail items are the percentage of items in the catalogue with less than five interactions.

Dataset	Users	Iems	Interactions	Average sequence length	Long tail items
MovieLens-1M	6,040	3,416	999,611	165.49	0.0%
Booking.com	140,746	34,742	917,729	6.52	61.8%
Gowalla	86,168	1,271,638	6,397,903	74.24	75.8%

- a technique where certain parameters of the model are forced to be equal. RecJPQ is a special case of parameter sharing: we force different items to share parts of their embeddings. This prevents the model from learning item embeddings that are too specific to only a few training sequences, as each part of the embedding appears in many other sequences via the sharing mechanism.

In our experiments (see Section 5), we indeed observe that RecJPQ may act as a model regulariser and improve the model's performance; this is especially apparent in the Gowalla dataset, where the proportion of long-tail items is the largest.

4.3 RecJPQ: Summary

In summary, RecJPQ is a model component that takes the place of the item embeddings tensor in sequential recommender systems. RecJPQ is based on the JPQ method, which is a variation of Product Quantisation in turn. RecJPQ addresses all of the limitations described in Section 2: it is model-agnostic (Limitation **L1**); does not require training full embeddings (Limitation **L2**); does not modify the backbone model's loss function (Limitation **L3**); it is suitable for item embeddings compression (Limitation **L5**); it can assign similar codes to similar items with the help of discrete truncated SVD or discrete BPR (Limitation **L4**). Futhermore, we argue that RecJPQ may act as a model regulariser, which is an additional advantage when there are many long-tail items in the catalogue. This concludes the description of RecJPQ. In the next section, we experimentally evaluate RecJPQ and analyse its effects on required memory and on the model performance.

5 EXPERIMENTS

Our experiments address the following research questions:

RQ1 What are the effects of centroid assignment strategy in RecJPQ?
RQ2 How do code length m and embedding size impact effectiveness?
RQ3 What is the effect of RecJPQ on size/effectiveness tradeoff?

5.1 Experimental Setup

5.1.1 Backbone Models. In our experiments, we use two state-of-the-art Transformer-based sequential recommendation models: BERT4Rec [48] - a model that uses a transformer encoder based on BERT [9]; and SASRec [25] - a model which utilises decoder part of the Transformer (similar to GPT [39]). For both models, we use the versions[6] from a recent reproducibility paper [38], which provides efficient & effective implementations (using the popular Huggingface transformers library [54]). Additionally, to demonstrate that RecJPQ can be applied to other architectures, we use a GRU [6]-based model from [37] available in the same repository. This model uses the GRU4Rec [17] architecture, but a slightly different configuration, e.g. it uses LambdaRank [3] as a loss function, which is shown to be effective [37].

5.1.2 Datasets. We experiment with three datasets: (i) MovieLens-1M (denoted ML-1M) [15] - this is a movie rating dataset that is one of the most popular benchmarks for recommender systems; (ii) Booking.com [11] - a multi-destination trips dataset, and (iii) Gowalla [5] - a check-in dataset. Following common practice [38, 48], we remove users with less than 5 interactions.

Table 3 lists the salient characteristics of the datasets after preprocessing. As can be seen from the table, the number of items in these datasets varies from relatively small (3416 in MovieLens-1M) to large (1,271,638 in Gowalla) - this allows testing RecJPQ in different settings (RecJPQ is designed for large datasets, and we expect it to compress the model by a larger factor on Gowalla).

The datasets are also diverse regarding the number of "long-tail items" (defined as items with less than five interactions). While the MovieLens-1M dataset does not have long-tail items, the Booking.com dataset has 60.8% long-tail items, and Gowalla has 75.8% long-tail items. As discussed in Section 4.2, RecJPQ acts as a model regulariser in long-tail distributions, and we expect to see the highest regularisation effect on the Gowalla dataset.

5.1.3 Evaluation Protocol. Overall, our evaluation protocol follows the protocol from the recent replicability paper [38]. We use a leave-one-out data splitting strategy: we hold out the last sequence in each sequence in the test set. Additionally, for 1024 randomly selected users, we hold out the second last action into a validation set, which we use for the early stopping mechanism.

