Rethinking ANN-based Retrieval: Multifaceted Learnable Index for Large-scale Recommendation System

Jiang Zhang, Yubo Wang, Wei Chang, Lu Han, Xingying Cheng, Feng Zhang, Min Li, Songhao Jiang, Wei Zheng, Harry Tran, Zhen Wang, Lei Chen, Yueming Wang, Benyu Zhang, Xiangjun Fan, Bi Xue, Qifan Wang
{jiangzhang2024,yubowang,wqfcr}@meta.com
Meta Platforms, Inc.

Abstract

Approximate nearest neighbor (ANN) search is widely used in the retrieval stage of large-scale recommendation systems. In this stage, candidate items are indexed using their learned embedding vectors, and ANN search is executed for each user (or item) query to retrieve a set of relevant items. However, ANN-based retrieval has two key limitations. First, item embeddings and their indices are typically learned in separate stages: indexing is often performed offline after embeddings are trained, which can yield suboptimal retrieval quality—especially for newly created items. Second, although ANN offers sublinear query time, it must still be run for every request, incurring substantial computation cost at industry scale. In this paper, we propose **M**ulti**F**aceted **L**earnable **I**ndex (MFLI), a scalable, real-time retrieval paradigm that learns multifaceted item embeddings and indices within a unified framework and eliminates ANN search at serving time. Specifically, we construct a multifaceted hierarchical codebook via residual quantization of item embeddings and co-train the codebook with the embeddings. We further introduce an efficient multifaceted indexing structure and mechanisms that support real-time updates. At serving time, the learned hierarchical indices are used directly to identify relevant items, avoiding ANN search altogether. Extensive experiments on real-world data with billions of users show that MFLI improves recall on engagement tasks by up to 11.8%, cold-content delivery by up to 57.29%, and semantic relevance by 13.5% compared with prior state-of-the-art methods. We also deploy MFLI in the system and report online experimental results demonstrating improved engagement, less popularity bias, and higher serving efficiency.

Keywords

MultiFaceted learnable index, Large-scale recommendation

ACM Reference Format:
Jiang Zhang, Yubo Wang, Wei Chang, Lu Han, Xingying Cheng, Feng Zhang, Min Li, Songhao Jiang, Wei Zheng, Harry Tran, Zhen Wang, Lei Chen, Yueming Wang, Benyu Zhang, Xiangjun Fan, Bi Xue, Qifan Wang. 2026. Rethinking ANN-based Retrieval: Multifaceted Learnable Index for Large-scale Recommendation System. In *Proceedings of Rethinking ANN-based Retrieval: Multifaceted Learnable Index for Large-scale Recommendation System (Recsys '26).* ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

Recsys '26, Minneapolis, MN

ACM ISBN 978-1-4503-XXXX-X/2026/09
https://doi.org/XXXXXXX.XXXXXXX

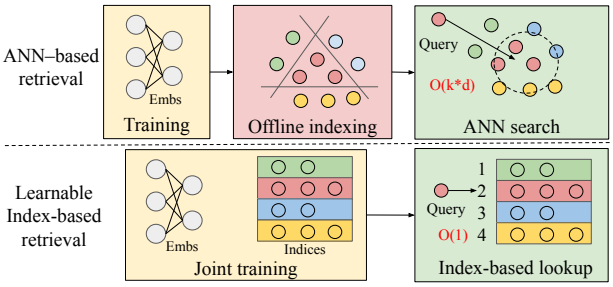


Figure 1: Traditional ANN-based retrieval contains three stages: item embedding learning, offline item indexing, and online ANN search at serving time. In contrast, LI-based retrieval jointly learns item embeddings and indices during training, and performs fast index lookup during serving.

1 Introduction

Approximate nearest neighbor (ANN) search [2, 11] lies at the core of large-scale retrieval in recommendation systems [21, 28, 36], particularly when the candidate set reaches billions of items and exhaustive scanning becomes computationally infeasible [16]. In general, ANN approaches construct an index over the item corpus such that similar items are mapped into the same (or adjacent) regions of the index, enabling efficient candidate item retrieval.

Despite their broad adoption, traditional ANN methods face two major challenges. First, item indexing is typically performed offline and is decoupled from item-embedding learning. Because indexing is refreshed only periodically while the model is updated continuously online, the resulting indices can become stale and inconsistent across refresh cycles, which limits their effectiveness [31, 44]. Moreover, constructing and refreshing indices in real time is expensive due to the heavy offline pipeline. This is particularly problematic for large-scale video recommendation, where millions of new videos may be created daily and freshness is important [13]. Second, the serving cost is high and often increases with candidate set size. For example, the widely used approach, the Inverted File Index (IVF) [19], probes a subset of nearby indices and then computes the k-nearest neighbors within those indices. At billion-scale, however, this still requires substantial online computation. While graph- and tree-based methods [26, 53] can improve search speed, they typically incur significant memory overhead and are difficult to parallelize efficiently on modern GPU accelerators [42], limiting

their serving efficiency (see Figure 1). Therefore, it is an important research problem to address these effectiveness and efficiency limitations in traditional ANN-based retrieval approaches.

Recent work on learning item indexing structures constructs item indices during training using Vector Quantization (VQ) techniques [3, 44], aiming to mitigate inconsistencies between item-embedding learning and indexing. However, these approaches typically rely on a single index structure, which can limit representational capacity and reduce the diversity of retrieved items. Moreover, these works often underemphasize major system-level challenges of real-time index updates, scalability, and flexibility in index design. To address these gaps, we propose **M**ulti-**F**aceted **L**earnable **I**ndex (MFLI), a scalable, real-time retrieval framework that jointly learns multifaceted item indices alongside item embeddings. We further describe MFLI's system design for industrial-scale recommendation systems, covering both modeling and systems considerations needed to enable robust, real-time, multi-way item retrieval. At the high level, MFLI comprises two key components (Figure 1): 1) *Joint item embedding and index training*. We introduce a unified training module that jointly learns item embeddings and a multifaceted, multi-layer codebook structure. Using distinct objective functions, the module produces richer and more diverse item representations, and enables each item to be mapped to multiple sub-codebooks in parallel. 2) *Efficient serving via direct multifaceted lookup*. At serving time, the learned indices are used for retrieval through direct multifaceted item lookup, eliminating the traditional ANN search process and enabling efficient candidate retrieval. The retrieved candidates are then passed to a reranker, which performs per-index reranking to produce the final ranked list.

Building such systems presents several key challenges. First, maintaining balanced index (cluster) sizes during online training to prevent indices from becoming too large or too small. We apply a set of index-balancing strategies during training and introduce a split-and-merge procedure that automatically splits oversized indices and prunes undersized ones at serving time (Section 3.2). Second, designing an indexing scheme that enables efficient storage and retrieval of multifaceted item-to-index and index-to-item mappings. We propose a unified representation for multifaceted indices, together with a compact data structure that supports efficient lookup (Section 3.3). Third, updating the indexing structure in real time, since item embeddings are refreshed asynchronously at different cadences during training and some candidate items may never appear in the training data. We develop an efficient delta-update strategy: a main stream periodically synchronizes indices for all items in the candidate pools, while a fast stream computes indices for fresh items in real time. At serving time, once an index is selected, we prefetch both existing and fresh items associated with that index in parallel and then merge the results, enabling efficient and effective index updates (Section 3.6).

We conduct extensive evaluations of MFLI on real-world data from a commercial platform serving billions of users. MFLI improves recall on engagement tasks by up to 11.8%, recall for cold-content delivery by up to 57.29%, and item semantic relevance by 13.5% over prior SOTAs (Section 4.2). We further deploy MFLI in the system and report online experimental results showing improved engagement, reduced popularity bias, and increased serving efficiency (Section 4.6). Last, we present comprehensive analysis of the learnt index distribution (Section 4.4). Our key contributions are summarized as follows:

- We propose MFLI, a scalable real-time retrieval framework that jointly learns multifaceted item indices alongside item embeddings and eliminates the ANN process during serving.
- We introduce a split-and-merge procedure with guaranteed index balance, a unified and compact multifaceted-index storage for fast lookup, and a real-time delta-update pipeline that keeps the indexing structure fresh, addressing critical system challenges.
- We evaluate MFLI on real-world data and show superior performance over prior SOTAs. The MFLI is deployed in recommendation system, with online experiments validating its effectiveness.

2 Related Work

ANN-based Retrieval. Approximate nearest neighbor (ANN) search has been widely adopted in large-scale retrieval systems [2]. Early work on ANN search proposed a variety of Euclidean-distance-based indexing methods, including k-means-based clustering [8, 19] and product quantization [18]. To improve retrieval accuracy, later approaches explored hashing-based methods [1, 35, 38], graph-based methods [5, 25, 26], as well as tree-based methods [15, 17, 27]. However, these methods typically construct indices offline and are not well suited to GPU parallelization, causing mismatches between the item embeddings and reducing timeliness and scalability of large-scale online recommendation systems.

