



Real-time Indexing for Large-scale Recommendation by Streaming Vector Quantization Retriever

Xingyan Bin*
binxingyan@bytedance.com
ByteDance Inc.
Beijing, China

Jianfei Cui*
cuijianfei.ies@bytedance.com
ByteDance Inc.
Beijing, China

Wujie Yan
yanwujie@bytedance.com
ByteDance Inc.
Beijing, China

Zhichen Zhao
hanshujin@bytedance.com
ByteDance Inc.
Beijing, China

Xintian Han
hanxintian@bytedance.com
ByteDance Inc.
Beijing, China

Chongyang Yan
yanchongyang.otw@bytedance.com
ByteDance Inc.
Beijing, China

Feng Zhang
feng.zhang@bytedance.com
ByteDance Inc.
Shanghai, China

Xun Zhou
zhouxun@bytedance.com
ByteDance Inc.
Beijing, China

Xiao Yang
wuqi.shaw@bytedance.com
ByteDance Inc.
Beijing, China

Zuotao Liu
michael.liu@bytedance.com
ByteDance Inc.
Shanghai, China

Abstract

Retrievers, which form one of the most important recommendation stages, are responsible for efficiently selecting possible positive samples to the later stages under strict latency limitations. Because of this, large-scale systems always rely on approximate calculations and indexes to roughly shrink candidate scale, with a simple ranking model. Most of the existing methods mainly focus on incorporating complicated ranking models. However, index structure is not improved, which also bottlenecks the whole effectiveness. In this paper, we propose a novel index structure: streaming Vector Quantization model, as a new generation of retrieval paradigm. Streaming VQ attaches items with indexes in real time, granting it immediacy. Moreover, through meticulous verification of possible variants, it achieves additional benefits like index balancing and reparability, enabling it to support complicated ranking models as existing approaches. Streaming VQ has been deployed and replaced all major retrievers in Douyin and Douyin Lite, resulting in remarkable user engagement gain.

CCS Concepts

• **Information systems** → **Retrieval models and ranking; Recommender systems; Retrieval effectiveness.**

Keywords

Retrieval method; Index structure; Real-time assignment

ACM Reference Format:

Xingyan Bin, Jianfei Cui, Wujie Yan, Zhichen Zhao, Xintian Han, Chongyang Yan, Feng Zhang, Xun Zhou, Xiao Yang, and Zuotao Liu. 2025. Real-time Indexing for Large-scale Recommendation by Streaming Vector Quantization Retriever. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25)*, August 3–7, 2025, Toronto, ON, Canada. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3711896.3737259>

1 Introduction

In modern recommendation systems, we face explosively growing corpus, so a cascade framework, which is composed of retrieval, pre-ranking and ranking stages has been prevalent. Among these stages, retrievers are tasked with differentiating candidates within the entire corpus but given the least time. For example, in Douyin they need to select thousands of candidates from billions of items, while the later stages shrink the candidate scale by 10 times.

However, scanning all candidates costs prohibitive computational overheads. People are thinking of cleverly traverse all candidates or only representative ones. The latter one becomes the mainstream route, and the retrieval stage relies on index structures and approximate calculations, such as Product Quantization (PQ [8]) and Hierarchical Navigable Small World (HNSW [11]). Meanwhile, user-side and item-side information is decoupled into two separate representations, and user representations are used

*equal contribution

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

KDD '25, Toronto, ON, Canada

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1454-2/2025/08
<https://doi.org/10.1145/3711896.3737259>

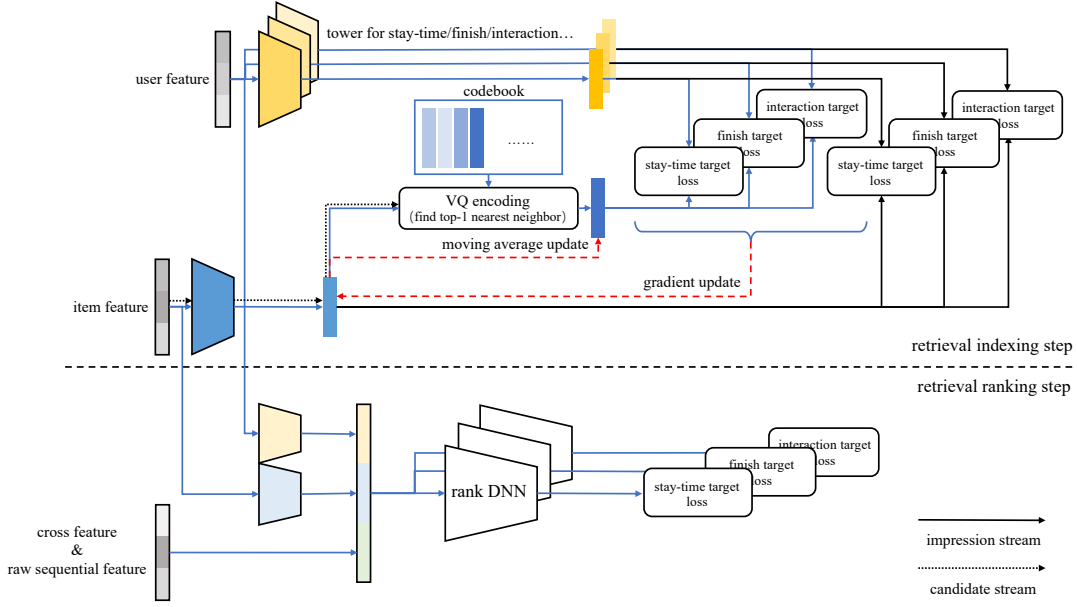


Figure 1: The training framework of the proposed streaming VQ model.

to search for relevant clusters. It results in a “two-tower” architecture [2, 7] where each tower is implemented by a Multi-layer Perceptron (MLP). Due to its remarkable ability to significantly reduce computational overheads, this method has prevailed in many industrial scenarios. Hereafter, we refer to it as “HNSW Two-tower”.

Despite its simplicity, HNSW Two-tower suffers from two drawbacks: (1) its index structure needs to be reconstructed periodically, given fixed item representations and item-index assignments. However, in a vibrant platform, new items are submitted every second and cluster semantics are changed by emerging trends, which is missed in modeling. Besides, this construction procedure is not aligned with the target of recommendation. (2) two-tower models rarely provide user-item interactions, and produce weak predictions. Unfortunately, in large-scale applications, complicated models such as MLPs have unaffordable computational costs.

Many existing methods have focused on these problems and have developed new index structures. However, they are mainly designed for affording complicated models, while neglecting critical problems of indexes themselves. Based on our practical experience, index immediacy and index balancing are equally crucial as model complication. If the index structure is seriously imbalanced, hot items gather in several indexes, causing model hardly distinguishes them.