We set the maximum sequence length at 200. If the sequence contains more than 200 interactions, we use 200 latest interactions. If the sequence contains less than 200 interactions, we left-pad it to ensure its length is exactly 200.

To ensure that the models are fully converged, following [37], we employ an early stopping mechanism on the NDCG@10 metric: we stop training if the metric is not improved for 200 epochs.

5.1.4 Metrics. The main topic of our research is the trade-off between model size and model effectiveness. For measuring effectiveness, following prior research [25, 38, 48], we use NDCG@10, and as the model size metric, we use the file size of the model checkpoint. Following recent recommendations [4, 7, 29], we measure NDCG without using negative sampling.

5.1.5 Baselines. We deploy an adaptation of Quotient Remainder [46] as a baseline compression approach, applied to each base model – this parameter-free hashing-based approach encodes each item using two hashes: the quotient and the remainder of the division of item id by $\left\lceil \sqrt{|I|} \right\rceil$ where $|I|$ is the catalogue size. Quotient Remainder guarantees that each item has a unique code.

We do not apply post-training embedding quantisation (e.g. float16), nor use other methods from Table 1 as baselines, as they are not suitable for our task: EODRec, LightRec, MDQE and MGQE require training full embeddings (we assume that training full embeddings is not an option for a large catalogue), and PreHash is specific for compressing user embeddings, so is not suitable for item embeddings. However, reducing the model size by decreasing the embedding dimensionality can also be seen as a simple baseline. We analyse models using different embedding sizes in Section 5.2.3.

5.2 Results

5.2.1 RQ1. Effect of centroid assignment strategy. To analyse the effect of the centroid assignment strategy on model performance/model

[6] The code for the paper is available at https://anonymous.4open.science/r/RecJPQ-6643/README.md

Table 4: Impact of RecJPQ with different centroid assignment strategies on model size and effectiveness. Relative Size corresponds to model checkpoint size as the percentage of the base model. $^=$, $^+$, and $^-$ denote significance testing results compared to the base, respectively: indistinguishable (*pvalue* > 0.05, Bonferroni multi-test correction), better or worse.

Model→	BERT4Rec		GRU		SASRec	
Strategy↓	NDCG @10	Relative Size	NDCG @10	Relative Size	NDCG @10	Relative Size
			ML-1M			
Base	0.157	100.0%	0.072	100.0%	0.131	100.0%
Hashing (Quotient-Remainder)	0.040^-	92.4%	0.017^-	61.6%	0.009^-	124.9%
RecJPQ-BPR	$0.156^=$	93.2%	$0.076^=$	62.5%	$0.130^=$	128.0%
RecJPQ-Random	$0.156^=$	93.2%	$0.075^=$	62.5%	$0.125^=$	127.6%
RecJPQ-SVD	$0.154^=$	93.2%	$0.074^=$	62.5%	$0.129^=$	127.9%
			Booking			
Base	0.376	100.0%	0.209	100.0%	0.137	100.0%
Hashing (Quotient-Remainder)	0.192^-	62.8%	0.186^-	27.6%	0.014^-	9.2%
RecJPQ-BPR	$0.375^=$	63.3%	0.334^+	27.5%	0.242^+	8.7%
RecJPQ-Random	0.316^-	62.3%	0.324^+	27.5%	0.256^+	8.9%
RecJPQ-SVD	0.379^+	63.3%	0.334^+	27.6%	0.185^+	8.8%

size tradeoff, we compare the original (base) versions of BERT4Rec, SASRec and GRU with RecJPQ versions trained with Random, discrete truncated SVD and discrete BPR centroid assignment strategies. We do not train GRU and BERT4Rec on Gowalla, as these models do not use negative sampling. Training models on this dataset without negative sampling is not feasible due to the large GPU memory requirement for storing output scores [37], while applying negative sampling is a substantial change to the models' training process that is outside of the scope of this paper.

