Joint Learning of Deep Retrieval Model and Product Quantization based Embedding Index

Han Zhang [1], Hongwei Shen [2], Yiming Qiu [1], Yunjiang Jiang [2], Songlin Wang [1], Sulong Xu [1], Yun Xiao [2], Bo Long [1] and Wen-Yun Yang [2*]

[1] JD.com, Beijing, China; [2] JD.com Silicon Valley Research Center, Mountain View, CA, United States

{zhanghan33, hongwei.shen1, qiuyiming3, yunjiang.jiang, wangsonglin3, xusulong, xiaoyun1, bo.long, wenyun.yang}@jd.com

Abstract

Embedding index that enables fast approximate nearest neighbor (ANN) search, serves as an indispensable component for state-of-the-art deep retrieval systems. Traditional approaches, often separating the two steps of embedding learning and index building, incur additional indexing time and decayed retrieval accuracy. In this paper, we propose a novel method called *Poeem*, which stands for **p**roduct quantizati**o**n based **e**mbedding index jointly trained with deep r**e**trieval **m**odel, to unify the two separate steps within an end-to-end training, by utilizing a few techniques including the gradient straight-through estimator, warm start strategy, optimal space decomposition and Givens rotation. Extensive experimental results show that the proposed method not only improves retrieval accuracy significantly but also reduces the indexing time to almost none. We have open sourced our approach[1] for the sake of comparison and reproducibility.

CCS Concepts

• **Information systems** → **Novelty in information retrieval**; **Information retrieval**; **Retrieval models and ranking**;

Keywords

Embedding index, neural network, information retrieval

ACM Reference Format:

Han Zhang [1], Hongwei Shen [2], Yiming Qiu [1], Yunjiang Jiang [2], Songlin Wang [1], Sulong Xu [1],, Yun Xiao [2], Bo Long [1] and Wen-Yun Yang [2*]. 2021. Joint Learning of Deep Retrieval Model and Product Quantization based Embedding Index. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '21), July 11–15, 2021, Virtual Event, Canada.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3404835.3462988

1 Introduction

Various types of indexes play an indispensable role in modern computational systems to enable fast information retrieval. As a traditional one, inverted index [8] has been widely used in web search, e-commerce search, recommendation and advertising in the past few decades. Recently, with the advent of the deep learning era, embedding indexes [6, 15], which embed user/query and item in a latent vector space, show excellent performance in many industrial retrieval systems [14, 20, 26]. Embedding index enjoys several appealing advantages: a) the embeddings can be learned to optimize downstream retrieval task of interests, and b) efficient algorithms for maximum inner product search (MIPS) or approximate nearest neighbors (ANN), such as LSH [7], Annoy [3] and state-of-the-art product quantization (PQ) based approaches [11, 17, 18], can be leveraged to retrieve items in a few milliseconds.

Embedding indexes, however, also suffer from a few drawbacks. The major one resides in the separation between model training and index building, which results in extra index building time and decayed retrieval accuracy. Thus, recently, there is a new trend of abandoning separately built embedding indexes but embracing jointly learned structural indexes which have shown improved performance than the former. In general, the approaches with joint learning structure can be summarized into two types, tree based ones [27, 28] and PQ based ones [4, 19, 25]. Those tree based approaches normally require special approximate training techniques, whose complications slow down their wide adoptions. Those existing PQ based approaches are designed for only small computer vision tasks, such as retrieval from tens of thousands of images, thus inapplicable to large scale information retrieval tasks with at least millions of items in a real-world industrial retrieval system.