In this paper, we propose a novel index structure, streaming Vector Quantization (streaming VQ), to improve retriever capability. Streaming VQ has the unique feature of attaching items to clusters fitly¹ in real time, allowing it to capture emerging trends as they occur. In addition, we also exhaust each variant to identify the optimal

solutions for achieving index balancing. Streaming VQ makes items within indexes distinguishable, so it is capable to generate a more compact set while maintaining excellent performance. Even though it primarily focuses on indexing step, it also supports complicated models and multi-task learning. With these innovative mechanisms, streaming VQ has already replaced all major retrievers in Douyin and Douyin Lite, gaining remarkable user engagement. The main advantages of the proposed model are summarized as follows:

- (1) Items are assigned to indexes in real time and indexes can update and repair themselves.
- (2) Streaming VQ provides well-balanced indexes. With a merge-sort modification, all clusters have probability to participate in the recommendation process.
- (3) Streaming VQ exhibits excellent compatibility with multi-task learning.
- (4) Last but not least, compared to recent works, streaming VQ stands out for its implementation-friendly nature. It features a simple and clear framework that primarily comes from off-the-shelf implementation of VQ-VAE [17], which allows it to be easily deployed in large-scale systems.

2 Related Work

As mentioned above, since scanning the entire corpus is unaffordable, various index structures are proposed to approximately select candidates under acceptable errors. Product Quantization [8] is such an example, which gathers items into clusters. When some clusters are selected, all items belonging to them get retrieved. Navigable Small World (NSW [10]) constructs a graph incrementally by inserting nodes, which forms shortcuts among nodes to accelerate search progress. Hierarchical Navigable Small World (HNSW [11]) provides hierarchical structures and rapidly shrinks candidate scale,

¹This operation alone is not hard. It can be achieved by HNSW but only performs a inferior baseline because indexes cannot fit new items. In this paper “attaching items to indexes in real time” potentially demands that indexes also fit items in real time.

so it is widely adopted, especially for large-scale scenarios. There are also several tree-based methods [6, 14] and Locality Sensitive Hashing (LSH) methods [15, 16] aiming at approximately selecting candidates.

As for modeling, the most popular and fundamental architecture, up to now, is so-called “two-tower model”, which mainly comes from DSSM [7]. The two-tower model feeds user-side and item-side raw features into two separate MLPs and obtains the corresponding representations (embeddings). User’s interest on a certain item is given as product of these two embeddings. Because it decouples item and user information, in the serving stage we can store item embeddings in advance, and search results by Approximate Nearest Neighbor (ANN) method.

However, decoupling user and item information discards their interactions, which can only be achieved by complicated models such as MLPs. To address this issue, Tree-based Deep Models (TDM [25], JTM [24], BSAT [26]) propose tree-like structures to hierarchically search candidates from coarse to fine. In TDM, items are collected on leaf nodes while there are some virtual non-leaf nodes to represent the overall property of its child nodes. TDM employs a complicated ranking model and crosses user and item information by an attention module. Considering that HNSW itself has provided a hierarchical structure, NANN [1] directly searches for candidates on HNSW, also with complicated models.

Another method attempts to avoid the Euclidean space assumption required by ANN algorithms. Deep Retrieval (DR [4]), which is mainly composed of isometrical layers, defines items into “paths”, and uses beam search to shrink candidates layer by layer. Compared with TDM and NANN, it focuses more on indexing rather than ranking model complication. There are also some methods [9, 12] using multi-index hash functions to encode items.

Despite that the above methods concentrate on model complication, BLISS [5] argues the importance of index balancing. It iteratively forces a model to map items into buckets and even manually assigns some items to the tail buckets to guarantee balancing.

Attaching items to indexes essentially “quantizes” them into enumerable clusters. So Vector Quantization (VQ) methods can be considered. From the vanilla version of VQ-VAE [17] which introduces learnable clusters, many methods [22, 23] have considered employing it or its variants in retrieval tasks. In this paper, we develop the VQ model into an index method that is updated in a streaming manner, maintains balancing, offers flexibility and is lightweight. We name it as “streaming VQ”.

3 The Streaming Vector Quantization

The proposed streaming VQ model consists of two chronological steps: an indexing step and a ranking step. The retrieval indexing step uses approximate search to shrink candidates from the initial corpus step by step, while the retrieval ranking step provides ordered results and a much smaller set for the subsequent stages. Most existing approaches also follow this two-step paradigm. For example, the most popular two-tower architecture essentially utilizes HNSW to efficiently search candidates. During a specific round of operation, it firstly ranks neighbor nodes by a ranking model (ranking step), and then selects items while discarding others (indexing step). Likewise, TDM and NANN model also rely on their

own index structures (tree-based/HNSW). In DR we also need to train a ranking model to order results and provide user-side input embedding to its indexing step. The difference between streaming VQ/DR and the others is that in streaming VQ/DR indexing step and ranking step are chronologically executed once, whereas for others these two steps are executed alternately.

In Fig 1 we show the entire training framework (for simplicity, we temporarily only consider the single-task scenario). In the indexing step we adopt a two-tower architecture (the reason will be discussed in Sec 5.6) and produce item-side and user-side intermediate embeddings \mathbf{v} and \mathbf{u} (deep blue and yellow blocks in Fig 1) by individual towers. First, these two intermediate embeddings are optimized by an auxiliary task, which employs an in-batch Softmax loss function

$$L_{aux} = \sum_o -\log \frac{\exp(\mathbf{u}_o^T \mathbf{v}_o)}{\sum_r \exp(\mathbf{u}_o^T \mathbf{v}_r)}, \quad (1)$$

where o, r denote sample numbers.

The quantization appears in item side: we keep a set of learnable clusters (16K for single-task version and 32K for multi-task version), and allocate K embeddings. When \mathbf{v} is produced, it searches the top-1 nearest neighbor in cluster set:

$$k_o^* = \arg \min_k \|\mathbf{e}^k - \mathbf{v}_o\|^2, \quad (2)$$

$$\mathbf{e}_o = \mathbf{e}^{k_o^*} = Q(\mathbf{v}_o), \quad (3)$$

where $Q(\cdot)$ means quantization. The embedding \mathbf{e} of selected cluster is also optimized with user-side embedding \mathbf{u} :

$$L_{ind} = \sum_o -\log \frac{\exp(\mathbf{u}_o^T \mathbf{e}_o)}{\sum_r \exp(\mathbf{u}_o^T \mathbf{e}_r)}. \quad (4)$$

This searched cluster serves as the “index” for the input item. Such item-index assignment is written back into Parameter Server (PS). We follow the standard Exponential Moving Average (EMA [17]) update: cluster embeddings are updated by moving average of their belonging items, and items rather than clusters receive gradients of clusters. EMA progress is indicated by red arrows in Fig 1.