In all cases, we use 512-dimensional embeddings and the code of length $m = 8$ (we experiment with other embedding sizes and lengths of the code in the next section). One exception is the base SASRec model on Gowalla; in this case, we use 128-dimensional item embeddings (item embeddings larger than 128 dimensions consume all available GPU memory when embedding compression techniques are not deployed).

5.2.2 RQ2. Effects of code length m and the embedding size on model performance. Table 4 shows the experimental results on the smaller ML-1M and Booking datasets, while Table 5 reports results for the Gowalla dataset. The tables compare NDCG@10 and model size of compressed variations of backbone models with the base (uncompressed) model. Significant differences compared to the corresponding base model (BERT4Rec, GRU or SASRec) are indicated. In general, the tables show that RecJPQ substantially reduces the model checkpoint size in most cases. For example, the RecJPQ versions of the GRU models on the Booking dataset are approximately 27% of the original in size. On the Gowalla dataset, compressed models are approximately 3% of the original. Moreover, model size does not depend on the centroid assignment strategy. Indeed, centroid assignments only influence the values of the model parameters but not the number of parameters. Moreover, Quotient Remainder models have approximately the same compression level as RecJPQ models. We speculate that after compression, the model checkpoint size is dominated by other model parameters (e.g., attention matrices). In our configuration, the centroid embeddings tensor only requires a few megabytes of memory (see Table 2). In contrast, the full model checkpoint of a compressed model is typically tens of megabytes (e.g., 92.8MB for SASRec using RecJPQ-BPR trained on Gowalla).

Table 5: Impact of RecJPQ with different centroid assignment strategies on SASRec model size and effectiveness on the large-scale Gowalla dataset. Notations follow Table 4.

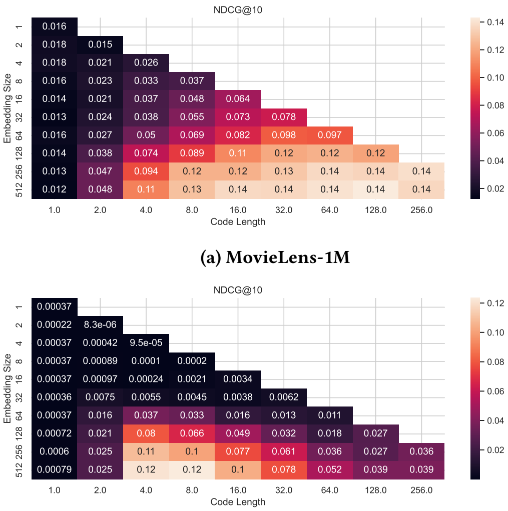
Strategy	NDCG@10	Relative Size
Base	0.110	100.0%
Hashing (Quotient-Remainder)	0.081^-	2.8%
RecJPQ-BPR	0.033^-	2.8%
RecJPQ-Random	0.173^+	2.9%
RecJPQ-SVD	0.122^+	2.9%

RecJPQ only increased the model size for SASRec on MovieLens-1M due to the dataset's small item count. The overhead of storing centroid embeddings and the codebook eclipses the benefit of compressing the embeddings table. Using RecJPQ with smaller embeddings might reduce the model size without affecting performance on this dataset (see Section 5.2.3).

On the other hand, we observe from Table 4 and Table 5 that the choice of the optimal strategy depends on both the model and the dataset. For example, on MovieLens-1M, the choice of the strategy is not important, and in all cases, RecJPQ versions of the models are statistically indistinguishable from the base for all models. On the larger Booking dataset, the choice of the best strategy is model-dependent. For BERT4Rec, the best results are achieved with BPR (NDCG@10 0.375, statistically indistinguishable from the base) and SVD (NDCG@10 0.379, +0.97%, significant). At the same time, the Random strategy significantly underperforms the base configuration (NDCG@10 0.316, -15.98%) - this shows that in some cases, assigning similar codes to similar items is indeed important. However, in 2 cases, Random performs statistically significantly better than SVD and BPR. For example, Random assignments perform best on Gowalla with the SASRec base (a significant improvement of +57% over the base). SVD assignments also moderately improve the result in this case (+10%, significant). At the same time, BPR decreases the quality by a large margin on Gowalla dataset (-70%)[7]. We explain the success of the Random strategy on the Gowalla dataset as giving a larger regularisation effect (random assignments make the learning task harder, so the model has fewer chances to overfit). This suggests that the centroid assignment strategy could be treated as a hyperparameter and tuned for each model/dataset combination. However, by default, we recommend using RecJPQ with SVD strategy - in all cases, it achieves significantly better (on the Booking and Gowalla datasets) or statistically indistinguishable (on the MovieLens-1M dataset) results compared to the base model. We also note that RecJPQ with the SVD strategy is always better than the Quotient Remainder baseline (Quotient Reminder is always significantly worse than the base mode, whereas RecJPQ is better or indistinguishable).