In this paper, we advance the approach of product quantization based embedding index jointly trained with deep retrieval model. It is not trivial and we have to overcome a few hurdles by appropriate techniques: 1) the quantization steps, as the core of PQ based embedding indexes, have non-differentiable operations, such as arg min, which disable the standard back propagation training. Thus, we leverage the gradient straight-through estimator [2] to bypass the non-differentiability in order to achieve end-to-end training. 2) The randomly initialized quantization centroids lead to very sparse centroid assignments, low parameter utilization and consequentially higher quantization distortion. Thus, we introduce a warm start strategy to achieve more uniform distributed centroid assigments. 3) The standard optimized product quantization (OPQ) [9] algorithm, which rotates the space by an orthonormal matrix to further reduce PQ distortion, can not iteratively run together with the joint model training. Thus, we develop a steepest block coordinate descent algorithm with Givens rotations [10] to learn the orthonormal matrix in this end-to-end training.

As a result, our proposed method *Poeem*, which stands for **p**roduct quantizati**o**n based **e**mbedding index jointly trained with deep r**e**trieval **m**odel, enjoys advantages of almost no index building

[1] https://github.com/jdcomsearch/poeem

* Corresponding author

SIGIR '21, July 11–15, 2021, Virtual Event, Canada

ACM ISBN 978-1-4503-8037-9/21/07…$15.00
https://doi.org/10.1145/3404835.3462988

time and no decayed retrieval accuracy. Aiming at a more practical approach ready for wide adoptions, our method is capsulated in a standalone indexing layer, which can be easily plugged into any embedding retrieval models.

2 Method

2.1 Revisiting Retrieval Model

A standard embedding retrieval model, as shown at the left side of Figure 1, is composed of a query tower Q and an item tower S. Thus, for a given query q and an item s, the output of the model is

$$f(q, s) = F(Q(q), S(s)) \tag{1}$$

where $Q(q) \in \mathbb{R}^d$ denotes query tower output embeddings in d-dimensional space. Similarly, $S(s) \in \mathbb{R}^d$ denotes an item tower output in the same dimensional space. The scoring function $F(.,.)$, usually inner product or cosine value, computes the final score between the query and item, and has been proven to be successful in many applications [6, 14, 26]. After the model is trained, traditional approaches still require an additional step – computing item embeddings and building an embedding index for them – before it is ready for online serving. However, this additional step not only spends extra time to build index, but also causes decay of recall rate [17]. In the following sections, we will present our approach to the rescue of these two shortcomings.

2.2 Embedding Indexing Layer

Figure 1 illustrates an overview of the proposed embedding indexing layer inside a typical deep retrieval model. Formally, the indexing layer defines a *full quantization function* $\mathcal{T} : \mathbb{R}^d \to \mathbb{R}^d$ that maps an input embedding $\mathbf{x}$ to an output embedding $\mathcal{T}(\mathbf{x})$, which can be decomposed into four functions: a *coarse quantization* function ψ, a *product quantization* function ϕ and a *decoder* function ρ, and a *rotation* function with an orthonormal matrix R. Note that we are using orthonormal matrix and rotation matrix interchangeably. Now let's explain in detail these operations in the following sections.

The **coarse quantization** function $\psi : \mathbb{R}^d \to \{1, \cdots, J\}$, which maps a continuous vector $\mathbf{x}$ into a J-way discrete *coarse code* r by a coarse centroid matrix $\mathbf{v} \in \mathbb{R}^{J \times d}$, can be defined as follows

$$\psi(\mathbf{x}) = r = \arg\min_k \text{dist}(\mathbf{x}, \mathbf{v}_k) \tag{2}$$

where $\text{dist}(.,.)$ stands for a distance measure, typically L2 distance, between two vectors, and the vector $\mathbf{v}_k \in \mathbb{R}^d$ stands for the k-th centroid, *i.e.*, k-th row in $\mathbf{v}$. It is not hard to see that this function just finds the nearest centroid for a given input vector $\mathbf{x}$, and outputs the centroid index. Thus, a coarse quantization residual exists as

$$\mathbf{y} = \mathbf{x} - \mathbf{v}_r.$$

We will further deal with it by product quantization.