Retrieval ranking step shares the same feature embeddings with the retrieval indexing step, and produces another set of compact user-side and item-side intermediate embeddings. We predict each task by an individual tower (head) based on concatenated embeddings, which are supervised by the corresponding labels. The detailed model architecture can be found in Sec 3.5.

In the serving stage, we first rank clusters by

$$\mathbf{u}^T \cdot Q(\mathbf{v}). \quad (5)$$

Then items of the selected clusters are fed into the next ranking step and generate ultimate results.

3.1 Index Immediacy

The overall updating period of existing retrieval models is composed of candidate scanning (check which of them can be recommended), index construction and model dump. Among them, the main cost comes from index construction.

For all existing retrieval models, index construction is interrupted, which misses momentary revision on index semantics. In Douyin, because we have a billion-size corpus, it costs about 1.5-2

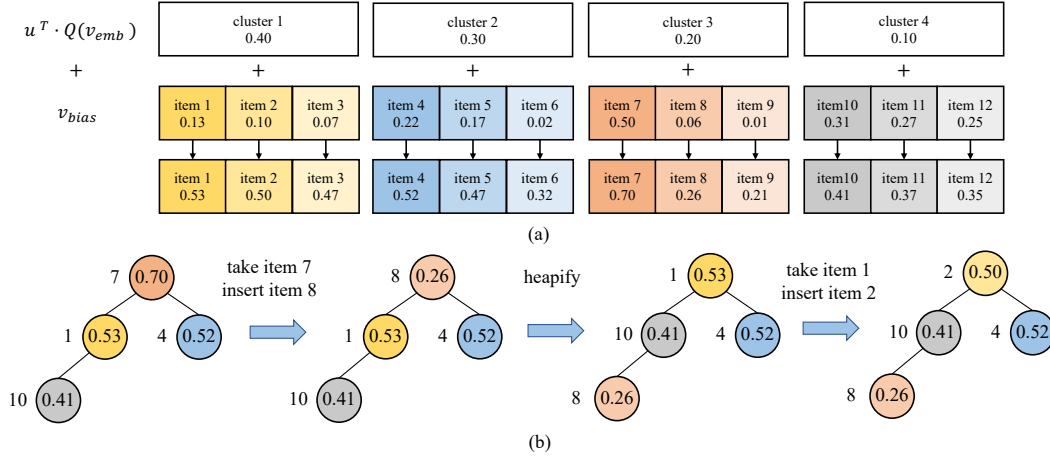


Figure 2: The merge sort solution to finely rank items. The clusters are selected by personality, while popularity can be used to rank items within clusters. Here we visualize the situation where chunk size is 1.

hours to construct HNSW and 1 hour to execute M-step in DR. During this period, indexes remain fixed. However, in a fast-growing platform, emerging trends appear everyday. The situation demands not only real-time assignment on newly submitted videos to the appropriate indexes, but also simultaneous updating of indexes themselves. Otherwise, they cannot fit each other. On the contrary, our model is trained by streaming samples, item-index assignment is immediately decided and stored in PS (key = Item ID, value = Cluster ID) in real time, no interrupted stages are needed, and cluster embeddings are forced to fit items by optimization targets. This endows it with the most important advantage: index immediacy.

Now, in streaming VQ, index construction turns to a real-time step so we have overcome the major obstacle. Moreover, we make candidate scanning asynchronous, so the overall model updating period equals to model dump, which only needs 5-10 minutes.

Even so, there exists a potential problem: item-index assignment is entirely decided by training samples. Popular items are frequently impressed, their assignment is being sufficiently updated. However, new submitted and unpopular ones get fewer opportunities to be impressed or updated, which further deteriorates their performance.

To tackle this problem, we add an additional data stream, candidate stream, to update them. Unlike the training stream which we refer to as the “impression stream”, candidate stream just inputs all candidates one by one with equal probability. As shown in Fig 1 (dotted black arrows), for these samples we just forward them to obtain and store item-index assignment to make sure it matches current semantics of the cluster set. Since for these samples we do not have real labels, no loss functions or gradients are calculated.

3.2 Index Reparability

The streaming update paradigm is a double-edged sword: since we discard index re-construction, the whole model faces risk of degradation. Such phenomenon widely exists in all retrieval models, but is exactly solved by their re-construction operations.

The vanilla VQ-VAE introduces two loss functions: one is the same with L_{ind} , the other emphasizes item-cluster similarity:

$$L_{sim} = \sum_o ||v_o - e_o||^2. \quad (6)$$

In the computer vision area [3, 13], patterns rarely change, so VQ-class methods perform well. However, in large-scale industrial recommendation scenarios, items naturally change their belonging, but L_{sim} counterproductively locks them.

In our early implementation, we followed the same configuration with vanilla VQ-VAE, at the beginning the online metrics were indeed improved. However, we observed model degeneration: the performance gradually worsened over time. Then we realize that in our platform, since global distribution drifts, as recapitulative representations for items, semantics of clusters are, and need to be changed everyday. The item-index relationship is not static, on the contrary, items may belong to various clusters across multiple days. Unfortunately, both L_{ind} and L_{sim} only describe the situation that the item belongs to a certain cluster. If it is no more appropriate, we do not know which cluster it should belong to. That is why the performance degrades.

This problem is solved by replacing L_{sim} with L_{aux} . Because of L_{aux} , item embeddings can be updated (assigned to new belongings) timely and independently, then L_{ind} adjusts clusters based on item representations. After this modification, we successfully observe consistent improvement. We summarize this as the principle of designing a retrieval model: item first. Items decide indexes, not vice versa.

3.3 Index Balancing

For retrieval models, we hope that they uniformly distribute items among indexes so we can select only a few of them to rapidly shrink candidate set. This property is called “index balancing”. Unfortunately, many existing methods suffer from popularity bias and they fail to propose effective techniques to prevent popular items from gathering in several top indexes.

