CUBER: KEEP DEPENDENCY IN PARALLEL SID FOR GENERATIVE RECOMMENDATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Semantic IDs (SIDs) play a critical role in generative recommendation (GR), with early approaches capturing hierarchical item information through progressively refined code sequences. To improve efficiency, more recent approaches based on product quantization (PQ) decompose item representations into multiple subspaces and quantize them independently, enabling parallel SID construction and single-forward code prediction. However, such methods do not explicitly encode structured dependencies among SID codes. To address this limitation, we present Cuber, which captures structured dependencies among SID codes while retaining parallel code prediction. Specifically, Cuber integrates three key techniques: (1) RPQ-Tree, a unified SID construction framework that recursively composes residual and product quantization, introducing dependencies across quantization components while supporting single-forward prediction; (2) a geometry-aware embedding initialization scheme that transfers the structural relationships learned during quantization into the code-token embeddings; and (3) a progressive training mechanism that exploits partial-tree predictions to mine increasingly challenging hard negatives, strengthening both code-level discrimination and catalog-level ranking. Extensive experiments demonstrate that Cuber consistently outperforms competitive baselines across multiple recommendation metrics, achieving 25.2% improvement in Recall and 23.6% in NDCG.

1 INTRODUCTION

Generative recommendation (GR) (Geng et al., 2022; Zhai et al., 2024) models the joint distribution over user behaviors and target items. By translating items into discrete tokens, these models can capture the semantics of individual items and generate the target items accordingly. Early GR methods typically represent items with predefined textual descriptions or fixed item IDs (Geng et al., 2022; Hua et al., 2023). However, as item catalogs grow and their content evolves, a fixed vocabulary fails to represent all items faithfully. Semantic IDs (SIDs) (Rajput et al., 2023; Zheng et al., 2024) have therefore emerged as an alternative: items that are semantically similar, e.g., those sharing the same category, tend to share a common token prefix. Specifically, successive tokens progressively refine broad categories into fine-grained traits, ultimately pinpointing individual items. Unfortunately, this scheme relies on auto-regressive token generation (Vaswani et al., 2017), whose latency scales with the number of output tokens and is often prohibitively high.

Existing works (Hou et al., 2025; Wei et al., 2026) also explore parallel SID generation, where multiple tokens are produced within a single forward pass, using techniques such as product quantization (PQ) (Jegou et al., 2010). PQ partitions the item representation into multiple subspaces that are quantized independently, with the resulting codewords jointly forming the SID. During inference, separate output heads predict these codewords in parallel, which are then combined to identify the predicted next item. This design substantially improves inference efficiency, but the independent treatment of subspaces overlooks hierarchical dependencies in the item space, potentially leading to suboptimal reconstruction quality (Guo et al., 2016). Since items naturally follow a hierarchical taxonomy, explicitly modeling these dependencies is essential.

In this paper, we propose to incorporate the hierarchical structure of the item space into SID construction, while retaining single-forward decoding for efficient inference. Unfortunately, achieving this goal is non-trivial, as it requires addressing three key challenges: (1) supporting arbitrary quan-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

tization configurations while preserving single-forward decoding; (2) allowing the model to capture the hierarchical structure of the item space encoded in the SIDs; and (3) effectively distinguishing semantically confusable items that share similar coarse-level codes but differ at finer levels of the SID hierarchy.

To this end, we present Cuber, a unified SID framework that supports flexible quantization configurations, helps the model capture the hierarchical structure encoded in SIDs, and enhances its discrimination among semantically confusable items, while retaining single-forward inference. Cuberachieves this through three key techniques:

(1) The **SID construction framework** introduces RPQ-Tree, which supports flexible compositions of product quantization (PQ) and residual quantization (RQ) through three recursively nestable node types. Its residual operations allow later stages to capture information not represented by preceding ones. Recent work has incorporated PQ into the final stage of RQ, but still relies on autoregressive decoding (Chen et al., 2025; Zhao et al., 2026). RPQ-Tree instead supports general PQ–RQ compositions with single-forward parallel decoding.

(2) The **geometry-aware embedding initialization** transfers the structural information encoded in the RPQ-Tree into the generator through its initial code-token embeddings. It recursively aligns codewords from different branches into a shared item-representation space and then projects them into the token embedding space with a shared geometry-preserving transformation, allowing the initialized embeddings to reflect the relationships encoded in the SIDs while remaining fully trainable.

(3) The **progressive training mechanism** leverages the hierarchy of the RPQ-Tree to progressively refine discrimination from coarse to fine levels. Hard negatives are mined using increasingly deeper tree levels and are used both to emphasize confusing non-target codes in the weighted cross-entropy objective and to form challenging candidates for catalog-level ranking, thereby improving code-level discrimination and item-level recommendation.