The **product quantization (PQ)** function $\phi : \mathbb{R}^d \to \{1, \cdots, K\}^D$, which maps the above residual vector $\mathbf{y}$ into a K-way D-dimensional *PQ code* $\mathbf{c}$, can be defined as follows. First, we denote the vector $\mathbf{y}$ as concatenation of D subvectors:

$$\mathbf{y} = [\mathbf{y}^1, \ldots, \mathbf{y}^D]. \tag{3}$$

For simplicity, it is a common practice to divide the original dimension d evenly. In other words, each subvector is $\mathbf{y}^j \in \mathbb{R}^{d/D}$. The Cartesian product $C = C^1 \times \ldots \times C^D$ is the set in which a *PQ code* $\mathbf{c} \in C$ is formed by concatenating the D sub-codewords: $\mathbf{c} = [c^1, \ldots, c^D]$, with each $c^j \in C^j$. Note that the D codes can be computed independently as follows

$$c^j = \arg\min_k \text{dist}(\mathbf{y}^j, \mathbf{v}_k^j) \tag{4}$$

where $\mathbf{v}_k^j$ stands for the k-th centroid for the j-th subvectors. Finally, we can define the function ϕ as follows

$$\phi(\mathbf{y}) = [c^1, \ldots, c^D]. \tag{5}$$

The **decoder** function

$$\rho : \left(\{1, \cdots, J\}, \{1, \cdots, K\}^D\right) \to \mathbb{R}^d$$

that maps the discrete codes back to a continuous vector, can be formally defined as follows,

$$\rho(r, [c^1, \ldots, c^D]) = \mathbf{v}_r + [\mathbf{v}_{c^1}^1, \ldots, \mathbf{v}_{c^D}^D]$$

which sums over the coarse quantized vector $\mathbf{v}_c$, and the concatenation of product quantized vector $\mathbf{v}_{c^j}^j$.

The **rotation** function by an orthonormal matrix R, also known as optimized product quantization (OPQ) [9], rotates the input embedding $\mathbf{x}$ by

$$\mathbf{x}' = R\mathbf{x}$$

to further reduce the quantization distortion. The rotation allows product quantization to operate in a transformed space, thus relaxes the constraints on PQ centroids. For example, note that any reordering of the dimensions can be represented by a rotation matrix.

After the above decoder function, we can also "rotate back" to the original space by R^{-1}, so that this rotation is fully encapsulated inside the standalone indexing layer. Also note that the inverse of an orthonormal matrix is just its transpose [10], *i.e.*, $R^{-1} = R^\top$, which is very cheap to compute.

Finally, with the above four functions, we can now define the **full quantization** function $\mathcal{T}$ as follows

$$\mathcal{T}(\mathbf{x}) = R^\top \rho(\psi(\mathbf{x}'), \phi(\mathbf{x}' - \mathbf{v}_{\psi(\mathbf{x}')})) \tag{6}$$

where $\mathbf{x}' = R\mathbf{x}$.

2.3 Training Algorithm

2.3.1 Loss Function Straightforward optimization of the above embedding indexing layer by the standard back propagation algorithm is infeasible, as the arg min operation in Eqs. (2) and (4) is non-differentiable. In fact, this is the difficulty that prevents previous researchers from training embedding indexes jointly with retrieval model. Here we propose to leverage the gradient straight-through estimator [2] by adjusting the original quantization function $\mathcal{T}$ in Eq. (6) as follows

$$\mathcal{H}(\mathbf{x}) = \mathbf{x} - \text{sg}(\mathbf{x} - \mathcal{T}(\mathbf{x})) \tag{7}$$

where sg is the *stop gradient* operation. During the forward pass, the quantized embedding $\mathcal{T}(\mathbf{x})$ is emitted; During the backward pass, the gradient is passed to the original embedding $\mathbf{x}$ directly, bypassing the entire quantization function $\mathcal{T}$. Note that Eq. (7) only approximates gradient for original embedding $\mathbf{x}$, does not update the quantization parameters (centroids) in the back propagation. Similar as previous works [5, 22], we add a regularization term into the loss function to minimize the quantization distortion as follows,

$$\mathcal{L}_{reg} = \sum_{\mathbf{x}} ||\mathcal{T}(\mathbf{x}) - \text{sg}(\mathbf{x})||^2$$