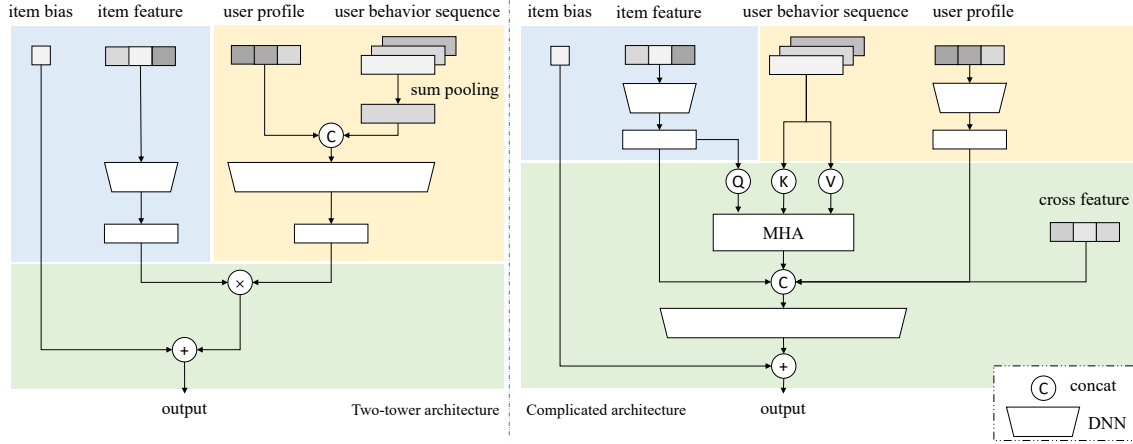


Figure 3: The two architectures of ranking step model. Here blue/yellow/green blocks denote item-side/user-side/cross features, respectively.

Note L_{ind} acquires minimum quantization errors on average. Popular items occupy far more impressions than others, so the most straightforward way to minimize L_{ind} is to break them up and attaching into as many clusters as possible, which naturally produces well balancing. In our implementation, streaming VQ indeed acts this strategy and results in a surprisingly balanced index distribution (see Sec 5.1).

To further improve index balancing, we modify the primary regularization technique. Let \mathbf{w} be preliminary cluster embedding, we insert a popularity term into EMA

$$\mathbf{w}_k^{t+1} = \alpha \cdot \mathbf{w}_k^t + (1 - \alpha) \cdot (\delta^t)^\beta \cdot \mathbf{v}_j^t, \quad (7)$$

where item j belongs to cluster k , t denotes timestamp and δ denotes item occurrence interval as proposed in [21]. Here we add a hyper-parameter β to adjust clustering behavior, and a greater β will impel clusters to focus more on unpopular items (in general, adjusting them brings no effectiveness improvement but only ecosystem change. For example, if we impress more popular items, live or e-com may be hurt. The main goal of introducing hyper parameters is to balance various businesses and avoid being blocked). Then, we also update the counter c that records cluster appearance

$$c_k^{t+1} = \alpha \cdot c_k^t + (1 - \alpha) \cdot (\delta^t)^\beta, \quad (8)$$

and the ultimate representation is calculated as

$$\mathbf{e}_k^{t+1} = \frac{\mathbf{w}_k^{t+1}}{c_k^{t+1}}. \quad (9)$$

We also propose a “disturbance” in vector quantization step, which is to say, modify Eq 2 into:

$$k_o^* = \arg \min_k \|\mathbf{e}_k - \mathbf{v}_o\|^2 \cdot r, \quad (10)$$

$$r = \min\left(\frac{c_k}{\sum_{k'} c_{k'}/K} \cdot s, 1\right),$$

where r denotes the discount coefficient and $s = 5$ is a threshold. It means that if the entire cluster’s impressions are less than $1/s$ times of average, it will be boosted when item searches its top-1 nearest cluster. It also helps to construct a well-balanced index structure.

3.4 Merge Sort for Serving

Representations of items may possess two kinds of intrinsic semantics: personality and popularity. We hope to cluster items according to their personality rather than popularity. To this end, we explicitly decouple item representation into personality part (embedding) and popularity part (bias). Mathematically, modify Eq 5 to

$$\mathbf{u}^T \cdot Q(\mathbf{v}_{emb}) + v_{bias}. \quad (11)$$

By this mean, we observe that items within the same clusters become more semantically consistent. All training loss functions follow the same modification.

Note in Eq 11, even items within the same clusters have a common $Q(\mathbf{v}_{emb})$, v_{bias} can be used to roughly rank them. So we propose a merge sort solution to effectively select candidates for the retrieval ranking step.

As shown in Fig 2(a), $\mathbf{u}^T \cdot Q(\mathbf{v}_{emb})$ provides cluster ranks while v_{bias} gives item ranks within clusters. Then merge sort is performed based on the sum of these two parts. It makes sure that all clusters (even those who have larger size than ranking step input) have probability to give candidates to the final results, thus we can collect a very compact set (50K, only 10% of input size compared with DR ranking step) to outperform other retrievers.

Specifically, we use a maximum heap here to implement k-way merge sort (Fig 2(b)). Items of clusters are first sorted independently and formed as lists, which are divided into chunks (size=8). These lists are then structured into a heap, initialized by their head elements. In each iteration we pop the top element from heap, but take away all elements in its chunk. Then, another chunk from the same list as well as its head element is added to the heap. This strategy effectively reduces computational overhead while maintaining performance quality. Please refer to Appendix.A for more details.

3.5 Model Complication

As mentioned above, in the retrieval indexing step and retrieval ranking step we evaluate 16K and 50K clusters/items, respectively. The scale is no longer prohibitive thus we can afford complicated

Table 1: Detailed comparison between streaming VQ and existing retrieval models. TDM’s data comes from Taobao Ads, others come from internal implementaton.

	HNSW	TDM	NANN	DR	streaming VQ
Are indexes recommendation-oriented?	No	Yes	No	No	Yes
Negative sampling (indexing step)	No	random	No	implicit	in-batch
Popularity debias (index balancing)	No	not mentioned	No	No	Yes
Time cost to construct indexes	1.5-2 hours	0.5 day	1.5-2 hours	1 hour	real-time
Candidate limitation (indexing step)	170M	$\geq 10M$	170M	250M	$\geq 350M$
Touch node (ranking step)	80K	$\geq 10K$	50K	500K	50K
Applicable ranking model	only two-tower	complicated	complicated	only two-tower	complicated

models. In Fig 3 we demonstrate two architectures for both indexing and ranking models: the two-tower architecture and the complicated architecture.

The two-tower (left side in Fig 3) model follows the typical DSSM [7] architecture. Item-side features and user-side features are fed into two individual towers (i.e., MLP) and obtain compact embeddings. User’s interest on this item is calculated as product of these two embeddings. The version using two-tower model in the ranking step is referred to as “VQ Two tower”.

The complicated version (right-side in Fig 3) also feeds item-side and user-side features to produce two intermediate embeddings. However, item-side embedding is fed into a Multi-head Attention module [19] as query to extract nonlinear user-item interaction cues, where user behavior sequence is treated as key and value. Then the transformed feature as well as all others (include cross feature) are fed into a deep MLP model to output the results. This version using complicated ranking model is denoted as “VQ Complicated”.

To reach positive Return of Investment (ROI) for VQ Complicated (see Sec 5.2), we spent nearly half a year improving GPU efficiency (such as assigning operations on specific devices, using fp16 techniques and so on). Then we can finally launch it. Specifically, HNSW/DR/VQ Two-tower/VQ Complicated cost 22K/20K/17K/30K cores (some use GPU, we exchange to CPU for comparison), respectively.