Index Learning. To overcome the limitations in ANN-based retrieval, recent work has proposed online indexing methods (e.g. Tree-based index models [52, 53], DeepRetrieval [9], Trinity [44], Streaming VQ [3], MERGE [43]) that learn item indices during training, mitigating the inconsistency between item embedding and index, and reducing serving latency and memory footprint. However, these approaches typically depend on a single index structure, which can constrain representational capacity and limit the diversity of retrieved items. Moreover, they often underemphasize important system-level challenges, including real-time index updates, scalability, and flexibility, in index design.

Multifaceted Embedding. Prior works have shown that a single embedding cannot capture different facets of item correlations. Therefore, multifaceted embeddings can help improve recall during retrieval [12, 33, 49, 50]. However, one practical challenge of using multifaceted embeddings for large-scale retrieval is the serving cost, which grows linearly with the number of facets and can become unaffordable in production. This paper addresses this challenge by jointly learning multifaceted item indices and embeddings online, such that the serving overhead is minimal.

Semantic IDs. Recent works on generative retrieval have presented different approaches based on vector quantization [22, 24, 39] for building semantic IDs for items, such that the model can be trained to predict semantic IDs with better generalization capability [7, 20, 31, 51]. There are several key differences between semantic IDs and item indices. First, semantic IDs typically require no hash collisions [37], meaning each item should have a unique semantic ID. By contrast, a learnable index is designed to assign a moderate number of co-engaged items to the same indices to enable effective retrieval. More importantly, the semantic ID of each item is static during model training [41]. However, the indices of items in

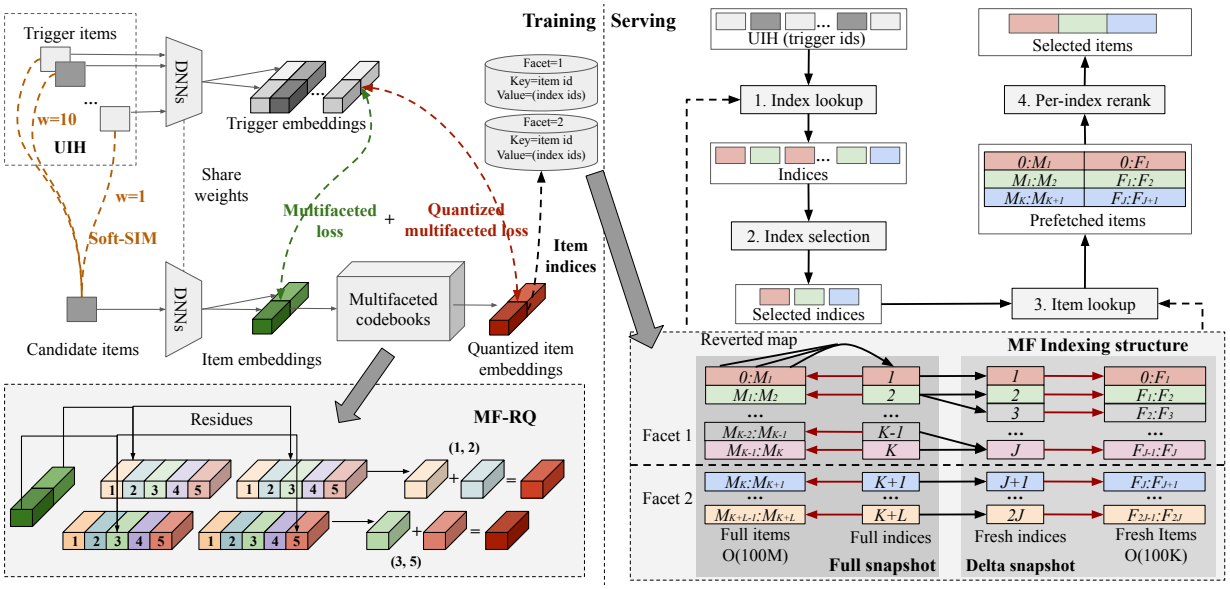


Figure 2: Overview of MFLI framework. During training phase, it learns a multifaceted embedding for each item, and conducts residual quantization on candidate item embeddings by facet in parallel. The codebook and item embeddings are jointly trained via a multifacted loss and a quantized multifaceted loss. The serving part consists of four modules: 1) index lookup, which looks up the multifaceted indices of query item ids; 2) index selection module that selects top K indices; 3) item lookup, which fetches all items in each selected index; 4) per-index rerank, which selects top N items for each selected index.

MFLI will change fast as real-time user engagement behavior shifts, capturing real-time engagement patterns. This poses significantly more challenges for online indexing learning, such as how to balance the index distribution online and how to guarantee cluster sizes for real-time retrieval serving.

3 Methodology

3.1 Overview

The overall training and serving workflow of MFLI are illustrated in Figure 2. Specifically, MFLI designs a multifaceted item representation to capture multiple aspects of the item (e.g., engagement, relevance, freshness), which is trained with a multifaceted loss between trigger-item embeddings and candidate-item embeddings, and a quantized multifaceted loss between trigger-item embeddings and quantized candidate-item embeddings (the codebook). All parameters, including the codebook, are differentiable and optimized via stochastic gradient descent [34]. At serving time, MFLI runs a four-step pipeline: 1) index lookup, which identifies the multifaceted indices for each query/trigger item; 2) index selection, which chooses the indices that best capture user interest; 3) item lookup, which retrieves items associated with the selected indices; and 4) per-index reranking, which selects the top items from each index.

To make MFLI robust and deployable at scale, we address several key challenges: 1) learning a stable multifaceted codebook efficiently under online learning (Section 3.2); 2) bounding the learned cluster sizes to avoid overly large or overly small indices at serving time (Section 3.3); 3) designing an indexing structure that supports efficient and scalable multifaceted index lookup and item retrieval (Section 3.4); and 4) updating the indexing structure in real time with consistency guarantees (Section 3.5).

3.2 Training

This section presents how the multifaceted hierarchical codebook and item embeddings are jointly learned. We first introduce *multifaceted residual quantization* (MF-RQ) to quantize each facet independently with an L-layer hierarchical codebook, and then present the training objective used to optimize both embeddings and codebook parameters end-to-end. Finally, we summarize the techniques we use to improve index balance and stabilize online training.

MF-RQ. Differing from traditional RQ [22] that uses a two dimensional hierarchical codebook to quantize item embeddings, MF-RQ employs a three-dimensional hierarchical codebook. Formally, a F-faceted and L-layer codebook can be denoted by $\{C_1, \ldots, C_L\}$, where $C_l \in \mathbb{R}^{F \times N_l \times d}$ is the lth-layer codebook with size N_l and dimension d. Given an item embedding $v \in \mathbb{R}^{F \times d}$, MF-RQ performs residual quantization on the f-th facet of an item embedding ($v[f]$) using the corresponding facet-specific codebook ($\{C_1[f], \ldots, C_L[f]\}$). In this way, different facets of item embeddings can be quantized independently and in parallel (see bottom left of Figure 2). Specifically, we initialize the residual as $r_0 = v$ and, for each layer $l \in \{1, \ldots, L\}$ and facet $f \in \{1, \ldots, F\}$, we select the nearest codeword from the f-th sub-codebook at layer l as:

$$k_{l,f} = \arg \min_{k \in \{1,\ldots,N_l\}} \|r_{l-1}[f] - C_l[f,k]\|_2^2, \tag{1}$$

where $r_{l-1}[f] \in \mathbb{R}^d$ denotes the f-th facet residual before layer l, and $C_l[f,k] \in \mathbb{R}^d$ is the k-th codeword for facet f at layer l. The layer-l quantized embedding and the updated residual are then given by:

$$q_l[f] = C_l[f, k_{l,f}], \qquad r_l[f] = r_{l-1}[f] - q_l[f]. \tag{2}$$

After l layers, the quantized embedding is obtained by summing the selected codewords across all previous layers:

$$\hat{v}^l[f] = \sum_{k=1}^{l} q_k[f], \qquad \hat{v}^l \in \mathbb{R}^{F\times d}. \tag{3}$$

This computation can be efficiently parallelized on GPU, allowing the indices for all facets to be computed simultaneously.