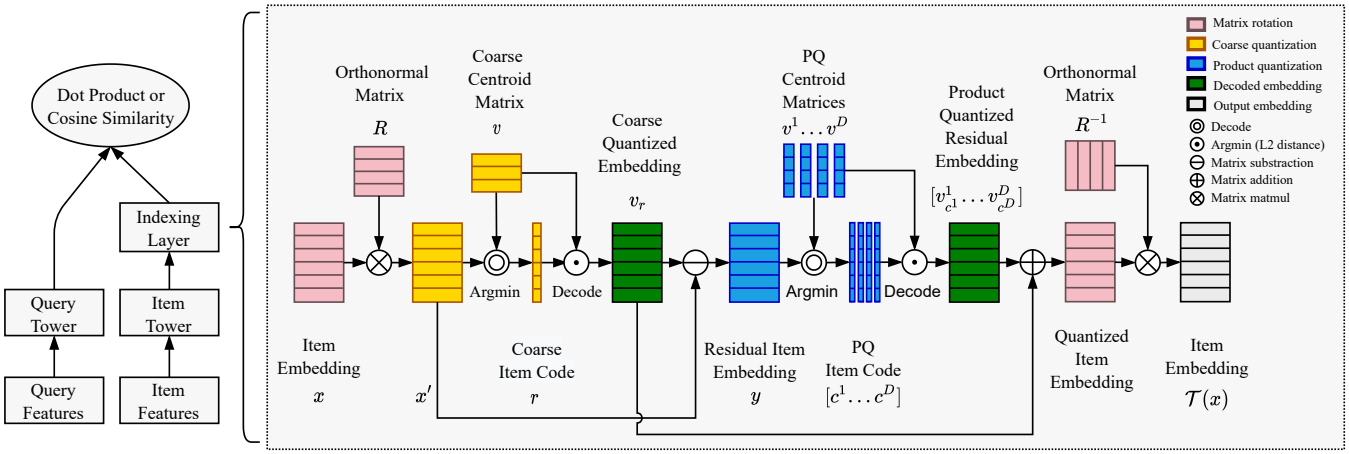


Figure 1: Illustration of a retrieval model with an embedding indexing layer, which is composed of four steps, coarse quantization function ψ (yellow), product quantization function ϕ (blue), decoder function ρ (green), and givens rotation (pink).

which essentially updates the coarse and PQ centroids to be arithmetic mean of their members.

2.3.2 Warm Start Centroids In practice, we find that random initialization of the quantization parameters (the centroids $\mathbf{v}_k$ and $\mathbf{v}_k^j$ in Eqs. (2) and (4)) results in very sparse centroid assignments, which consequentially hurts the retrieval quality (see experiments, Section 3.3). Fortunately, we are able to overcome this hurdle by simply warm starting the centroids. In detail, we train the model without the above indexing layer for a number of warm-start steps, then we plug in the indexing layer with centroids initialized by a standard k-means clustering.

2.3.3 Givens Rotation Learning the optimal rotation matrix with fixed embeddings $\mathbf{x}$, previously formulated as the Orthogonal Procrustes problem [9, 21], is incompatible with our end-to-end training, since the embeddings $\mathbf{x}$ is not fixed. Owing to the previous work [16] which shows that any rotation matrix can be represented by a product of Givens rotations [10], we are able to jointly learn the rotation matrix by a steepest block coordinate descent algorithm [1, 24] with each coordinate as a specific Givens rotation within two axes.

3 Experiment

3.1 Setup

In Table 1, we evaluate our methods on three datasets: a JD.com search click log data where a user input query is used to retrieve items, and two public datasets of MovieLens [12] and Amazon Books [13] where user historical behaviors are used to retrieve next behavior. We evaluate retrieval accuracy by precision@k (p@k) and recall@k (r@k), which are standard retrieval quality metrics.