Theoretically, these two architectures can both be deployed in the indexing and ranking steps. However, in our experiments complicated indexing model provides no improvement. As discussed in Sec 5.6), we keep indexing model as the two-tower architecture. On the contrary, for ranking step the complicated version outperforms two-tower ones. However, it also costs more. Considering return on investment (ROI), not all targets are deployed as the complicated version. The details are shown in Sec 5.4.

As an entertainment platform, Douyin has many hot topics and emerging trends, which crowd in users’ recent behavior sequences. However, top topics have already been well estimated and delivered, so a sequential feature dominated by hot topics hardly benefits interest modeling. To solve this problem, we utilize statistical histograms provided by Trinity [20], and filter out items that fall into user’s top 5 secondary clusters. The resulting sequence tends to long-tail interest and gives more semantic cues. Some targets are significantly improved by modified sequence features (see Sec 5.4).

We also add dozens of features in VQ Complicated to reach its best performance.

3.6 Multi-task Streaming VQ

Even the former discussion is demonstrated on single-task framework, streaming VQ can be extended to multiple tasks. As shown in Fig 1, in the indexing step user has individual representations for each task, and they share the same cluster set. For each task we calculate L_{aux} and L_{ind} and propagate gradients simultaneously.

For multi-task version, cluster representations need to be specialized to various tasks. Specifically, Eq 7 and Eq 8 are modified as

$$\mathbf{w}_k^{t+1} = \alpha \cdot \mathbf{w}_k^t + (1 - \alpha) \cdot \prod_p (1 + h_{jp})^{\eta_p} \cdot (\delta^t)^\beta \cdot \mathbf{v}_j^t, \quad (12)$$

$$c_k^{t+1} = \alpha \cdot c_k^t + (1 - \alpha) \cdot \prod_p (1 + h_{jp})^{\eta_p} \cdot (\delta^t)^\beta, \quad (13)$$

where η is another hyper-parameter to balance tasks. h_{jp} is the reward for item j in task p . For example, $h_{jp} = 0/1$ if the video is not/is finished. For stay-time target, it is designed as logarithmic play time. Note that the whole reward is always greater than 1, so clusters will tend to items that produce greater reward scores.

Retrieval ranking step shares feature embeddings for all tasks, and trains their own two-tower or complicated models.

4 Detailed Analysis of Retrieval Models

Table 1 lists 7 aspects that we care for retrieval models, and we discuss them one by one.

Are indexes recommendation-oriented? In this paper, we use “recommendation-oriented” to measure whether the index construction procedure is optimized for recommendation target. In HNSW, indexes are constructed without awareness of their assigned task. Similarly, DR is not a recommendation-oriented retriever because of its M-step. Since L_{aux} and L_{ind} are both supervised by recommendation target, streaming VQ is recommendation-oriented.

Negative sampling method in indexing step. HNSW and NANN involve no negative sampling methods in indexing step. Specially, DR has implicit negative sampling: since all nodes are normalized by Softmax, when we maximize one of them, others are equivalently minimized. However, such minimization does not consider sample distribution, thus DR still heavily suffers from popularity bias. In our implementation, Streaming VQ keeps two-tower architecture in its indexing step, so we can just employ the off-the-shelf in-batch debias solution introduced by [21].

Popularity debias. As mentioned above, DR cannot avoid popular items gathering in the same path. In our system, we collect

500K candidates overall after DR’s indexing step, while the top path has provided 100K. On the contrary, because of all techniques proposed in Sec 3.3, popular items widely distribute among indexes in streaming VQ. Even though most existing methods focus on complication, we argue that popularity debias is essentially another neglected and vital problem.

Time cost to construct indexes. In Douyin, we need 1.5-2 hours to setup HNSW and 1 hour to execute DR’s M-step. In streaming VQ, indexes are constructed and updated with training procedure in real time.

Candidate limitation for indexing step. It means how many candidates we can handle as input. Since we need to store some meta information (e.g. edges) and our corpus size exceeds limitations of a single machine memory, some items are randomly dropped at regular intervals in HNSW. DR’s structure, where one item can be retrieved by 3 paths, is much simplified so we can extend the threshold to 250M. Streaming VQ has an exclusive structure so theoretically it can store 3 times of candidates compared with DR (the detailed analysis can be found in Appendix.B). We just extend to 350M since more candidates may bring some outdated messages.

Touch node for ranking step. Here we show the practical settings rather than the upper bound for each method in our system. Since HNSW/TDM/NANN retrieve candidates in hierarchical structures, for them ranking step touch node refers to the count they calculate overall, and for DR/streaming VQ, it denotes ranking list size. We set the same touch node count for NANN and streaming VQ for fair comparison (see Sec 5.5). Note since streaming VQ has well-balanced index structure and can finely select items within clusters, it outperforms DR even by 10% ranking candidate scale.

Applicable ranking model. Using a complicated ranking model always significantly increases computational overheads. As is well-known, HNSW can not afford complicated architecture. In Douyin, DR also uses two-tower model for ranking step because of poor ROI. Other retrieval models use complicated architecture.

5 Experiments

In this section, we dissect the performance of streaming VQ, including clustering visualization and online metrics. Then, we demonstrate why we focus more on index structures rather than developing complicated ranking models.

5.1 Balanced and Popularity-agnostic indexes

In Fig 4 we visualize index distributions by both statistical histogram (upper) and t-SNE [18] (lower). In the histogram, we aggregate clusters by their belonging item counts. From that, a large proportion clusters has $\leq 25K$ items. Considering that we have a billion-size corpus and 16K clusters, in an ideally uniform distribution, each cluster will be assigned tens of thousands of items. The result we obtain is fairly close to such ideal distribution.

The other figure describes their degree of aggregation in a 2-d space where deeper points represent larger clusters. First, all points uniformly cover the whole area, meaning that they are semantically different with others. Then, points of each hierarchy, especially large-cluster ones disperse, not even locally gathering. It implies that the index structure resists popularity bias.

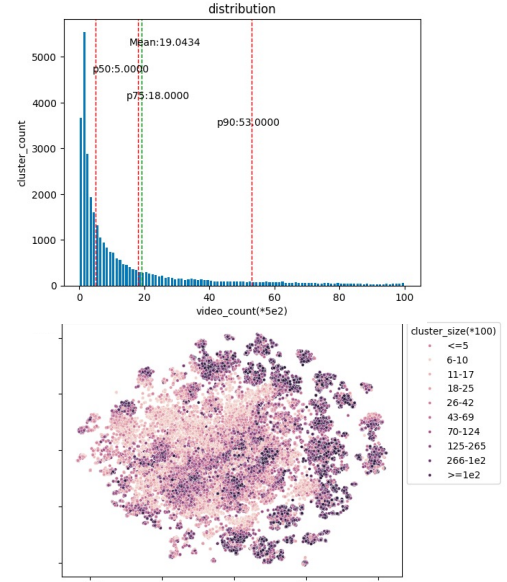


Figure 4: Cluster distributions of streaming VQ.