It is also worth mentioning that we do not observe a degradation of the training efficiency when training the RecJPQ versions of the models. Indeed, while there are some fluctuations in the training time the model requires to converge, the magnitude of the required training time remains the same: for example, training of the base version of BERT4Rec requires 18.8 hours on Booking.com, and the RecJPQ-SVD version of BERT4Rec requires 16.1 hours. The training time of the SVD model (used for initial centroid assignments) in the same case is negligible compared to the training of the main model (it takes approximately a minute). Inference time is also unaffected (e.g. on Gowalla, full evaluation across the 86k users requires 10 minutes for both "base" and RecJPQ versions of the models).

[7] The percentages for the Gowalla dataset seem large because this dataset is difficult: it has the largest number of items and the largest proportion of long-tail items.



(a) MovieLens-1M

(b) Gowalla

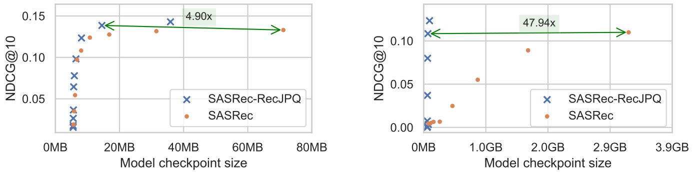
Figure 3: RecJPQ performance while varying embedding size d and the number of centroids per item m.

In summary, in answer to RQ1, we conclude that RecJPQ achieves large model compression levels. The achieved compression is particularly impressive on datasets with large catalogues (like Gowalla). The compression does not depend on the centroid assignment strategy. However, the centroid assignment strategy greatly affects the model performance. The effect is model- and dataset-dependent, so the strategy should be treated as a hyperparameter. However, the SVD is a safe choice, as it always provides results that are comparable (i.e statistically indistinguishable) or better than the base model.

To answer RQ2, we perform a grid search over embedding size and code length on the MovieLens-1M and Gowalla datasets. We use SASRec as the backbone (the only model that can be easily trained on Gowalla) and apply the SVD centroid assignment strategy. We select the embedding size d from $\{2^0, 2^1, 2^2, ..., 2^9\}$ and code length m from $\{2^0, 2^1, 2^2, ..., 2^8\}$. Note that $m \leq d$, as RecJPQ splits each embedding of size d into m sub-embeddings.

Figure 3 illustrates the results of the grid search. The figure shows the NDCG@10 of the SASRec-RecJPQ model for each combination of code length (x-axis) and embedding size (y-axis), in the form of a heatmap for both datasets. As we can see from the figure, a larger embedding size generally positively affects the model performance. This result echoes similar findings of a recent reproducibility paper [43]; however, interestingly, in the RecJPQ case, increasing embedding dimensionality does not change the amount of information we store per each item, as the length of the code defines it rather than the embedding size. Instead, it increases model capacity increasing the amount of information that can be stored in each centroid, allowing centroids to account for more item characteristics. For example, on Gowalla, the largest embedding we can train using base SASRec is 128-dimensional embedding, while with RecJPQ, we can train the model even with 512-dimensional embeddings.