We implement Poeem in Tensorflow 1.15, and train models in a single machine with 4 Nvidia V100 GPU cards, and evaluate all methods in a 48-cores CPU machine. A typical set of parameters are as follows: a two-tower retrieval model with cosine scoring and hinge loss of margin 0.1, embedding size 512 for private dataset, 128 for MovieLens and 128 for Amazon Books, Adagrad optimizer with learning rate 0.01, and batch size 1024.

Table 1: Dataset statistics.

Dataset	# Examples	# Users	# Items
Private	9,989,135	1,031,583	1,541,673
MovieLens	9,939,873	129,797	20,709
Amazon Books	8,654,619	294,739	1,477,922

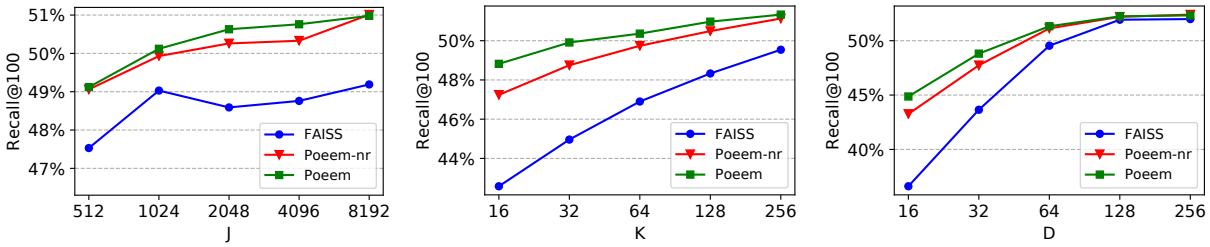
Table 2: Comparison between the baseline methods and Poeem.

Method	Private		MovieLens		Amazon Books	
	p@100	r@100	p@100	r@100	p@100	r@100
LSH	1.28%	25.64%	7.48%	34.50%	0.51%	4.11%
Annoy	1.24%	23.42%	7.85%	35.53%	0.72%	5.71%
ScaNN	2.25%	47.30%	7.91%	36.50%	0.72%	5.71%
Faiss	2.37%	49.54%	8.02%	36.72%	0.68%	5.46%
Poeem	**2.42%** (+0.05%)	**51.13%** (+1.59%)	**8.22%** (+0.20%)	**37.48%** (+0.76%)	**0.73%** (+0.05%)	**5.90%** (+0.44%)

3.2 Comparison with Offline Indexing

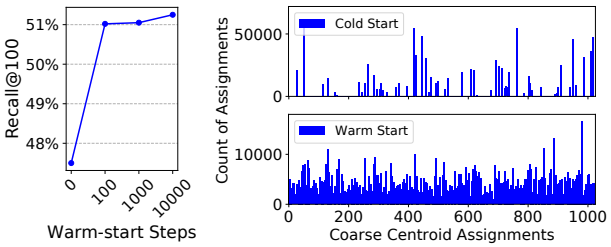
Table 2 presents comparison results with the baseline methods, offline indexing with LSH [7], Annoy [3], ScaNN [11] and Faiss [18]. To conduct fair comparisons, Faiss uses the index type of IVFPQ which shares the same product quantization parameters as Poeem. Since other baselines do not have similar product quantization structure, we choose parameters which have the same retrieval time cost as Poeem. We can observe that the proposed Poeem outperforms all the baseline methods by precision@100 and recall@100.

Figure 2 shows the recall@100's trend with varying values of J, K and D, with or without R. It is clear that Poeem outperforms the baseline for all different parameter values. Firstly, the rotation matrix has a positive impact on recall@100. Secondly, we can see a trend that, with smaller parameters, the gap between the proposed method and baseline is enlarged. This indicates that the best scenario of the proposed method is for indexes of large dataset where large compression ratio is required. Thirdly, the performance gap almost diminishes when D approaches to d (512 in this case), which, however, is actually impractical but just for reference purpose. Since if $D = d$, product quantization reduces to numerical quantization whose low compression ratio is infeasible for large data. Note that a proper choice of J is necessary to prevent imbalance distribution of coarse codes and consequentially unstable retrieval time.