5.2 Industrial Experiment Environment

In this paper, all experiments are implemented on our large-scale industrial applications: Douyin and Douyin Lite, for video recommendation. As an entertainment platform, we focus on improving user engagement, i.e. Daily Active Users (DAUs). Since users are uniformly grouped into control/experimental groups, DAUs can not be directly measured. We follow the same metrics in Trinity [20]: Average Active Days (AAD), Average Active Hours (AAH), Watch Time. Among these metrics, AAD and AAH are the most important, since they directly refer to DAUs and HAU. Today in Douyin’s ecosystem we have many “everyday” users who cannot contribute to DAUs anymore, so AAH mainly describes high-active users’ engagement and AAD for low-active users. Even Watch Time has strong correlation with DAUs/HAUs, it cannot individually support launch because too long Watch Time may hurt Ads revenue. Since retrievers are trained as single-task models, there are always metric tradeoffs. For example, optimizing finish-target retriever may be hacked by simply enhancing short video delivery, which results in more impressions (VV) but Watch Time degrades². Generally, an acceptable launch gives equal impression change with Watch Time (e.g., increases 0.1% Watch Time but decreases 0.1% impressions). A more effective retriever is expected to improve Watch Time and impressions at the same time.

In the retrieval stage, we already have hundreds of retrievers deployed. So we prefer retrievers that occupy enough impressions, which is measured by Impression Ratio (IR). IR calculates how many impressions this retriever contributes, without deduplication. According to our experience, IR is the most sensitive as well as predictive metric.

²“More impressions degrades Watch Time” just describes our ecosystem, not all platforms follow this rule.

Table 2: Results of online A/B experiments where we only show statistically significant improvement. For each target, we list models for control group (CG) and experimental group (EG). All latest streaming VQ versions have been deployed. Versions that have not been implemented yet are omitted.

Target	CG/EG models	Douyin				Douyin Lite			
		Watch Time	AAD	AAH	IR	Watch Time	AAD	AAH	IR
ST	HNSW/VQ Two-tower	+0.0755%	-	+0.0301%	+25.5%	+0.0612%	+0.0102%	-	+18.23%
FSH	DR/VQ Two-tower	+0.0873%	-	+0.0505%	+43.1%	+0.0766%	+0.0102%	+0.0454%	+31.18%
FSH	VQ Two-tower/VQ Complicated*	+0.0369%	-	+0.0115%	+9.1%	-	-	-	+7.91%
FSH	VQ Two-tower/VQ Complicated	+0.0613%	+0.0099%	+0.0359%	+19.5%	+0.027%	-	-	+18.22%
EVR	HNSW/VQ Two-tower	+0.1270%	+0.0093%	+0.0568%	+59.4%	+0.0931%	+0.0068%	+0.0305%	+26.5%
OST					+17.7%				+66.6%
AST					+151.7%				+125.8%
CST					+96.9%				NA
LST	HNSW/VQ Two-tower	NA	NA	NA	NA	+0.0933%	+0.0112%	+0.0267%	+13.4%
PST	HNSW/VQ Two-tower	+0.0340%	+0.0068%	+0.0196%	+85.4%	+0.0297	+0.0153%	+0.0328%	+85.1%
EVR	VQ Two tower/VQ Complicated	+0.0591%	+0.0075%	+0.0193%	+10.2%	+0.0850%	+0.0152%	+0.0397%	+17.5%

According to our past experience, offline metrics (such as Recall, AUC) of a single head can foretell the corresponding metrics like “finish/play”, but sometimes have a contrary trend with online metrics like Watch Time and DAUs. This is maybe because retrievers cannot ultimately determine all impressed results. We habitually train a model and quickly take online A/B experiments to measure its performance, and only take offline metrics to check whether the model is broken.

Upgrading retrieval models into streaming VQ involves the following target: stay-time (ST), finish (FSH), effective view (EVR), active stay-time (AST), personal page stay-time (PST), old candidate stay-time (OST) comment section stay-time (CST) and lite stay-time (LST). Specifically, stay-time target measures how long the user watches this video, if he/she watches more than 2 seconds, it is recorded as a positive sample. We attach rewards to positive samples according to their actual played time. AST/PST/CST describe the same signal, but under favorite page/personal page/comment section rather than feed tab. OST and LST also model stay-time target, OST just applies ST on candidates published 1-3 months ago, and LST is only for Douyin Lite. Finish directly describes whether the video is finished. Effective view is a comprehensive target: it first predicts the Watch Time’s 60%/70%/80%/90% duration by quantile regression, and then fuses predictions by weighted sum.

Note we have an exchange sheet to define Return of Investment (ROI) between engagement and other metrics. One experiment may have business damage and computational overheads increase, we exchange all metrics into AAD using that relationship (gains as positive, damage as negative), obtaining ultimate results. Only when the overall gain is positive can the change be launched. In this paper, all online results have positive overall gains. Besides, each experimental group should have at least 20% traffic volume and be observed at least 7 days to support its launch. In our platform, retrievers actually need 1-2 months to produce significant results.

5.3 Detailed Baseline Parameters

For HNSW, we keep a queue whose length=5000, each point’s embedding dim is 64, and has edge with 32 other points. In the serving

stage, each point searches 120 candidate points. As for DR, we have 3 levels of paths and each level has 1000 nodes. Each item is assigned to 3 paths.

5.4 Online Experiments

In Table 2 we show the online performance, where only statistically significant metrics are listed. First, for each model, both kinds of change (from HNSW/DR to VQ Two-tower and from VQ Two tower to VQ Complicated) provide large-margin improvement on IR. As demonstrated before, it implies better intrinsic effectiveness, which always refers to index balancing, immediacy and so on.

All experiments produce significant improvement or at least competitive performance on Watch Time, AAD, and AAH. We can conclude that streaming VQ is a better index structure than HNSW and DR (also compared with NANN, see Sec 5.5), and VQ Complicated outperforms VQ Two tower. However, surprisingly, index upgrade alone has produced convinced AAD gain. It suggests that while most existing works focus on complication, index effectiveness is also crucial.

For finish target, “*” means that the sequence features of complicated model are not debiased by Trinity. By comparing the two adjacent rows, the debiased version outperforms the other on all metrics, which suggests that long-tail action provides complementary cues to comprehensively describe user’s interest.