Extensive experiments demonstrate that Cuber consistently outperforms competitive baselines across multiple key metrics, achieving a 25.2% improvement in Recall and a 23.6% in NDCG.

2 PRELIMINARIES

Formulation of Semantic ID. Let $\mathcal{I}$ denote the item set and $\boldsymbol{h}_u = (i_1, i_2, \ldots, i_t)$ the interaction sequence (t for t-th item related) of user u. Given $\boldsymbol{h}_u$, sequential recommendation aims to identify the items that the user is most likely to interact with next. In the SID-based framework, each item is represented by a sequence of discrete codewords (codes), known as a semantic identifier (SID). Specifically, a pretrained text encoder converts the textual content of item i into an embedding $\boldsymbol{e}_i \in \mathbb{R}^d$, and quantizes it into an M-code representation $\boldsymbol{z}_i = (z_{i,1}, z_{i,2}, \ldots, z_{i,M})$.

Quantization Encoding. Residual quantization (RQ) (Chen et al., 2010) and product quantization (PQ) (Jegou et al., 2010) are two representative vector quantization methods that minimize reconstruction error $\|\mathbf{e}_i - \hat{\mathbf{e}}_i\|_2$. Let $Q_m(\boldsymbol{x})$ return the nearest codeword to $\boldsymbol{x}$ in the m-th codebook. The two methods differ fundamentally in how the quantization process is decomposed:

$$\text{RQ:}\quad \boldsymbol{r}_{i,0} = \boldsymbol{e}_i,\quad \boldsymbol{q}_{i,m} = Q_m(\boldsymbol{r}_{i,m-1}),\quad \boldsymbol{r}_{i,m} = \boldsymbol{r}_{i,m-1} - \boldsymbol{q}_{i,m},\quad \hat{\boldsymbol{e}}_i = \sum\nolimits_{m=1}^{M} \boldsymbol{q}_{i,m};$$
$$\text{PQ:}\quad \boldsymbol{e}_i = [\boldsymbol{e}_{i,1}; \ldots; \boldsymbol{e}_{i,M}],\quad \boldsymbol{q}_{i,m} = Q_m(\boldsymbol{e}_{i,m}),\quad \hat{\boldsymbol{e}}_i = [\boldsymbol{q}_{i,1}; \ldots; \boldsymbol{q}_{i,M}].$$

Building on PQ, OPQ (Ge et al., 2013) first rotates the input using a learned orthogonal matrix $\boldsymbol{R}$ and then performs PQ in the rotated space.

Decoding from Inferred Codeword. Auto-regressive generation produces a valid item SID by iteratively decoding each code, factorizing the SID probability as follows:

$$p(\boldsymbol{z}_i \mid \boldsymbol{h}_u) = \prod_{m=1}^{M} p(z_{i,m} \mid \boldsymbol{h}_u, z_{i,<m}).$$

In contrast, parallel decoding uses M code-specific prediction heads to predict all code distributions in a single pass, and then ranks the items via

$$s(i \mid \boldsymbol{h}_u) = \sum_{m=1}^{M} \log p_m(z_{i,m} \mid \boldsymbol{h}_u), \tag{1}$$

to obtain the Top-K results. To avoid costly full-catalog scoring, graph decoding (Hou et al., 2025) can be employed to narrow the search space and approximately retrieve a set of high-scoring items.

Motivation of Quantization Combination. Residual quantization (RQ) provides progressive refinement through sequential residual encoding, but is typically coupled with autoregressive generation, whereas product quantization (PQ) and its optimized variant OPQ enable parallel decoding at the cost of relatively lower reconstruction fidelity. As shown in Table 1, augmenting OPQ with residual quantization yields increasingly larger reductions in reconstruction error as the SID length grows, with the maximum improvement exceeding 50%. These observations suggest that the two quantization paradigms are complementary, motivating us to develop a unified scheme that combines their strengths while remaining compatible with single-forward inference.

Table 1: Reconstruction error using various quantization methods.

SID length	OPQ	OPQ + Residual	Improved
4	0.500	0.425	14.9%
8	0.426	0.338	20.6%
16	0.296	0.216	27.2%
32	0.126	0.087	31.3%
64	0.030	0.012	58.6%

3 METHODOLOGY

Overview. We present Cuber, a unified SID scheme for generative recommendation (GR). Subsection 3.1 introduces RPQ-Tree, which recursively combines residual and product quantization to construct SIDs with the desired structure. Subsection 3.2 introduces the geometry-aware transformer, which captures the quantization structure of the RPQ-Tree and enables the model to learn this structure. And, Subsection 3.3 introduces the progressive training mechanism, which focuses more on the codewords that are difficult to distinguish.