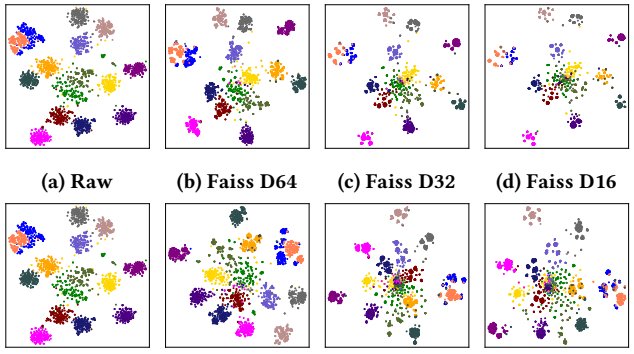
(a) Parameter J (b) Parameter K (c) Parameter D

Figure 2: Comparison between baseline (offline indexing with Faiss) and the proposed Poeem for different values of J, K and D. Poeem-nr stands for the variant of Poeem without using rotation matrix.



(a) (b)

Figure 3: Effects of warm start centroids. (a) recall@100 within different warm-start steps, (b) histogram of coarse centroid assignments with cold start (upper) and warm start (lower).



(a) Raw (b) Faiss D64 (c) Faiss D32 (d) Faiss D16

(e) Raw (f) Poeem D64 (g) Poeem D32 (h) Poeem D16

Figure 4: t-SNE visualizations of Faiss and Poeem item embeddings with varying parameter D.

3.3 Effects of Warm Start Centroids

Figure 3a illustrates the effect of warm-start centroids, where we can observe that the retrieval quality significantly improves with warm started centroids over cold started centroids, but only slightly improves with more warm-up steps. Note that the warm-up steps 0 corresponds to cold start where the centroids are actually randomly initialized, which results in sparse utilization of both coarse centroids and PQ centroids. In a typical run as shown in Figure 3b, the coarse centroid assignments with warm start distribute more uniformly than those with cold start. In more detail, only 67 out of 1024 cold start centroids are actually used by coarse quantization, which spikes to 1004 out of 1024 for warm start centroids.

3.4 Computational Cost

Extensive experiments show that the training time of the proposed method slightly increases by around 1%. And we do not observe any significant memory increasing since the coarse centroids and PQ centroids are negligible compared to the other part of retrieval model. As for the indexing time, the proposed method only needs to save items' coarse codes and PQ codes into index file. Thus, for 1 million 512-dimensional embeddings, Poeem only needs 5 seconds of indexing time compared to 641 seconds of Faiss, 101 seconds of ScaNN, 93 seconds of Annoy, and 4.6 seconds of LSH.

3.5 Ablation Study

To have an intuitive understanding of how Poeem works, Figure 4 shows 2-D t-SNE [23] visualizations on randomly chosen items from top 15 popular categories in our private dataset. Figures 4a and 4e show the same figure of raw item embedding distribution from the two-tower retrieval model, which serves as the best scenarios since there is no quantization distortion. Figures 4b to 4d illustrate the progressive quantization distortion effect with decreasing parameter D for Faiss, and Figures 4f to 4h illustrate that for Poeem.

For both Faiss and Poeem, we can observe that product quantization has the effect of "shrinking" and "collapsing" those normally distributed clusters, with the progress that the well distributed cluster is first divided into sub-clusters, which then further shrink into much smaller ones. This effect makes sense if we consider that the product quantization forces those embeddings to share the same set of subvector centroids. Thus, those nearby embeddings are "pulled" closer if they are assigned to the same subvector centroid.

With the above observation, we can now see that the proposed method Poeem improves on the baseline Faiss, by slowing down the progress of "shrinking" and "collapsing" those clusters. While the parameter D is decreasing, Poeem maintains the cluster's normally distributed shape much better than Faiss, especially for those outskirt clusters where the comparison is clear.