Training Loss. To jointly learn item embeddings and indices, we minimize a multifaceted loss between trigger item embeddings and candidate item embeddings, and the same multifaceted loss between trigger item embeddings and quantized candidate item embeddings. Specifically, trigger items are previously engaged items selected from the user interaction history (UIH), while candidate items are the most recently engaged items. We use the Search-based Interest Model (SIM [30] to identify semantically relevant items within the user's UIH and assign them higher training weights (see Figure 2). Moreover, we leverage Sampled SoftMax (SSM) loss [40] as the retrieval loss train MFLI. Given a co-engaged item embedding pair (v_i, v_j) and a set of randomly negative item embedding set V_k, the SSM loss for (v_i, v_j) is defined as:

$$L^s(v_i, v_j, V_k) = -\sum_{f=1}^{F} w_{i,j}[f] \log \frac{e^{v_i^T[f]v_j[f]}}{e^{v_i^T[f]v_j[f]} + \sum_{v\in V_k} e^{v_i^T[f]v[f]}}, \tag{4}$$

where $v_i[f] \in \mathrm{R}^d$, $v_j[f] \in \mathrm{R}^d$, $v[f] \in \mathrm{R}^d$ represents the f-th embedding of v_i, v_j, v respectively, d is item embedding dimension, F is the total number of facets, and $w_{i,j}[f]$ is the f-th co-engagement label between item i and j. Note that different facets of item embeddings and codebook are trained on different labels. In addition, we compute the SSM loss between $(v_i, \hat{v}_j^l)$ as $L^{\text{ssm}}(v_i, \hat{v}_j^l, V_k)$, where $\hat{v}_j^l$ denotes the quantized embedding of item j after the l-th quantization layer. The overall training objective for the pair (v_i, v_j) is defined as:

$$L^e(v_i, v_j, V_k) = w_0 L^s(v_i, v_j, V_k) + \sum_{l=1}^{L} w_l L^s(v_i, \hat{v}_j^l, V_k) + L^{\text{a}}(v_i, v_j, V_k), \tag{5}$$

where $w_0, w_1, \ldots, w_L$ are tunable weights and L^{a} denotes auxiliary losses from other non-engagement tasks. In this paper, we incorporate a semantic relevance loss into L^{aux} and apply it to the second facet to improve semantic relevance, referring to the design in [49].

Index Balance Control. We employ two main index-balancing strategies. First, we adopt a *delayed start* scheme by warming up item embeddings before activating the codebook and commencing joint training. In addition, we progressively activate the codebook in a layer-wise manner to improve the stability and robustness of multi-layer codebook training. Second, we introduce a *regularization term* that penalizes over-utilized (i.e., overly popular) codewords to encourage balanced codeword usage. Note that for codebook initialization, we randomly sample a set of item embeddings and use them to initialize the codebook layer by layer.

3.3 Index Rebalance

While we develop several techniques to improve codebook balance during training, these methods do not directly guarantee bounded index sizes at serving time. As a result, some indices can still become overly large or overly small in production, which degrades retrieval quality and efficiency. This issue is further exacerbated in online recommendation systems, where distribution shift between the training item set (primarily exposed items) and the serving-time candidate pool (exposed + unexposed items) makes it difficult to control the serving-time index distribution [45, 48]. To address this, we introduce a index-rebalance step, *split-and-merge*, which splits oversized indices and merges undersized ones as shown in Figure 3. This procedure guarantees that the number of items assigned to each index remains bounded in production. In addition, we design a flexible per-facet index-pruning mechanism for MFLI, enabling us to easily customize the candidate pool for different facets.

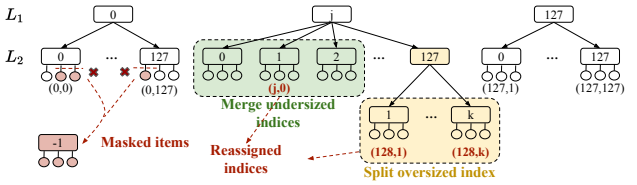


Figure 3: Illustration of index rebalancing, where undersized indices are merged, oversized indices are split, and mask items are assigned with an invalid index.

Split-and-Merge with Index-Size Guarantees. For each oversized index (size > B_{upp}), we independently run k-means clustering on the last-layer residual item embeddings associated with that index, so the splitting step can be fully parallelized. Suppose splitting produces N_s new centroids. We then increase the total number of original indices from N to $N + N_s$. If an item is assigned to new centroid i, we reset its index to $N + i$. Moreover, to prune undersized indices, we scan all L-th-layer indices under each $(L-1)$-th-layer index and identify those with size below B_{low}. We then merge these small indices by reassigning their items to the smallest index among them. In the rare case where the merged index remains smaller than B_{low}, we further merge it with a nearby index that is not oversized.

Item Pruning for Candidate Set Customization. With MFLI, we can flexibly derive a subset of the full candidate corpus for each index facet, enabling facet-specific customization of retrieved items. For example, suppose the second facet is trained with an auxiliary semantic-relevance loss to group semantically related items into the same index, which is beneficial for cold-start items. We can then customize the pool by assigning cold-start items only to this second facet. Specifically, as illustrated in bottom left of Figure 3, we extend the total number of indices by one to represent an *invalid* index, and reassign all masked items to this invalid index, so they are excluded from retrieval at serving time.

3.4 Indexing Structure

The indexing structure in MFLI primarily consists of two mappings: an item-to-index mapping and an index-to-item mapping. These mappings are stored as in-memory tensors to enable efficient lookups. We describe the details below.

Unified Index. In MFLI, an item is assigned with F indices after RQ, one per facet, each of which is originally represented as a tuple $(c_1^f, \ldots, c_L^f)$. Therefore, we first cover them into unified integer indices as:

$$c^f = \sum_{1}^{L-1} c_l^f * (\Pi_{k=1}^{L-l} N_k) + c_L^f, \tag{6}$$

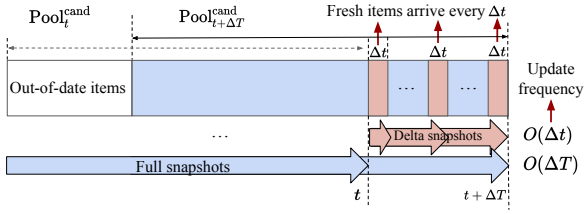


Figure 4: Delta update of MFLI's indexing structure. Every Δt time (e.g., 1 min), a delta snapshot storing the indexing structure of fresh items is produced. Every ΔT time (e.g., 30 minutes), a full snapshot storing the refreshed indexing structure of the full item pool is produced. MFLI fetches items from both the full and delta snapshots during serving.

Suppose facet f has M_f distinct index values in total after index-rebalancing. We then define a *unified index* vector by offsetting the index of each facet into a single contiguous range:

$$\tilde{c}(v) = \left[\, c^1,\ M_1 + c^2,\ \ldots,\ \textstyle\sum_{f=1}^{F-1} M_f + c^F \,\right] \in \mathbb{R}^F, \tag{7}$$

where c^f denotes the (scalar) merged index for facet f, and M_f is the total number of merged indices for facet f.

Item-to-index Mapping. With unified index, we maintain an item-to-index mapping using (i) a dense tensor that stores the unified indices for all items with size (I, F), and (ii) a lightweight *direct lookup* structure that maps each item id to its row offset $i \in [0, I)$, where I is the pool size. Given an item id, we first retrieve its row offset in constant time, and then fetch the corresponding multifaceted indices via a single indexed read from the (I, F) tensor. The time and space complexity are $O(1)$ and $O(I \times F)$ respectively.

Index-to-item Mapping. MFLI stores the index-to-item mapping using two tensors: (i) a count tensor $T^{\text{index}} \in \mathbb{N}_+^{\sum_{f=1}^F M_f}$, where $T^{\text{index}}[m]$ is the number of items assigned to the m-th unified index and $\sum_{f=1}^F M_f$ is the total amount of indices; and (ii) an item-id tensor $T^{\text{item}} \in \mathbb{N}^I$, which stores all item ids sorted by unified indices. Consequently, the items under the m-th unified index correspond to the tensor segment

$$T^{\text{item}}\Big[\textstyle\sum_{k=1}^{m-1} T^{\text{index}}[k] \ :\ \sum_{k=1}^{m} T^{\text{index}}[k]\Big]. \tag{8}$$

The time and space complexity for finding items under an index are $O(1)$ and $O(I + \sum_{f=1}^F M_f)$ respectively. Note that details on how we use these mappings in serving are shown in Section 3.6.

3.5 Real-time Update

In online training, both item embeddings and item inventory evolve continuously, hence it is important to keep MFLI's indexing structure up to date in real time. To this end, we design a *delta-update* strategy: a *delta snapshot* that indexes newly arrived items can be produced within $O(\text{min})$, while a *full snapshot* that indexes the entire item pool is refreshed periodically. At serving time, MFLI fetches items under the selected indices from both the full and delta snapshots and then merges the retrieved candidates (Figure 4).