On the other hand, larger code lengths is not always helpful. As we discussed in Section 4.2, RecJPQ forces the model to share parts of embeddings with other items and therefore acts as a regularisation mechanism. Shorter code length forces items to share more information, and therefore it causes a stronger regularisation effect. As we can see, on the less sparse MovieLens-1M – where all items have more than five interactions – regularisation is not an issue, and longer codes are beneficial. For example, the best result is achieved with 512-dimensional embeddings and a code of length 128 (NDCG@10 0.14). In contrast, for Gowalla, where most items are long-tail items with less than five interactions (hence the embeddings of these items should be regularised), the best NDCG is achieved with the code of length 8 (NDCG@10 0.12). The fact that the model can perform better with shorter codes confirms that RecJPQ can work as a regularisation technique.



(a) MovieLens-1M

(b) Gowalla

Figure 4: Model Performance/Model Size tradeoff for SASRec and SASRec-RecJPQ.

In short, in answer to RQ2, we conclude that larger embeddings are generally beneficial for model performance. However, the sparser Gowalla dataset benefits from shorter code lengths, due to the regularisation effect of parameter sharing brought by RecJPQ.

5.2.3 RQ3. Size-Performance tradeoff. To address our last research question, we analyse the trade-off between model checkpoint size and NDCG@10 achieved by the model when trained with different embedding sizes. We select the embedding size from {1, 2, 4, 8, 16, 32, 64, 128, 256, 512} and train the original versions of SASRec and SASRec-RecJPQ with the SVD strategy on the MovieLens-1M and Gowalla datasets. For RecJPQ, we select code length m optimal for the dataset/embedding size pair (according to grid search from Section 5.2.2). For the original SASRec on Gowalla, we only train up to the embedding size of 128 (larger embedding sizes cannot be trained on our GPUs’ memory).

Figure 4 illustrates the tradeoff between model checkpoint size and NDCG@10 for both SASRec and SASRec-RecJPQ on the two datasets. Each point on the figure corresponds to one embedding size. As can be seen from the table, a larger model size (corresponding to larger embeddings) leads to better performance for both SASRec and SASRec-RecJPQ (this echoes findings in the previous research question). However, SASRec-RecJPQ’s performance grows much faster with increasing model size than observed for vanilla SASRec. For example, the largest vanilla SASRec model achieves roughly the same performance as the 4.9× smaller SASRec-RecJPQ version of the model (71MB vs. 15 MB). This effect is even more prominent in Gowalla, where the number of items is larger: the largest SASRec model achieves roughly the same performance as the 47.94× smaller SASRec-RecJPQ model (3.2GB vs. 69MB).

Overall in answer to RQ3, we conclude that while larger models benefit model performance, RecJPQ improves this tradeoff by a large margin (i.e. to achieve the same performance, RecJPQ requires much fewer parameters than the original model). This effect is more markedly pronounced for the larger Gowalla dataset.

6 CONCLUSIONS

In this paper, we discussed the challenge of training sequential recommender systems with large datasets, primarily due to the large item embedding tensor. Existing compression methods have limitations, leading to the proposed method, RecJPQ, based on Joint Product Quantisation. Our evaluation of RecJPQ on three datasets resulted in significant model size reduction, e.g., 47.94×

compression of the SASRec model on the Gowalla dataset. Additionally, RecJPQ serves as a model regulariser, improving model quality, with SASRec-RecJPQ using SVD strategy outperforming the original SASRec model (+35% NDCG@10 on Booking, +10% on Gowalla). This paper's method could bridge the gap between academic research, especially on transformer-based architectures, and large-scale production recommender systems of large companies.

REFERENCES

[1] Jan Van Balen and Mark Levy. 2019. PQ-VAE: Efficient Recommendation Using Quantized Embeddings. In *Proc. RecSys*.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models Are Few-Shot Learners. In *Proc. NeurIPS*, Vol. 33. 1877–1901.

[3] Christopher Burges. 2010. From RankNet to LambdaRank to LambdaMART: An overview. *Learning* 11 (2010).

[4] Rocío Cañamares and Pablo Castells. 2020. On Target Item Sampling in Offline Recommender System Evaluation. In *Proc. RecSys*. 259–268.