3.1 RPQ-TREE WITH COMBINATION

Tree Structure. RPQ-Tree is a tree-structured framework, encoding the item as desired. Specifically, an RPQ-Tree $\mathcal{T}$ consists of three types of nodes that can be recursively nested, as in Fig. 1(a):

- ***R***(*esidual*). The R-node consists of L ($L \geq 2$) ordered sub-trees, denoted $R(\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_L)$. Except for $\mathcal{T}_1$, each subsequent sub-tree operates on the quantization residual of its immediate predecessor. The output of this node is the reconstruction of its input, obtained by summing the outputs of all its sub-trees.
- ***P***(*roduct*). The P-node consists of $G \geq 2$ parallel sub-trees, denoted $P_G(\mathcal{T})$. It first applies an orthogonal rotation to the input vector and partitions the rotated vector into G sub-vectors. Each sub-vector is then processed by a sub-tree. Finally, the outputs are concatenated and transformed back to the original space, yielding a reconstruction.
- ***Q***(*uantization*). The quantization node terminates a recursive branch by applying K-Means quantization to its input, producing one SID code and a corresponding reconstruction vector (the assigned cluster centroid). This node marks the end of recursion within its subspace.

Notably, R-nodes increase the tree depth, allowing sub-trees to progressively encode the residual information from their predecessors, whereas P-nodes enable finer-grained encoding within rotated and partitioned subspaces. Since each node in an RPQ-Tree takes a vector as its input and outputs its reconstruction, different types of nodes can be recursively composed, giving rise to a rich space of tree structures. As special cases, with a code budget of M, RQ-KMeans and OPQ can be expressed as $\mathcal{T}_{\mathrm{RQ-KMeans}} = R(Q_1, \ldots, Q_M)$ and $\mathcal{T}_{\mathrm{OPQ}} = P_M(Q)$, respectively. Fig. 1(b) and (c) further show two typical topologies with more than one layer, which we detail in Appendix A.

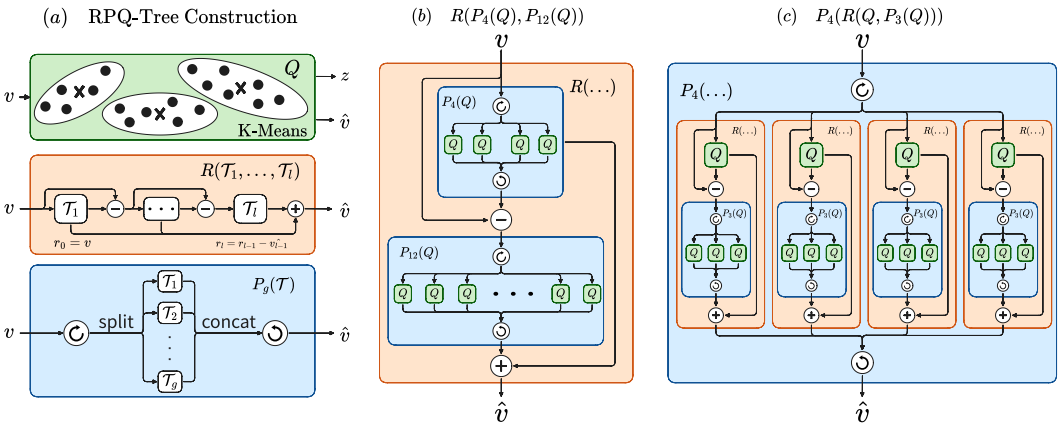

Figure 1: Node types in RPQ-Tree and structured examples.

Compatibility with Training. The training of an RPQ-Tree follows a simple and natural layer-wise strategy. We proceed through the layers sequentially, training codebooks with K-Means and learning rotation matrices via OPQ (while discarding the codebooks learned by OPQ). Note that, for a group of Q-nodes sharing the same nearest preceding P-node, a single OPQ procedure suffices to jointly learn the rotation matrix for the P-node and the codebooks for the Q-nodes. Let D denote the tree depth, C_l the set of code indices at layer k. For item decoding, we adopt a similar score-and-rank approach as in Eq. 1, but via a weighted sum of the layer-wise average codeword scores:

$$s_D(i \mid \boldsymbol{h}_u) = \sum_{l=1}^{D} \alpha_l^{score} \frac{1}{|C_l|} \sum_{m \in C_l} \log p_m(c_{i,m} \mid \boldsymbol{h}_u), \qquad \sum_{l=1}^{D} \alpha_l^{score} = 1. \tag{2}$$