Motivation. Conceptually, there are two approaches to updating the indexing structure. The first is an *asynchronous in-training update*: we maintain a cached mapping from items to indices and update it whenever an item is re-embedded and re-quantized during training. This approach guarantees the freshness of indexing structure, but it has two major drawbacks. First, because the updates are asynchronous with respect to the checkpoint used for serving, the published indices can become inconsistent across items at serving time. Second, training data typically covers only impression (exposed) items; consequently, unexposed or newly arrived items in the item pool may never be refreshed through training-time updates. Alternatively, the second approach is to periodically recompute indices for the item pool using a fixed model checkpoint. This yields mutually consistent embeddings, codebook, and indices, but incurs higher compute cost and longer end-to-end refresh latency, delaying the delivery of fresh items.

Design Details. In practice, we adopt the second approach and mitigate its freshness limitation with our delta-update design: periodic full-snapshot refreshes ensure global consistency of the indexing structure, while real-time delta snapshots ensure newly arrived items become retrievable within $O(\text{min})$. Since the number of fresh items in the delta snapshot is much smaller than the size of the full candidate set (roughly 100×), MFLI skips index-rebalancing for fresh indices and items, and the computational cost of producing the delta snapshot is minimal. Note that the fresh indices are the original mapped codeword indices of fresh items. However, this can introduce some inconsistencies between the indexing structures of the full and delta snapshots, since the full snapshot may end up with more indices than the delta snapshot due to index-rebalancing. To address this, we maintain a mapping from fine-tuned indices to their original indices, as shown in bottom right of Figure 2. At serving time, once an index from the full snapshot is selected, we directly look up all associated fresh indices and fetch all items contained in those indices (more details are provided in Section 3.6).

3.6 Serving

The overall serving flow of MFLI follows an item-to-item retrieval paradigm, as demonstrated in Figure 2. Upon receiving a user request, we extract a set of T engaged items from the user's interaction history as triggers. Each trigger item is first mapped to a multifaceted index vector $\tilde{c}$ via the *index lookup module*. Next, the *index selection module* selects the indices that best capture the user's current interests. The *item selection module* then retrieves items from the selected indices. Finally, a reranker performs *per-index reranking* to select the top-k candidates for each index and merges the results across indices. We describe each module in detail below.

Index Lookup. For each trigger item, this module retrieves its mapped unified index vector $\tilde{c}(t)$ using the *item-to-index mapping* (see Section 3.4). Concretely, given an item ID, its row offset $i \in [0, I)$ is first obtained via a direct-lookup structure, and then the corresponding $\tilde{c}(t)$ is fetched with a single indexed read from the cached item-to-index mapping tensor. The total time complexity for each request is $O(T)$, and $T * F$ indices will be found.

Index Selection. After receiving $T*F$ indices from the index lookup module, the index selection module selects K unique indices, each of which contains a set of correlated items that a user may be interested in. Because multiple trigger items can map to the same single-facet index, we construct an index-frequency histogram per facet, which can be used for multi-user-interest mining [44]. We describe several strategies used in MFLI:

- *Multi-interest extraction*: MFLI applies multinomial sampling [10] over the index-frequency distribution, so indices mapped from

Table 1: Performance comparison of MFLI and baseline methods. For recall, we report recall@1000 on VVC (Video View Complete), Like, LWT (Long Watch Time), and CCD (Cold Content Delivery) tasks. For semantic relevance, we report the average topic match rate between the trigger and retrieved items at the T_1 (coarse-grained) and T_2 (fine-grained) levels.

Baseline	Recall				Semantic Relevance	
	VVC	Like	LWT	CCD	T1 topic	T2 topic
NeuCF [14]	21.51%	24.44%	25.85%	7.54%	24.80%	17.10%
MoL [46]	22.67% (+5.39%)	25.78% (+5.48%)	26.84% (+3.83%)	7.60% (+0.80%)	39.65% (+59.88%)	26.84% (+56.96%)
HLLM [4]	22.39% (+4.09%)	25.28% (+3.44%)	26.61% (+2.94%)	7.97% (+5.70%)	38.94% (+57.02%)	26.48% (+54.85%)
HSTU [47]	23.64% (+9.90%)	**28.07% (+14.85%)**	26.46% (+2.36%)	**12.08% (+60.21%)**	28.00% (+12.90%)	19.07% (+11.52%)
MTMH [49]	22.85% (+6.23%)	26.11% (+6.83%)	27.23% (+5.34%)	8.18% (+8.49%)	46.10% (+85.89%)	33.55% (+96.20%)
VQIndex [3]	19.41% (-9.76%)	22.77% (-6.83%)	22.98% (-11.10%)	7.60% (+0.80%)	47.47% (+91.41%)	34.56% (+102.11%)
MFLI (Ours)	**24.04% (+11.76%)**	26.72% (+9.33%)	**27.49% (+6.34%)**	11.86% (+57.29%)	**52.31% (+110.93%)**	**38.70% (+126.32%)**

more trigger items are more likely to be selected. Unless otherwise specified, we use this strategy to sample top-k indices in subsequent experiments.

- *Recent-interest boosting*: for indices associated with the most recent triggers, MFLI increases their priority to capture users' latest interests more responsively.
- *Long-tail interest exploration*: for long-tail users with limited interaction history, fewer trigger items are extracted, which yields fewer mapped indices. To mitigate this, for each selected index, we additionally include neighboring indices that share the same upper-level indices, enabling more diversity for exploration.

Item Selection. This module takes as input the K selected unified indices and retrieves all items associated with them from both the *full snapshot* and the *delta snapshot*. For a given index i, the module first fetches the associated items from the latest *full snapshot*, denoted by C_i^{full} (see Eq. (8)). Next, it performs a direct lookup to obtain the *fresh* index (or indices) associated with i. Without index-rebalancing, the associated fresh index is simply i itself. With index-rebalancing, if i is produced by splitting an originally oversized index, then the fresh index corresponds to the original (pre-split) index; if i is produced by merging a set of undersized indices, then the associated fresh indices correspond to those original undersized indices. After extracting the fresh indices, the module fetches the corresponding fresh items from the latest *delta snapshot*, denoted by C_i^{delta}. The final set of items retrieved is $C = \bigcup_{i \in \{1,\dots,K\}} C_i$, where $C_i = C_i^{\text{full}} \cup C_i^{\text{delta}}$ denotes the retrieved items from index i, and $|C| = O(K * B_{upper})$ is bounded due to split-and-merge step.

Per-index Reranking. This module performs *per-index* reranking using a user-to-item reranker model [46]. For each selected index, it ranks the items retrieved in the previous step (in parallel across indices) and retains the top-N items for that index. The resulting per-index shortlists are then merged into a single set, which is passed to subsequent ranking stages. With per-index reranking, MFLI preserves multi-interest coverage across indices while ensuring strong trigger-to-item relevance in retrieval. This design helps prevent any single dominant interest from overwhelming the candidates, improving their quality and diversity[23].

4 Experiments

In this section, we evaluate MFLI on a commercial recommendation system. We first compare MFLI's offline performance with prior state-of-the-art industry retrieval models. Next, we present an ablation study of MFLI and provide a comprehensive analysis of the learned index distribution. We then assess the impact of key hyperparameters on performance. Finally, we deploy MFLI and benchmark it against the baseline retrieval model in online experiments. More training data details are presented in Appendix A.

4.1 Baselines and Evaluation Metrics

For offline evaluation, we compare MFLI against six baseline models in terms of recall and item semantic relevance. Specifically, we report recall@1000 on four tasks: Video View Complete (VVC), Like, Long Watch Time (LWT), and Cold Content Delivery (CCD). We define semantic relevance as the average topic match rate between the input trigger items and the retrieved candidate items. We consider two types of topic IDs, denoted by T_1 and T_2: T_1 captures broader item topics, whereas T_2 captures more fine-grained topics. Unless otherwise specified, MFLI uses two facets. Each facet employs a two-layer codebook with sizes 512 and 128, respectively. The first facet is trained using only a co-engagement loss, while the second facet is trained with both a co-engagement loss and a relevance loss [49]. Moreover, we train MFLI and baselines using real data from Period P_1, and evaluate all models on real data from Period P_2, which follows P_1 (see Appendix A for additional details on model hyperparameters and training setups). We describe the compared baselines in detail below:

- **NeuCF** [14]: it uses only content-agnostic ID features as input and produces a single-facet item embedding via a sparse neural network (NN) followed by a dense NN.
- **MoL** [46]: it extends NeuCF by additionally incorporating item content features as inputs to the DNN.
- **HLLM** [4]: it augments the item tower with content embeddings generated by pretrained LLMs.
- **VQIndex** [3]: it adapts streaming VQ on item retrieval and learns a single-layer codebook of size 10k during training.
- **HSTU** [47]: It is a state-of-the-art sequential recommendation model based on Transformers. For evaluation, we use its learned item embeddings to perform item-to-item retrieval.
- **MTMH** [49]: It is the state-of-the-art retrieval model based on a multi-head architecture with multi-task learning.