[5] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. 2011. Friendship and mobility: user movement in location-based social networks. In *Proc. KDD*. 1082–1090.

[6] Kyunghyun Cho and Bart van Merrienboer. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. (2014). arXiv:1409.1259 [cs, stat]

[7] Alexander Dallmann, Daniel Zoller, and Andreas Hotho. 2021. A Case Study on Sampling Strategies for Evaluating Neural Sequential Item Recommendation Models. In *Proc. RecSys*. 505–514.

[8] Kalyanmoy Deb and Kalyanmoy Deb. 2014. Multi-objective Optimization. In *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. 403–449.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc. NAACL-HLT*. 4171–4186.

[10] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2014. Optimized Product Quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 4 (2014), 744–755.

[11] Dmitri Goldenberg and Pavel Levin. 2021. Booking.com Multi-Destination Trips Dataset. In *Proc. SIGIR*. 2457–2462.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

[13] Robert M. Gray. 1984. Vector Quantization. *IEEE Assp* 1, 2 (1984).

[14] N. Halko, P. G. Martinsson, and J. A. Tropp. 2011. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. 53, 2 (2011), 217–288.

[15] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. 5, 4 (2015), 19:1–19:19.

[16] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *Proc. WWW*. 173–182.

[17] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent Neural Networks with Top-k Gains for Session-based Recommendations. In *Proc. CIKM*. 843–852.

[18] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-Based Recommendations with Recurrent Neural Networks. In *Proc. ICLR*.

[19] Yupeng Hou, Zhankui He, Julian McAuley, and Wayne Xin Zhao. 2023. Learning Vector-Quantized Item Representation for Transferable Sequential Recommenders. In *Proc. WWW*. 1162–1171.

[20] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning Deep Structured Semantic Models for Web Search Using Clickthrough Data. In *Proc. CIKM*. 2333–2338.

[21] Gautier Izacard, Fabio Petroni, Lucas Hosseini, Nicola De Cao, Sebastian Riedel, and Edouard Grave. 2020. A Memory Efficient Baseline for Open Domain Question Answering. arXiv:2012.15156 [cs]

[22] Herve Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011), 117–128.

[23] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2021), 535–547.

[24] Wang-Cheng Kang, Derek Zhiyuan Cheng, Ting Chen, Xinyang Yi, Dong Lin, Lichan Hong, and Ed H. Chi. 2020. Learning Multi-granular Quantized Embeddings for Large-Vocab Categorical Features in Recommender Systems. In *Proc. WWW*. 562–566.

[25] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. In *Proc. ICDM*. 197–206.

[26] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proc. EMNLP*. arXiv:2004.04906 [cs]

[27] Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *Proc. SIGIR*. 39–48.

[28] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (2009), 30–37.

[29] Walid Krichene and Steffen Rendle. 2022. On sampled metrics for item recommendation. *Commun. ACM* 65, 7 (2022), 75–83.

[30] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2020. Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, and Improvement. *IEEE Transactions on Knowledge and Data Engineering* 32, 8 (2020), 1475–1488.

[31] Defu Lian, Haoyu Wang, Zheng Liu, Jianxun Lian, Enhong Chen, and Xing Xie. 2020. LightRec: A Memory and Search-Efficient Recommender System. In *Proc. WWW*. 695–705.

[32] J. MacQueen. 1967. Some Methods for Classification and Analysis of Multivariate Observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. Vol. 5.1. 281–298.

[33] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2020), 824–836.

[34] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed Precision Training. In *Proc. ICLR*.

[35] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. 2019. Deep Learning Recommendation Model for Personalization and Recommendation Systems. arXiv:1906.00091 [cs]

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

[37] Aleksandr Petrov and Craig Macdonald. 2022. Effective and Efficient Training for Sequential Recommendation Using Recency Sampling. In *Proc. RecSys*. 81–91.

[38] Aleksandr Petrov and Craig Macdonald. 2022. A Systematic Review and Replicability Study of BERT4Rec for Sequential Recommendation. In *Proc. RecSys*. 436–447.

[39] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Languaga Models are Unsupervised Multitask Learners. *OpenAI blog* (2019).