4 Conclusion

In this paper, we have proposed a novel method called Poeem to learn embedding indexes jointly with any deep retrieval models. We introduce an end-to-end trainable indexing layer composed of space rotation, coarse quantization and product quantization operations. Experimental results show that the proposed method significantly improves retrieval metrics over traditional offline indexing methods, and reduces the index building time from hours to seconds.

References

[1] Amir Beck and Luba Tetruashvili. 2013. On the convergence of block coordinate descent type methods. *SIAM journal on Optimization* 23, 4 (2013), 2037–2060.

[2] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. In *arXiv*. 1308.3432.

[3] Erik Bernhardsson. 2018. *Annoy: Approximate Nearest Neighbors in C++/Python*. https://pypi.org/project/annoy/ Python package version 1.13.0.

[4] Yue Cao, Mingsheng Long, Jianmin Wang, Han Zhu, and Qingfu Wen. 2016. Deep quantization network for efficient image retrieval. In *AAAI*, Vol. 30.

[5] Ting Chen, Lala Li, and Yizhou Sun. 2020. Differentiable product quantization for end-to-end embedding compression. In *ICML*. 1617–1626.

[6] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *RecSys*. 191–198.

[7] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*. 253–262.

[8] Jeffrey Dean. 2009. Challenges in building large-scale information retrieval systems. In *WSDM*, Vol. 10.

[9] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization for approximate nearest neighbor search. In *CVPR*. 2946–2953.

[10] Gene H. Golub and Charles F. Van Loan. 1996. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, USA.

[11] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating large-scale inference with anisotropic vector quantization. In *ICML*. 3887–3896.

[12] F Maxwell Harper and Joseph A Konstan. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2015), 1–19.

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

[14] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based retrieval in facebook search. In *SIGKDD*. 2553–2561.

[15] 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 *CIKM*. 2333–2338.

[16] Adolf Hurwitz. 1963. Ueber die erzeugung der invarianten durch integration. In *Mathematische Werke*. Springer, 546–564.

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

[18] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* (2019).

[19] Benjamin Klein and Lior Wolf. 2017. In defense of product quantization. *arXiv preprint arXiv:1711.08589* 2, 3 (2017), 4.

[20] Chao Li, Zhiyuan Liu, Mengmeng Wu, Yuchi Xu, Huan Zhao, Pipei Huang, Guoliang Kang, Qiwei Chen, Wei Li, and Dik Lun Lee. 2019. Multi-Interest Network with Dynamic Routing for Recommendation at Tmall. In *CIKM*. 2615–2623.

[21] Peter Schönemann. 1966. A generalized solution of the orthogonal procrustes problem. *Psychometrika* 31, 1 (1966), 1–10.

[22] Aaron Van Den Oord, Oriol Vinyals, et al. 2017. Neural discrete representation learning. In *NIPS*. 6306–6315.

[23] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9 (2008), 2579–2605.

[24] Stephen J Wright. 2015. Coordinate descent algorithms. *Mathematical Programming* 151, 1 (2015), 3–34.

[25] Tan Yu, Junsong Yuan, Chen Fang, and Hailin Jin. 2018. Product quantization network for fast image retrieval. In *ECCV*. 186–201.

[26] Han Zhang, Songlin Wang, Kang Zhang, Zhiling Tang, Yunjiang Jiang, Yun Xiao, Weipeng Yan, and Wen-Yun Yang. 2020. Towards Personalized and Semantic Retrieval: An End-to-EndSolution for E-commerce Search via Embedding Learning. *arXiv preprint arXiv:2006.02282* (2020).

[27] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. 2018. Learning tree-based deep model for recommender systems. In *SIGKDD*. 1079–1088.

[28] Jingwei Zhuo, Ziru Xu, Wei Dai, Han Zhu, Han Li, Jian Xu, and Kun Gai. 2020. Learning Optimal Tree Models under Beam Search. In *ICML*, Vol. 119. 11650–11659.