Note that VQIndex is one of the SOTA index-based retrieval approaches with single facet, while all other baselines are ANN-based methods (see Appendix D for other index-based retrieval baselines).

Table 2: Ablation study results of MFLI. We independently remove split-and-merge index rebalancing, top-k index selection, and delayed start to quantify the impact of each component on MFLI's recall and semantic relevance. VVC, Like, LWT, and CCD denote the video view complete, like, long watch time, and cold content delivery tasks, respectively.

Baseline	Recall				Semantic Relevance	
	VVC	Like	LWT	CCD	T1 topic	T2 topic
MFLI	24.04%	26.72%	27.49%	11.86%	52.31%	38.70%
- Split-and-merge	19.60% (-18.47%)	23.05% (-13.74%)	22.65% (-17.61%)	11.32% (-4.55%)	49.35% (-5.66%)	34.49% (-10.88%)
- Index selection	21.61% (-10.11%)	25.15% (-5.88%)	25.30% (-7.97%)	12.50% (+5.40%)	53.39% (+2.06%)	40.17% (+3.80%)
- Delayed start	23.60% (-1.83%)	26.38% (-1.27%)	27.64% (+0.55%)	12.24% (+3.20%)	51.70% (-1.17%)	38.28% (-1.09%)

4.2 Main Results

We first compare the offline evaluation results of MFLI against baselines on four engagement tasks: VVC, LWT, and CCD, as well as two item semantic relevance tasks: T_1 and T_2 topic match rates. As demonstrated in Table 1, MFLI achieves both top recalls and relevance across various tasks. Specifically, among the four engagement tasks, MFLI has the highest recall on VVC and LWT, with up to +11.76% and +6.34% improvements respectively. For Like and CCD, MFLI achieves 26.72% and 11.86%, outperforming all other baselines, expect for HSTU. However, HSTU has significantly worse semantic relevance rate. This is expected since the item embeddings of HSTU is trained to maximize the retrieval recall on co-engagement tasks, without explicitly optimizing the item semantic relevance tasks.

Moreover, MFLI outperforms all baselines on semantic relevance, with up to +110.93%/+126.32% gains in I_1/I_2 topic match rate. This shows MFLI retrieves more co-engaged items while preserving intra-index semantic coherence, improving interest matching. Finally, our MFLI delivers higher end-to-end throughput than ANN-based retrieval models. Particularly, compared with MTMH, a multi-ANN retrieval model, MFLI increases throughput by 60%.

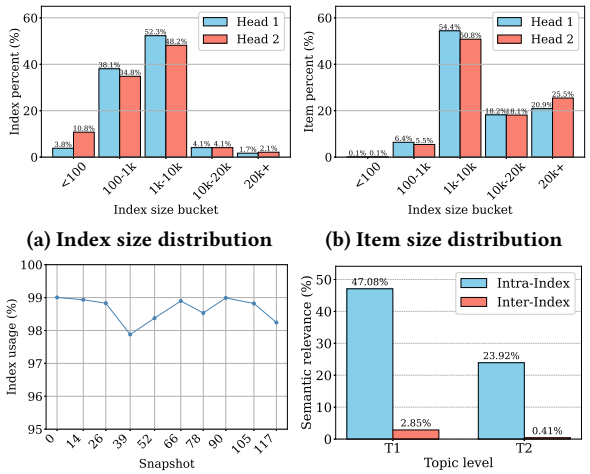
4.3 Ablation Study

Next, we conduct an ablation study on MFLI to quantify the impact of three design choices: split-and-merge index-rebalancing, top-k index selection, and delayed start. We report offline recall on four engagement tasks (VVC, Like, LWT, CCD) and item semantic relevance measured by T_1/T_2 topic match rates.

As shown in Table 2, *split-and-merge* significantly improves MFLI's performance. Removing it leads to consistent regressions in both recall and semantic relevance (e.g., −18.47% Recall on VVC and −10.88% T_1 match rate). This is expected: pruning overly small indices avoids retrieving too few items, while splitting overly large indices improves intra-index homogeneity so that each index corresponds to a tighter cluster of similar items. Moreover, removing top-k index selection improves semantic relevance (T_1/T_2: +2.06%/+3.80%) but reduces recall on the primary engagement tasks (e.g., −10.11% and −5.88% on VVC and Like), indicating a trade-off between engagement-oriented retrieval effectiveness and semantic alignment. Finally, removing delayed start reduces recall on VVC and Like, while yielding only marginal gains in semantic relevance.

4.4 Index Distribution Analysis

In this subsection, we conduct a comprehensive evaluation of the quality of the learned indices in MFLI. We use a two-layer codebook of size 512×128, which maps approximately 200M items into discrete indices. We first examine the distribution of index sizes. As shown in Figure 5a, 98% of indices contain fewer than 20k items, and approximately 90% fall between 100 and 10k items. In addition, we analyze the item-level distribution across indices (Figure 5b).



(a) Index size distribution (b) Item size distribution

(c) Index usage across snapshot (d) Index-level relevance

Figure 5: Analysis of learnt indices in MFLI. (a) Index-size distribution (items per index). (b) Item distribution across index-size buckets (items per index-size buckets). (c) Index usage (percentage of non-empty indices) over training snapshots. (d) Intra-index vs. inter-index relevance.

We observe that around 80% of items are assigned to indices with fewer than 20k items, and more than 60% are assigned to indices smaller than 10k. Note that these statistics are computed *before* split-and-merge; afterward, overly small (size < 100) and overly large indices (size > 20k) are removed. Moreover, we analyze how index usage evolves during online training over a 7-day window. As shown in Figure 5c, more than 98% of indices are actively used, and this ratio remains stable throughout continuous training. This indicates that MFLI maintains broad index coverage over time and can be trained robustly in online recommendation systems.

Last, we compute intra-index and inter-index semantic relevance to assess the semantic purity of the learned indices. Specifically, we randomly sample item pairs from the same index and compute the average topic match rate as the intra-index relevance. We also randomly sample item pairs from different indices and compute the average topic match rate as the inter-index relevance. As demonstrated in Figure 5d, intra-index relevance is substantially higher than inter-index relevance, indicating that items within the same index are indeed semantically related.

Table 3: Effects of number of facets and codebook size.

Num. of facets	Recall of VVC	Recall of LWT	Throughput
One	20.72% (-13.81%)	24.49% (-10.91%)	+0.00%
Two	24.04%	27.49%	-18.8%
Three	24.13% (+0.37%)	28.19% (+2.55%)	-23.9%

(a) Varying number of facets.

Codebook size	Recall of VVC	Recall of LWT	T_1 topic
10k	22.54% (-6.24%)	26.71% (-2.84%)	55.10% (+5.33%)
256×128	24.07% (+0.12%)	27.53% (+0.15%)	52.21% (-0.19%)
512×128	24.04%	27.49%	52.31%
1024×256	23.18% (-3.58%)	27.15% (-1.24%)	52.73% (+0.80%)

(b) Varying codebook size.

4.5 Varying Hyperparameters

In this subsection, we study the sensitivity of MFLI to two key hyperparameters: the *number of facets* and the *codebook size*. We report recall on VVC/CCD and item semantic relevance on T_1 topics under different settings in Tables 3a and 3b, respectively (full results are reported in D).

Number of Facets. We vary the number of facets from 1 to 3. The first facet is trained on an aggregated engagement objective (i.e., the sum of engagement labels), the second emphasizes item semantic relevance via multi-task learning, and the third is specialized for time-spent-related engagement signals. For a fair comparison, we fix the total number of retrieved items by adjusting the number of selected indices per facet; therefore, increasing the number of facets (heads) does *not* increase the downstream ranking overhead, and only incurs small overhead during index lookup and selection.

As shown in Table 3a, overall recall improves as the number of facets increases. Using a single facet results in substantial regressions across all metrics (VVC: −13.81%; LWT: −11.21%; semantic relevance: −12.98%), indicating that a single representation is insufficient to capture diverse engagement patterns. Increasing from two to three facets yields a modest gain in VVC (+0.37%) but a larger improvement in LWT (+3.12%), consistent with the third facet being explicitly optimized for time-spent-related objectives. Importantly, because we keep the total number of retrieved items fixed across settings, increasing the number of facets *does not* linearly decrease end-to-end serving throughput (three facets: -23.9%).