[40] Rajput, Shashank, Mehta, Nikhil, Singh, Anima, Keshavan, Raghunandan, Vu, Trung, Heldt, Lukasz, Hong, Lichan, Tay, Yi, Tran, Vinh Q., Samost, Jonah, Kula, Maciej, Chi, Ed H., and Sathiamoorthy, Maheswaran. 2023. Recommender Systems with Generative Retrieval. arXiv:2305.05065 [cs.IR]

[41] Steffen Rendle. 2022. Item Recommendation from Implicit Feedback. In *Recommender Systems Handbook*. 143–171.

[42] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proc. UAI*.

[43] Steffen Rendle, Walid Krichene, Li Zhang, and Yehuda Koren. 2022. Revisiting the Performance of iALS on Item Recommendation Benchmarks. In *Proc. RecSys*. 427–435.

[44] Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022. ColBERTv2: Effective and Efficient Retrieval via Lightweight Late Interaction. arXiv:2112.01488 [cs]

[45] Ozan Sener and Vladlen Koltun. 2018. Multi-Task Learning as Multi-Objective Optimization. In *Advances in Neural Information Processing Systems*, Vol. 31.

[46] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. 2020. Compositional Embeddings Using Complementary Partitions for Memory-Efficient Recommendation Systems. In *Proc. KDD*. 165–175.

[47] Shaoyun Shi, Weizhi Ma, Min Zhang, Yongfeng Zhang, Xinxing Yu, Houzhi Shan, Yiqun Liu, and Shaoping Ma. 2020. Beyond User Embedding Matrix: Learning to Hash for Modeling Large-Scale Users in Recommendation. In *Proc. SIGIR*. 319–328.

[48] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. In *Proc. CIKM*. 1441–1450.

[49] Jiaxi Tang and Ke Wang. 2018. Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding. In *Proc. WSDM*. 565–573.

[50] 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 *Proc. NeurIPS*.

[51] Feng Wang, Miaomiao Dai, Xudong Li, and Liquan Pan. 2022. Compressing Embedding Table via Multi-dimensional Quantization Encoding for Sequential

Recommender Model. In *Proc. ICCIP*. 234–239.

[52] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & Cross Network for Ad Click Predictions. In *Proc. KDD*. 1–7.

[53] Kilian Weinberger, Anirban Dasgupta, Josh Attenberg, John Langford, and Alex Smola. 2010. Feature Hashing for Large Scale Multitask Learning. arXiv:0902.2206 [cs]

[54] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. HuggingFace's Transformers: State-of-the-art Natural Language Processing. arXiv:1910.03771 [cs]

[55] Xin Xia, Junliang Yu, Qinyong Wang, Chaoqun Yang, Nguyen Quoc Viet Hung, and Hongzhi Yin. 2023. Efficient On-Device Session-Based Recommendation. *ACM Transactions on Information Systems* (2023).

[56] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers. In *Proc. NeurIPS*, Vol. 35. 27168–27183.

[57] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M. Jose, and Xiangnan He. 2019. A Simple Cenvolutional Generative Network for Next Item Recommendation. In *Proc. WSDM*. 582–590.

[58] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. 2021. Jointly Optimizing Query Encoder and Product Quantization to Improve Retrieval Performance. In *Proc. CIKM*. 2487–2496.

[59] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. 2021. Optimizing Dense Retrieval Model Training with Hard Negatives. In *Proc. SIGIR*. 1503–1512.

[60] Yin Zhang, Derek Zhiyuan Cheng, Tiansheng Yao, Xinyang Yi, Lichan Hong, and Ed H. Chi. 2021. A Model of Two Tales: Dual Transfer Learning Framework for Improved Long-tail Item Recommendation. In *Proc. WWW*. 2220–2231.

[61] Xun Zhou, Jing He, Guangyan Huang, and Yanchun Zhang. 2012. A Personalized Recommendation Algorithm Based on Approximating the Singular Value Decomposition (ApproSVD). In *Proc. WI-IAT*, Vol. 2. 458–464.