Douyin and Douyin Lite already have a remarkably high baseline in terms of DAUs. Retrieval model change has not provided significant benefit on AAD for several years. However, by streaming VQ instead, we have witnessed impressive improvement in several launches. It verifies the potential of streaming VQ as a novel paradigm of retrieval model.

5.5 Indexing First, or Ranking First?

To better understand the roles that indexing and ranking steps play in large-scale scenarios, we also conduct online experiments to compare NANN [1] (the SOTA retrieval model) with the proposed method based on EVR target. For fair comparison, we ensure to

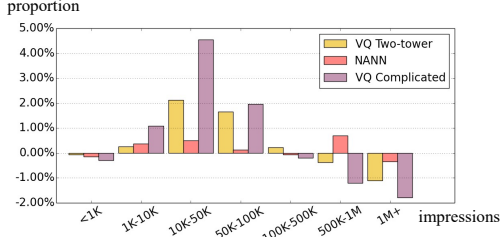


Figure 5: Detailed distribution of impressions. All bars denote relative difference compared with HNSW Two-tower.

match exactly the same features and calculation complexity for NANN and VQ Complicated.

Table 3: A comparison among HNSW, VQ Two-tower, NANN and VQ Complicated.

Method	Watch Time	AAH	VV
VQ Two-tower	+0.0335%	+0.0140%	+0.0604%
NANN	+0.0782%	+0.0232%	-0.1376%
VQ Complicated	+0.0941%	+0.0313%	-0.0748%

In Table 3, we set “HNSW Two-tower” as baseline, and list other models’ performance. VQ Two-tower, NANN, VQ Complicated provide better and better results one by one, measured by Watch Time/AAH. From these results, NANN seems to be competitive with both VQ architectures. However, on the one hand, as we demonstrated in Sec 5.2, NANN loses more VV than its obtained Watch Time, which is not very effective. On the other hand, in Fig 5 we visualize the impression distribution for them (relative difference compared with HNSW Two-tower), which also leads to a different conclusion.

To attract users, delivering more hot items (VV>100M) has been a shortcut, as they easily produce more Watch Time and upvote counts. But a more effective system matches interests precisely so niche topics can seize more impressions. From this aspect, two VQ architectures meet our expectations: VQ Two-tower enhances impressions for “10K-100K” by about 2%, while reducing “100M+” by 1%. In addition, VQ Complicated even improves “10K-50K” impressions by nearly 5% and reduces “100M+” by nearly 2%. However, NANN keeps an unchanged distribution, which suggests that it does not fully leverage features and complication because of indexing bottleneck. In a word, VQ Complicated outperforms NANN on Watch Time and AAH, meanwhile with fewer hot items delivered. Thus it is a better model for our applications.

5.6 Index Complication

As demonstrated in Sec 3, we could also use a complicated model in the indexing step. However, it unexpectedly provides inferior results. To figure out the reason we further implement the following change: (1) Keep a two-tower head, and attach indexes by Eq 10, determining item-index assignment. (2) Feed \mathbf{e} and \mathbf{v} into a complicated model as shown in Fig 3, but receive no gradients from it. (3)

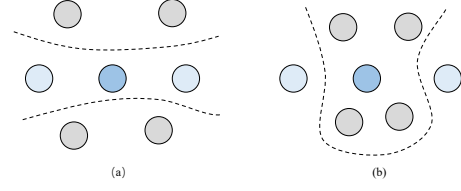


Figure 6: (a) Simplified two-tower model interfaces. (b) Complicated model interfaces.

Except for \mathbf{e} and \mathbf{v} , share all other feature embeddings and DNN parameters of the two heads. By this way we force item intermediate embeddings and cluster embeddings into the same semantic space and as similar as possible. Surprisingly, it still gives inferior results.

To understand this phenomenon, imagine that we have two positive samples (items) and their clusters (blue circles in Fig 6, the deeper one denotes cluster). In the two-tower index version (left), which obeys the Euclidean assumption, the model just produces near-linear interfaces, so the cluster stays in the same subspace with its items. However, in the complicated version it produces nonlinear interfaces that may divide cluster and items into different sub-spaces. The user may miss some interested topics during search. In the future, we may eventually need to construct complicated indexing model, otherwise indexing and ranking steps will gradually drift away, when Large Recommendation Model techniques are applied to ranking models.

5.7 Quantization Error

Since streaming VQ approximately attaches items to clusters, inevitably, quantization error exists. To understand whether we have reached an acceptable situation, we expand cluster count by 10 times, which results in moderate performance. It suggests that the current quantization error has no critical impact, which is maybe because in a streaming clustering paradigm, major quantization error can be gradually corrected in time with the training procedure. Based on this experiment, the single-layer VQ has met our demand.

6 Conclusion

In this paper, we focus on retrieval model of recommender systems. Unlike existing methods, we argue that index performance plays a critical role for the entire model. Aiming at index immediacy, reparability and balancing, we propose a novel retrieval model, streaming VQ, to improve effectiveness. Thanks to its ability to assign items to indexes in real time, it can capture the momentary semantic changes effectively, thereby enabling it to generate candidates in a more precise manner. We also extend its primary version to cooperate with complication and multi-task learning. Streaming VQ has replaced all major retrievers in Douyin and Douyin Lite, leading to remarkable improvement in user engagement.

References

- [1] Rihan Chen, Bin Liu, Han Zhu, Yaoyuan Wang, Qi Li, Buting Ma, Qingbo Hua, Jun Jiang, Yunlong Xu, Hongbo Deng, and Bo Zheng. 2022. Approximate Nearest Neighbor Search under Neural Similarity Metric for Large-Scale Recommendation. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*.