Codebook Size. We consider four variants: a single-layer codebook of size 10k, and three two-layer codebooks with increasing capacity. As shown in Table 3b, the single-layer 10k setting substantially degrades recall (VVC: -6.24%, LWT: -2.84%) despite achieving the highest topic relevance (+5.33%), suggesting that coarse quantization forms semantically coherent but less discriminative clusters. In contrast, among two-layer codebooks, increasing capacity improves semantic alignment with only a slight drop in recall: moving from 512×128 to 1024×256 increases topic relevance (+0.80%) while slightly reducing recall (VVC: -3.58%; LWT: -1.24% → 27.15%). Overall, 512×128 offers a robust balance between recall and relevance.

4.6 Online A/B Testing

We deployed MFLI on a real-world recommendation system and conducted a 7-day A/B experiment. Table 4 reports aggregated impact on (i) *video volume (VV) distribution* to quantify the effectiveness on popularity debias, (ii) *freshness* to measure how quickly fresh items are surfaced, (iii) *engagement efficiency* to capture user-value changes (positive and negative interactions), and (iv) *serving capacity* measured by Queries Per Second (QPS).

Table 4: Aggregated 7-day online A/B testing results. MFLI reduces popularity bias by shifting exposure toward low-VV items and improves freshness, engagement efficiency (more explicit and diverse engagement), and serving throughput.

Category	Metric	Relative change
Video view	[0, 5k)/[5k, 50k)	+279%/+54%
	[50k, 100k)/[100k, ∞)	+11%/-2%
Freshness	[0, 6h)	+221%
	[0, 24h)	+10%
Engagement	Explicit signals	+0.08%
	Diversity	+0.30%
Throughput	QPS	+60%

Video View Distribution Shift. We bucket items by their historical view volumes (VV) and compute the relative changes of total views contributed by each bucket. A shift toward lower-VV buckets indicates that exposure is less dominated by already-popular items (i.e., reduced popularity bias). As shown in Table 4, MFLI substantially increases exposure to low-/mid-popularity content ([0, 5k): +279%; [5k, 50k): +54%), while slightly decreasing exposure to the highest-popularity bucket ([100k, ∞): −2%), indicating that MFLI effectively redistributes traffic from head items toward the long tail.

Freshness. Freshness is measured as the fraction of served items whose age (time since publish) falls within a given time window. We report two windows: *[0, 6h)* capturing "ultra-fresh" content discovery, and *[0, 24h)* capturing same-day freshness. MFLI improves ultra-fresh exposure by +221% and increases same-day exposure by +10%, enabling faster delivery of newly created items.

Engagement Efficiency. We track two engagement signals: *Explicit* engagement (e.g., likes, shares) and *Diversity* (i.e. interest diversity of engagement). An increase in explicit engagement alongside an increased diversity in engaged items indicates improved engagement efficiency (higher user value per impression). MFLI yields a +0.08% lift in explicit engagement and improves its diversity by +0.30%, suggesting better engagement efficiency even as traffic shifts toward fresher content.

Serving Capacity. We evaluate GPU server throughput using QPS (queries per second). MFLI improves serving QPS by +60%, demonstrating substantially higher serving efficiency and increased headroom for reranking model scaling (see Appendix C for more details).

5 Conclusion

We present MFLI, a scalable real-time retrieval framework that jointly optimizes multifaceted item embeddings and indices, eliminating the resource-intensive offline index construction and ANN search required by conventional retrieval systems. We address key production challenges through: (i) index balancing, via training-time strategies and a split-and-merge index rebalancing procedure; (ii) a unified, compact indexing structure for efficient multifaceted index lookup; and (iii) a real-time update pipeline for index refreshing. Extensive offline and online experiments on an industrial recommendation system demonstrate consistent gains in engagement efficiency, user–interest matching, and popularity debiasing, along with improved serving efficiency.

References

[1] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. 2015. Practical and optimal LSH for angular distance. *Advances in neural information processing systems* 28 (2015).

[2] Martin Aumüller and Matteo Ceccarello. 2023. Recent Approaches and Trends in Approximate Nearest Neighbor Search, with Remarks on Benchmarking. *IEEE Data Eng. Bull.* 46, 3 (2023), 89–105.

[3] 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*. 4273–4283.

[4] Junyi Chen, Lu Chi, Bingyue Peng, and Zehuan Yuan. 2024. HLLM: Enhancing sequential recommendations via hierarchical large language models for item and user modeling. *arXiv preprint arXiv:2409.12740* (2024).

[5] Rihan Chen, Bin Liu, Han Zhu, Yaoxuan Wang, Qi Li, Buting Ma, Qingbo Hua, Jun Jiang, Yunlong Xu, Hongbo Deng, et al. 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*. 3013–3022.

[6] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.

[7] Jiaxin Deng, Shiyao Wang, Kuo Cai, Lejian Ren, Qigen Hu, Weifeng Ding, Qiang Luo, and Guorui Zhou. 2025. Onerec: Unifying retrieve and rank with generative recommender and iterative preference alignment. *arXiv preprint arXiv:2502.18965* (2025).

[8] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2025. The faiss library. *IEEE Transactions on Big Data* (2025).

[9] Weihao Gao, Xiangjun Fan, Chong Wang, Jiankai Sun, Kai Jia, Wenzhi Xiao, Ruofan Ding, Xingyan Bin, Hui Yang, and Xiaobing Liu. 2020. Deep retrieval: learning a retrievable structure for large-scale recommendations. *arXiv preprint arXiv:2007.07203* (2020).

[10] Ned Glick. 1973. Sample-based multinomial classification. *Biometrics* (1973), 241–256.

[11] Rajib Kumar Halder, Mohammed Nasir Uddin, Md Ashraf Uddin, Sunil Aryal, and Ansam Khraisat. 2024. Enhancing K-nearest neighbor algorithm: a comprehensive review and performance analysis of modifications. *Journal of Big Data* 11, 1 (2024), 113.

[12] Qilong Han, Chi Zhang, Rui Chen, Riwei Lai, Hongtao Song, and Li Li. 2022. Multi-faceted global item relation learning for session-based recommendation. In *Proceedings of the 45th international ACM SIGIR conference on research and development in information retrieval*. 1705–1715.

[13] Lucky Harichandan, Sasmita Kumari Nayak, and Satyabrata Lenka. 2025. A Comprehensive Review on Video Recommendation System: Models, Challenges, and Applications. *Journal of Engineering Science & Technology Review* 18, 3 (2025).

[14] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[15] Michael E Houle and Michael Nett. 2014. Rank-based similarity search: Reducing the dimensional dependence. *IEEE transactions on pattern analysis and machine intelligence* 37, 1 (2014), 136–150.

[16] Junjie Huang, Jizheng Chen, Jianghao Lin, Jiarui Qin, Ziming Feng, Weinan Zhang, and Yong Yu. 2025. A comprehensive survey on retrieval methods in recommender systems. *ACM Transactions on Information Systems* 44, 1 (2025), 1–43.

[17] Ville Hyvönen, Teemu Pitkänen, Sotiris Tasoulis, Elias Jääsaari, Risto Tuomainen, Liang Wang, Jukka Corander, and Teemu Roos. 2016. Fast nearest neighbor search through sparse random projections and voting. In *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 881–888.

[18] 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.

[19] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

[20] Clark Mingxuan Ju, Liam Collins, Leonardo Neves, Bhuvesh Kumar, Louis Yufeng Wang, Tong Zhao, and Neil Shah. 2025. Generative Recommendation with Semantic IDs: A Practitioner's Handbook. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*. 6420–6425.

[21] Hyeyoung Ko, Suyeon Lee, Yoonseo Park, and Anna Choi. 2022. A survey of recommendation systems: recommendation models, techniques, and application fields. *Electronics* 11, 1 (2022), 141.

[22] Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. 2022. Autoregressive image generation using residual quantization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 11523–11532.

[23] Nian Li, Yunzhu Pan, Chen Gao, Depeng Jin, and Qingmin Liao. 2024. Full-stage Diversified Recommendation: Large-scale Online Experiments in Short-video Platform. In *Proceedings of the ACM Web Conference 2024*. 4565–4574.

[24] Xinchen Luo, Jiangxia Cao, Tianyu Sun, Jinkai Yu, Rui Huang, Wei Yuan, Hezheng Lin, Yichen Zheng, Shiyao Wang, Qigen Hu, et al. 2025. Qarm: Quantitative alignment multi-modal recommendation at kuaishou. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*. 5915–5922.

[25] Yury 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.

[26] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[27] 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.

[28] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).

[29] Biswajit Paria, Chih-Kuan Yeh, Ian EH Yen, Ning Xu, Pradeep Ravikumar, and Barnabás Póczos. 2020. Minimizing flops to learn efficient sparse representations. *arXiv preprint arXiv:2004.05665* (2020).

[30] Qi Pi, Guorui Zhou, Yujing Zhang, Zhe Wang, Lejian Ren, Ying Fan, Xiaoqiang Zhu, and Kun Gai. 2020. Search-based user interest modeling with lifelong sequential behavior data for click-through rate prediction. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2685–2692.