3.2 Geometry-Aware Embedding Initialization

Lightweight Scheme. Beyond the discrete indices, SID codes are associated with continuous codewords, whose geometry captures the structural information learned during quantization. However, most SID-based GR methods separate SID construction from model training, leaving this codeword geometry unexploited by the generator. Although recent studies Liu et al. (2025); Fu et al. (2026) have explored bridging this gap through end-to-end optimization, such coupling introduces additional optimization complexity and may pose challenges to training stability. We thus propose a lightweight code-token initialization scheme, tailored to the recursive structure of the RPQ-Tree, that allows the initialized embeddings to inherit the structural information from the codeword space.

Geometry Behavior. We aim to map the codewords learned during codebook construction into the token embedding space while preserving their relative geometry. For ease of exposition, we consider the case where all codewords reside in the same d-dimensional space as the original item representation, with embedding dimension $D > d$. To align the two spaces, we introduce a semi-orthogonal transformation matrix $\boldsymbol{W} \in \mathbb{R}^{D \times d}$ satisfying $\boldsymbol{W}^\top \boldsymbol{W} = \mathbf{I}_d$. In practice, we construct $\boldsymbol{W}$ by applying thin QR decomposition to a standard Gaussian matrix $\boldsymbol{A} \in \mathbb{R}^{D \times d}$:

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}, \qquad \boldsymbol{W} = \boldsymbol{Q}.$$

The same transformation $\boldsymbol{W}$ is applied to all codewords, placing their mapped representations in a common coordinate system. Let $\boldsymbol{c}_{m,k} \in \mathbb{R}^d$ denote the k-th codeword in the codebook associated with the m-th SID position. We initialize the corresponding code-token embedding as

$$\tilde{\boldsymbol{e}}_{m,k} = \boldsymbol{W}\boldsymbol{c}_{m,k}, \qquad \gamma = \frac{\sigma_{\text{init}}}{\sigma_{\text{map}}}, \qquad \boldsymbol{e}_{m,k}^{(0)} = \gamma \tilde{\boldsymbol{e}}_{m,k},$$

where σ_{map} denotes the global standard deviation of the mapped codeword embeddings, and γ rescales the mapped embeddings to match the generator's default embedding initialization scale σ_{init}. By construction, the resulting code-token embeddings preserve the geometry of the codeword space up to a global scale. Specifically, for any two codewords indexed by i and j, we have

$$\left\| \boldsymbol{e}_i^{(0)} - \boldsymbol{e}_j^{(0)} \right\|_2 = \gamma \left\| \boldsymbol{c}_i - \boldsymbol{c}_j \right\|_2, \quad \cos\left(\boldsymbol{e}_i^{(0)}, \boldsymbol{e}_j^{(0)}\right) = \cos(\boldsymbol{c}_i, \boldsymbol{c}_j), \quad \left\langle \boldsymbol{e}_i^{(0)}, \boldsymbol{e}_j^{(0)} \right\rangle = \gamma^2 \langle \boldsymbol{c}_i, \boldsymbol{c}_j \rangle.$$

Alignment in a Shared Space. The above formulation assumes that all codewords reside in the same vector space as the original item representation. In a general RPQ-Tree, however, the transformations introduced by P-nodes place codewords in branch-specific spaces with different orientations and dimensionalities. Therefore, before applying the shared mapping $\boldsymbol{W}$, all codewords must first be expressed in the original item-representation space.

To this end, we define the inverse mapping from a P-node's child subspaces back to its input space, which corresponds to the single-branch form of its reconstruction operation. Consider a P-node $P_G(T)$ with an orthogonal rotation $\boldsymbol{R}$ and G child sub-trees, each operating in a d_s-dimensional subspace. For a codeword $\boldsymbol{c}_{m,k} \in \mathbb{R}^{d_s}$ in the g-th sub-tree, we first embed it into the full rotated space by zero-padding the coordinates outside the corresponding subspace:

$$\mathrm{Pad}_{g,G}^{d_s}(\boldsymbol{c}_{m,k}) = \Big[\underbrace{0,\ldots,0}_{(g-1)d_s},\ \boldsymbol{c}_{m,k},\ \underbrace{0,\ldots,0}_{(G-g)d_s}\Big] \in \mathbb{R}^{Gd_s}.$$

We apply the inverse rotation $\boldsymbol{R}^\top = \boldsymbol{R}^{-1}$ to map the padded codeword back to the input space:

$$\bar{\boldsymbol{c}}_{m,k} = \boldsymbol{R}^\top \mathrm{Pad}_{g,G}^{d_s}(\boldsymbol{c}_{m,k}).$$

Traceable and Learnable. The node-level inverse mapping can be applied recursively. Starting from a quantization node, we trace its codeword along the path to the root and apply this mapping whenever a P-node is encountered, ultimately obtaining the codeword's reconstruction contribution. Such scheme transfers the geometric structure learned during SID construction into the initial code-token embeddings. This scheme transfers the geometric structure learned during SID construction into the initial code-token embeddings, which remain fully trainable and are further adapted to the recommendation objective during training.