- Knowledge Management* (Atlanta, GA, USA) (CIKM '22). Association for Computing Machinery, New York, NY, USA, 3013–3022. doi:10.1145/3511808.3557098
- [2] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems* (Boston, Massachusetts, USA) (RecSys '16). Association for Computing Machinery, New York, NY, USA, 191–198. doi:10.1145/2959100.2959190
- [3] Patrick Esser, Robin Rombach, and Björn Ommer. 2020. Taming Transformers for High-Resolution Image Synthesis. arXiv:2012.09841 [cs.CV]
- [4] Weihao Gao, Xiangjun Fan, Chong Wang, Jiankai Sun, Kai Jia, Wenzhi Xiao, Ruofan Ding, Xingyan Bin, Hui Yang, and Xiaobing Liu. 2021. Learning An End-to-End Structure for Retrieval in Large-Scale Recommendations. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management* (Virtual Event, Queensland, Australia) (CIKM '21). Association for Computing Machinery, New York, NY, USA, 524–533. doi:10.1145/3459637.3482362
- [5] Gaurav Gupta, Tharun Medini, Anshumali Shrivastava, and Alexander J. Smola. 2022. BLISS: A Billion scale Index using Iterative Re-partitioning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Washington DC, USA) (KDD '22). Association for Computing Machinery, New York, NY, USA, 486–495. doi:10.1145/3534678.3539414
- [6] Michael E. Houle and Michael Nett. 2015. Rank-Based Similarity Search: Reducing the Dimensional Dependence. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37, 1 (2015), 136–150. doi:10.1109/TPAMI.2014.2343223
- [7] 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 *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management* (San Francisco, California, USA) (CIKM '13). Association for Computing Machinery, New York, NY, USA, 2333–2338. doi:10.1145/2505515.2505665
- [8] 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. doi:10.1109/TPAMI.2010.57
- [9] Wang-Cheng Kang and Julian McAuley. 2019. Candidate Generation with Binary Codes for Large-Scale Top-N Recommendation. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (Beijing, China) (CIKM '19). Association for Computing Machinery, New York, NY, USA, 1523–1532. doi:10.1145/3357384.3357930
- [10] Yuri Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems* 45 (2014), 61–68. doi:10.1016/j.is.2013.10.006
- [11] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (April 2020), 824–836. doi:10.1109/TPAMI.2018.2889473
- [12] Tharun Medini, Qixuan Huang, Yiqiu Wang, Vijai Mohan, and Anshumali Shrivastava. 2019. *Extreme classification in log memory using count-min sketch: a case study of amazon search with 50M products*. Curran Associates Inc., Red Hook, NY, USA.
- [13] Fabian Mentzer, David Minnen, Eirikur Agustsson, and Michael Tschannen. 2023. Finite Scalar Quantization: VQ-VAE Made Simple. arXiv:2309.15505 [cs.CV] <https://arxiv.org/abs/2309.15505>
- [14] Marius Muja and David G. Lowe. 2014. Scalable Nearest Neighbor Algorithms for High Dimensional Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 11 (2014), 2227–2240. doi:10.1109/TPAMI.2014.2321376
- [15] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for sublinear time Maximum Inner Product Search (MIPS). In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2* (Montreal, Canada) (NIPS'14). MIT Press, Cambridge, MA, USA, 2321–2329.
- [16] Ryan Spring and Anshumali Shrivastava. 2017. A New Unbiased and Efficient Class of LSH-Based Samplers and Estimators for Partition Function Computation in Log-Linear Models. arXiv:1703.05160 [stat.ML] <https://arxiv.org/abs/1703.05160>
- [17] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. 2017. Neural Discrete Representation Learning. *CoRR* abs/1711.00937 (2017). arXiv:1711.00937 <http://arxiv.org/abs/1711.00937>
- [18] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9, 86 (2008), 2579–2605. <http://jmlr.org/papers/v9/vandermaten08a.html>
- [19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>
- [20] Jing Yan, Liu Jiang, Jianfei Cui, Zhichen Zhao, Xingyan Bin, Feng Zhang, and Zuo Tao Liu. 2024. Trinity: Syncrizing Multi-/Long-Tail/Long-Term Interests All in One. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Barcelona, Spain) (KDD '24). Association for Computing Machinery, New York, NY, USA, 6095–6104. doi:10.1145/3637528.3671651
- [21] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Ajit Kuntekar, Zhe Zhao, Li Wei, and Ed Chi (Eds.). 2019. *Sampling-Bias-Corrected Neural Modeling for Large Corpus Item Recommendations*.
- [22] 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 *Proceedings of the 30th ACM International Conference on Information & Knowledge Management* (Virtual Event, Queensland, Australia) (CIKM '21). Association for Computing Machinery, New York, NY, USA, 2487–2496. doi:10.1145/3459637.3482358
- [23] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. 2022. Learning Discrete Representations via Constrained Clustering for Effective and Efficient Dense Retrieval. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining* (Virtual Event, AZ, USA) (WSDM '22). Association for Computing Machinery, New York, NY, USA, 1328–1336. doi:10.1145/3488560.3498443
- [24] Han Zhu, Daqing Chang, Ziru Xu, Pengye Zhang, Xiang Li, Jie He, Han Li, Jian Xu, and Kun Gai. 2019. *Joint optimization of tree-based index and deep model for recommender systems*. Curran Associates Inc., Red Hook, NY, USA.
- [25] 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 *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (London, United Kingdom) (KDD '18). Association for Computing Machinery, New York, NY, USA, 1079–1088. doi:10.1145/3219819.3219826
- [26] Jingwei Zhuo, Ziru Xu, Wei Dai, Han Zhu, Han Li, Jian Xu, and Kun Gai. 2020. Learning optimal tree models under beam search. In *Proceedings of the 37th International Conference on Machine Learning (ICML '20)*. JMLR.org, Article 1080, 10 pages.

A K-way Merge Sort Implementation

Here we detail the k-way merge sort algorithm used in indexing step serving in Alg.1.

Algorithm 1: K-way Merge Sort Algorithm in Indexing Serving.

Data: sorted intra-cluster lists $\{L_k\}$, indexes $\{i_k\}$, a maximum heap h , heap size H , step size l , target size S .

Result: sorted candidate list R .

$R \leftarrow \emptyset$;

foreach L_k **do**

 insert $L_k[0]$ into h ;

$i_k \leftarrow 0$;

end

heapify h ;

while $|R| < S$ **do**

 find the list $L_{k'}$ that the heap's top element belongs to;

$R \leftarrow R \cup \{L_{k'}[i]\}_{i=i_{k'}+l-1}^{i_{k'}+l-1}$;

 pop top element of h ;

$i_{k'} \leftarrow i_{k'} + l$;

 insert $L_{k'}[i_{k'}]$ into h ;

 heapify h ;

end

return R ;

We set heap size as cluster counts, so in the beginning all top elements of sorted lists are used to establish this heap. Note we take multiple elements in a single operation, which is not equivalent with executing one-element operation repeatedly. However, in practice it significantly speeds up the whole stage and we can stand some mistakes.

B Index Storage Structure of DR and Streaming VQ

For both DR and streaming VQ, item candidates are stored as a compact list, such as $[item_1, item_2, \dots]$, where items of a cluster are segmented by $[seg_1, seg_2, \dots]$. That means from the first item to the $seg_1 - 1$ -th item belong to the first cluster, from the seg_1 -th item to the $seg_2 - 1$ -th item belong to the second cluster, and so on.

In streaming VQ, items are exclusively arranged to clusters. In contrast, DR allows one item to belong to multiple paths ($=3$ in our implementation). So DR needs a three-time long item list compared to streaming VQ. This is why we say that theoretically streaming VQ can afford three times of candidates than DR model. Note this relationship changes with specific settings for each model.