[31] Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan Hulikal Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Tran, Jonah Samost, et al. 2023. Recommender systems with generative retrieval. *Advances in Neural Information Processing Systems* 36 (2023), 10299–10315.

[32] Kaushik Rangadurai, Siyang Yuan, Minhui Huang, Yiqun Liu, Golnaz Ghasemiesfeh, Yunchen Pu, Xinfeng Xie, Xingfeng He, Fangzhou Xu, Andrew Cui, et al. 2024. Hierarchical Structured Neural Network for Retrieval. *arXiv e-prints* (2024), arXiv–2408.

[33] Julian Risch, Philipp Hager, and Ralf Krestel. 2021. Multifaceted domain-specific document embeddings. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Demonstrations*. 78–83.

[34] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

[35] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). *Advances in neural information processing systems* 27 (2014).

[36] Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamny, Gopal Srinivasa, et al. 2022. Results of the NeurIPS'21 challenge on billion-scale approximate nearest neighbor search. In *NeurIPS 2021 Competitions and Demonstrations Track*. PMLR, 177–189.

[37] Anima Singh, Trung Vu, Nikhil Mehta, Raghunandan Keshavan, Maheswaran Sathiamoorthy, Yilin Zheng, Lichan Hong, Lukasz Heldt, Li Wei, Devansh Tandon, et al. 2024. Better generalization with semantic ids: A case study in ranking for recommendations. In *Proceedings of the 18th ACM Conference on Recommender Systems*. 1039–1044.

[38] 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 preprint arXiv:1703.05160* (2017).

[39] Aaron Van Den Oord, Oriol Vinyals, et al. 2017. Neural discrete representation learning. *Advances in neural information processing systems* 30 (2017).

[40] Jiancan Wu, Xiang Wang, Xingyu Gao, Jiawei Chen, Hongcheng Fu, and Tianyu Qiu. 2024. On the effectiveness of sampled softmax loss for item recommendation. *ACM Transactions on Information Systems* 42, 4 (2024), 1–26.

[41] Yi Xu, Moyu Zhang, Chaofan Fan, Jinxin Hu, Xiaochen Li, Yu Zhang, Xiaoyi Zeng, and Jing Zhang. 2025. MMQ-v2: Align, Denoise, and Amplify: Adaptive Behavior Mining for Semantic IDs Learning in Recommendation. *arXiv preprint arXiv:2510.25622* (2025).

[42] Bi Xue, Hong Wu, Lei Chen, Chao Yang, Yiming Ma, Fei Ding, Zhen Wang, Liang Wang, Xiaoheng Mao, Ke Huang, et al. 2025. SilverTorch: A Unified Model-based System to Democratize Large-Scale Recommendation on GPUs. *arXiv preprint arXiv:2511.14881* (2025).

[43] Jing Yan, Yimeng Bai, Zongyu Liu, Yahui Liu, Junwei Wang, Jingze Huang, Haoda Li, Sihao Ding, Shaohui Ruan, and Yang Zhang. 2026. MERGE: Next-Generation Item Indexing Paradigm for Large-Scale Streaming Recommendation. *arXiv preprint arXiv:2601.20199* (2026).

[44] Jing Yan, Liu Jiang, Jianfei Cui, Zhichen Zhao, Xingyan Bin, Feng Zhang, and Zuotao Liu. 2024. Trinity: Syncretizing Multi-/Long-Tail/Long-Term Interests All in One. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 6095–6104.
[45] Zhengyi Yang, Xiangnan He, Jizhi Zhang, Jiancan Wu, Xin Xin, Jiawei Chen, and Xiang Wang. 2023. A generic learning framework for sequential recommendation with distribution shifts. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 331–340.
[46] Jiaqi Zhai, Zhaojie Gong, Yueming Wang, Xiao Sun, Zheng Yan, Fu Li, and Xing Liu. 2023. Revisiting Neural Retrieval on Accelerators. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 5520–5531.
[47] Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Michael He, et al. 2024. Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations. *arXiv preprint arXiv:2402.17152* (2024).
[48] An Zhang, Wenchang Ma, Jingnan Zheng, Xiang Wang, and Tat-Seng Chua. 2024. Robust collaborative filtering to popularity distribution shift. *ACM Transactions on Information Systems* 42, 3 (2024), 1–25.
[49] Jiang Zhang, Sumit Kumar, Wei Chang, Yubo Wang, Feng Zhang, Weize Mao, Hanchao Yu, Aashu Singh, Min Li, and Qifan Wang. 2025. Optimizing Recall or Relevance? A Multi-Task Multi-Head Approach for Item-to-Item Retrieval in Recommendation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*. 5194–5204.
[50] Jihai Zhang, Fangquan Lin, Cheng Yang, and Wei Wang. 2022. Deep multi-representational item network for CTR prediction. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2277–2281.
[51] Guorui Zhou, Hengrui Hu, Hongtao Cheng, Huanjie Wang, Jiaxin Deng, Jinghao Zhang, Kuo Cai, Lejian Ren, Lu Ren, Liao Yu, et al. 2025. Onerec-v2 technical report. *arXiv preprint arXiv:2508.20900* (2025).
[52] 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. *Advances in Neural Information Processing Systems* 32 (2019).
[53] 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*. 1079–1088.

A Training Details

Training setups. We train MFLI alongside the baseline models on real data from Period P_1, and evaluate all models on real data from Period P_2, which follows P_1. For MFLI, we set the total number of selected indices to $K = 200$ and the number of items retained after reranking to $N = 15$. For offline evaluation, top-k index selection is enabled. To compute recall@1000, we sort retrieved items by their reranking scores and keep the top 1000. We use 1,000 because, in industrial systems, the retrieval stage typically passes on the order of 1k candidates to downstream stages [6]. We train the model on 48 A100 GPUs with a batch size of 2048 per GPU. We use the Adagrad optimizer with a learning rate of 0.01, and each training example is used exactly once. Both item embeddings and codewords have dimension 128 (see Table 7 for details).

Codebook regularization loss. As mentioned in Section 3.2, we introduce a *regularization term* that penalizes over-utilized (i.e., overly popular) codewords to encourage balanced codeword usage, following [32]. Specifically, for each codebook layer $C_l \in \mathbb{R}^{F\times N_l\times d}$ and a batch of item (residual) embeddings $r_{l-1} \in \mathbb{R}^{F\times B\times d}$, we define the regularization loss as:

$$Loss(C_l, r_{l-1}) = \frac{1}{FN_l}\sum_{f,j}\Big(\frac{\sum_k ||C_l[f,j] - r_{l-1}[f,k]||_2}{B}\Big)^2. \quad (9)$$

It is used to minimize the sum of squares of the mean distance between each codeword and all item embeddings in a batch. This loss is originally introduced in [29] to ensure that the in-batch cluster distribution is even.

B Index Selection Strategies

After receiving $T \times F$ indices from the index lookup module, the index selection module selects K unique indices. Each selected index represents an *interest bucket* containing a set of correlated items that a user may be interested in. Each dimension $\tilde{c}_f(t)$ corresponds to a single-facet item index for facet $f \in \{1,\dots,F\}$. Because multiple trigger items may map to the same single-facet index, we aggregate these mappings into an index–frequency histogram for each facet, which can be used for multi-interest mining [44]. Specifically, for facet f we compute:

$$h_f(m) \;=\; \sum_{t\in\mathcal{T}} \mathbf{1}\big[\tilde{c}_f(t) = m\big], \qquad m \in \{0,\dots,M_f - 1\}, \quad (10)$$

and its normalized distribution

$$p_f(m) \;=\; \frac{h_f(m)}{\sum_{m'} h_f(m')}. \quad (11)$$

Here, $\mathcal{T}$ denotes the set of trigger items selected from the user's interaction history; $h_f(m)$ counts how many triggers map to index m under facet f; and $p_f(m)$ estimates the selection probability of index m. Intuitively, higher-frequency indices provide stronger evidence of user preference for the corresponding interest bucket. With this, MFLI applies user-interest mining approaches similar to [44] and runs several index selection strategies in parallel. We denote the selected indices for facet f by $\mathcal{S}_f$, and the final merged set by

$$\mathcal{S} \;=\; \bigcup_{f=1}^{F} \mathcal{S}_f,$$

typically with $|\mathcal{S}| = O(10^2)$.Below we summarize several index selection strategies used in MFLI:

Multi-interest extraction. MFLI applies multinomial sampling over the index-frequency distribution to select top-k indices, so that indices supported by more trigger items are more likely to be selected. Specifically, we sample from p_f with a temperature $\tau > 0$:

$$\pi_f(m) \;=\; \frac{p_f(m)^{1/\tau}}{\sum_{m'} p_f(m')^{1/\tau}}, \quad (12)$$

and draw k_1 indices from π_f to form $\mathcal{S}_f$. This encourages the selected indices to match the user's multi-interest profile while allowing controllable exploration via τ. By default, we use $\tau = 1.0$

Recent-interest boosting. Since the trigger set $\mathcal{T}$ is time-ordered, for indices associated with the most recent triggers, MFLI increases their priority to capture users' latest interests more responsively. In practice, this can be implemented by upweighting histogram counts contributed by recent triggers (equivalently, modifying h_f before normalization), or by injecting a small set of indices derived from the most recent triggers directly into $\mathcal{S}_f$.

Long-tail interest exploration. For long-tail users with limited interaction history, fewer trigger items are extracted, which yields fewer mapped indices. To mitigate this, for each selected index, we additionally include neighboring indices that share the same upper-level indices, enabling more diverse retrieval for exploration.

Retrieve more/less for strong/weak interests. Higher/lower index frequency implies stronger/weaker interest signals. To reflect this, MFLI passes $h_f(m)$ to *Per-index rerank* module, in order to control how many items are selected per index. For example, given

Table 5: Effects of number of facets on MFLI's performance. Note that VVC, Like, LWT, CCD represent video view complete, like, long watch time, cold content delivery tasks respectively.

Num. of facets	Recall				Semantic relevance		Throughput
	VVC	Like	LWT	CCD	I_1 topic	I_2 topic	
One	20.72% (-13.81%)	24.19% (-9.47%)	24.49% (-10.91%)	10.53% (-11.21%)	45.52% (-12.98%)	31.51% (-18.58%)	+0.0%
Two (default)	24.04%	26.72%	27.49%	11.86%	52.31%	38.70%	-18.8%
Three	24.13% (+0.37%)	27.53% (+3.03%)	28.19% (+2.55%)	12.23% (+3.12%)	50.60% (-3.27%)	37.11% (-4.11%)	-23.9%

Table 6: Effects of codebook size on MFLI's performance. Note that VVC, Like, LWT, CCD represent video view complete, like, long watch time, cold content delivery tasks respectively.

Codebook size	Recall				Semantic relevance	
	VVC	Like	LWT	CCD	I_1 topic	I_2 topic
10k	22.54% (-6.24%)	25.51% (-4.53%)	26.71% (-2.84%)	10.82% (-8.77%)	55.10% (+5.33%)	43.10% (+11.37%)
256×128	24.07% (+0.12%)	26.82% (+0.37%)	27.53% (+0.15%)	11.43% (-3.63%)	52.21% (-0.19%)	38.76% (+0.16%)
512×128 (default)	24.04%	26.72%	27.49%	11.86%	52.31%	38.70%
1024×256	23.18% (-3.58%)	25.97% (-2.81%)	27.15% (-1.24%)	11.98% (+1.01%)	52.73% (+0.80%)	39.49% (+2.04%)

Table 7: Hyperparameters of MFLI.

Name	Value
Number of GPUs	48 A100s
Batch size	2048
Learning rate	0.01
Optimizer	Adagrad
Training epoch	1
Item embedding dim	128
Codeword embedding dim	128

Table 8: Comparing MFLI with other index-based retrieval baselines (i.e. Trinity and SID).

Baseline	Recall delta	Semantic relevance delta
MFLI	–	–
SIDs [31]	-46.92%	+14.70%
Trinity [44]	-10.32%	+2.34%

Table 9: Ablation study of regularization loss in MFLI for codebook balance.

Baseline	Recall delta	Semantic relevance delta
MFLI	–	–
MFLI w/o reg. loss	-11.28%	-3.11%

a facet-level candidate item quota Q_f^{tot}, we allocate an index-level item quota as:

$$Q_f(m) = \left\lceil Q_f^{\text{tot}} \cdot \frac{\left(h_f(m)\right)^{\alpha}}{\sum_{m' \in \mathcal{S}_f} \left(h_f(m')\right)^{\alpha}} \right\rceil, \tag{13}$$

where $\alpha \geq 0$ controls how aggressively we prioritize strong interests (with $\alpha = 0$ yielding uniform allocation).

These selection paths operate in parallel for each facet, and we merge the indices selected by all paths across all facets to obtain the final set $\mathcal{S}$. Notably, if we fix the total number of selected indices $|\mathcal{S}|$, increasing the number of facets does not increase the reranking cost; therefore, the serving overhead remains bounded.

C Scaling MFLI

There are two ways to scale MFLI. The first is to increase the number of facets, where each facet is trained with a distinct objective so that multi-way retrieval can be performed at serving time. Note that, by controlling the total number of selected indices, this does not increase serving cost linearly. During online A/B testing of MFLI, we did not observe any latency regression. Second, since MFLI does not rely on ANN search, its end-to-end throughput is significantly higher than ANN-based approaches. This leaves headroom to use more complicated reranker models, improving reranking precision.

D Supplementary Evaluation Results

In this section, we study the sensitivity of MFLI to two key hyperparameters: the *number of facets* and the *codebook size*. We report recall on four engagement tasks and item semantic relevance on T_1/T_2 topics under different settings in Tables 5 and 6, respectively.

Number of Facets. We vary the number of facets from 1 to 3. The first facet is trained on an aggregated engagement objective (i.e., the sum of engagement labels). The second facet emphasizes item semantic relevance via multi-task learning, following [49]. The third facet is specialized for time-spent-related engagement signals. For a fair comparison, we fix the total number of retrieved items by adjusting the number of selected indices per facet. Specifically, for the single-facet model we retrieve the top-200 indices. For the two-facet model, we retrieve 100 indices per facet. For the three-facet model, we retrieve 75/50/75 indices from facets 1/2/3, respectively.

As shown in Table 5, increasing the number of facets (from 1 to 3) consistently improves recall because each facet's item embeddings and indices are optimized for a distinct objective—aggregated engagement, semantic relevance, or time spent—so different facets retrieve complementary sets of correlated items. Moreover, because we keep the total number of selected indices fixed across settings,

the downstream reranking cost remains unchanged; as a result, end-to-end serving overhead does not scale linearly with the number of facets.

Codebook Size. We compare four codebook variants: a single-layer codebook of size 10k and three two-layer codebooks with increasing capacity. As shown in Table 6, the single-layer 10k setting consistently regresses recall across all tasks (e.g., VVC: −6.24%, Like: −4.53%, LWT: −2.84%, CCD: −8.77%) despite achieving the strongest interest matching (e.g., I_1 topic: +5.33%, I_2 topic: +11.37%), suggesting that coarse quantization yields semantically coherent but less discriminative clusters. Among the two-layer codebooks, 256×128 and 512×128 deliver the best overall recall, while larger capacity (1024×256) improves interest matching (I_1: +0.80%, I_2: +2.04%) at the cost of lower recall (e.g., VVC: −3.58%, LWT: −1.24%). Overall, 512×128 provides a robust balance between recall and interest matching and is used as the default setting.

Comparison with Other Index-based Retrieval Models. We compare MFLI with two other representative index-based retrieval approaches: Trinity [44] and Semantic IDs (SIDs) [31]. Note that Trinity is not a single retrieval model but rather a system comprising three retrieval generators targeting different types of user interest. For a fair comparison, we compare MFLI against the index learning component of Trinity, which uses a two-layer VQ codebook (size: 128 × 1024). Similar to VQIndex, Trinity achieves slightly better semantic relevance but at the cost of a notable regression in recall (see Table 8). Moreover, Semantic IDs (SIDs) are trained using RQ-VAE with a two-layer RQ codebook (size: 1024 × 1024) to assign semantically similar items to the same indices without leveraging real-time co-engagement data. As shown in Table 8, SID-based retrieval suffers a 46.92% drop in recall despite achieving significantly better semantic relevance. This is expected, as SIDs optimize for semantic coherence within clusters rather than capturing user engagement patterns. These results demonstrate that MFLI consistently outperforms existing index-based retrieval models. Finally, we note that MERGE [43], the most recent baseline in this space, focuses on improving the index balance of VQ-based approaches. Its contributions are orthogonal to MFLI and could be combined with our method to further improve index balance. We do not include a direct comparison due to the lack of publicly available reproducible code.

Ablation Study on Regularization Loss. The regularization loss increases when more items are assigned to the same codebook entry (i.e., their embeddings cluster around a single code vector; see Section 3.2). Minimizing this loss penalizes oversized indices and encourages a more balanced codebook utilization. We conduct an ablation study by removing this loss and observe an 11.94% drop in recall and a 3.11% drop in relevance (see Table 9), indicating that the regularization loss effectively improves codebook balance and, consequently, the overall retrieval performance of MFLI.