3.3 Progressive Training

Hierarchy-Guided Hard-Negative Mining. The hierarchical structure of RPQ-Tree captures item characteristics at multiple levels of granularity, with deeper levels generally reflecting finer-grained distinctions. We exploit this hierarchy to progressively identify hard negatives during training. Specifically, given an RPQ-Tree of depth D, we divide training into up to D stages. At stage d, given $\boldsymbol{h}_u$ and the target item i^+, we define the hardness score of item j as $s_j^{\mathrm{hard},(d)} = s_{d-1}(j \mid \boldsymbol{h}_u)$, using only the first $d-1$ levels in Eq. 2. The hard-negative set is then obtained by

$$\mathcal{N}_{i^+}^{\mathrm{hard},(d)} = \mathrm{TopK}_{j \in \mathcal{I}\setminus\{i^+\}}(s_j^{\mathrm{hard},(d)}, n^{\mathrm{hard}}),$$

where $\mathcal{I}$ is the item catalog and n^{hard} the number of hard negatives. As d increases, progressively deeper levels are incorporated into the mining process, enabling the model to focus on increasingly fine-grained item confusions.

Weighted Cross-Entropy. At SID position m, let $y_m = z_{i^+,m}$ denote the target code and $\ell_{m,k}$ the pre-softmax logit for code k from a codebook of size K. The standard cross-entropy is

$$\mathcal{L}_m^{\mathrm{CE}} = -\ell_{m,y_m} + \log\Big(\exp(\ell_{m,y_m}) + \sum\nolimits_{k \neq y_m} \exp(\ell_{m,k})\Big),$$

where all non-target codes are equally weighted. To emphasize more confusable non-target codes, we estimate a position-wise confusion distribution from the retrieved hard negatives:

$$n_{m,k}^{(d)} = \sum\nolimits_{j \in \mathcal{N}_{i^+}^{\mathrm{hard},(d)}} \mathbb{I}(z_{j,m} = k)\,, \qquad q_{m,k}^{(d)} = \frac{n_{m,k}^{(d)}}{\sum_{r \neq y_m} n_{m,r}^{(d)}}, \quad k \neq y_m.$$

Here, $n_{m,k}^{(d)}$ counts the occurrences of code k at position m among the hard negatives, while $q_{m,k}^{(d)}$ characterizes how frequently each non-target code is confused with the target under the current stage. We then re-weight the non-target codes according to this confusion distribution:

$$w_{m,y_m}^{(d)} = 1, \qquad w_{m,k}^{(d)} = (1-\beta) + (K-1)\beta q_{m,k}^{(d)}, \quad k \neq y_m,$$

where $\beta \in [0, 1]$ controls the re-weighting strength. By construction, $\sum_{k \neq y_m} w_{m,k}^{(d)} = K - 1$, so the total non-target weight remains unchanged while greater emphasis is placed on more frequently confused codes. The resulting code-level cross-entropy is

$$\mathcal{L}_m^{\mathrm{WCE},(d)} = -\ell_{m,y_m} + \log \sum\nolimits_{k=1}^{K} w_{m,k}^{(d)} \exp(\ell_{m,k}).$$

If $\sum_{r \neq y_m} n_{m,r}^{(d)} = 0$, we set $w_{m,k}^{(d)} = 1$ for all k, recovering the standard cross-entropy. Code-level losses are then averaged within each layer and weighted across layers to form the final loss:

$$\mathcal{L}_l^{\mathrm{WCE},(d)} = \frac{1}{|C_l|} \sum\nolimits_{m \in C_l} \mathcal{L}_m^{\mathrm{WCE},(d)}, \qquad \mathcal{L}^{\mathrm{WCE},(d)} = \sum\nolimits_{l=1}^{D} \alpha_l^{\mathrm{ce}} \mathcal{L}_l^{\mathrm{WCE},(d)}.$$

Catalog-Level Ranking. To align position-wise SID supervision with item-level ranking, we further introduce a catalog-level ranking loss. At stage d, we construct the candidate set

$$\mathcal{S}_{i^+}^{(d)} = \{i^+\} \cup \mathcal{N}_{i^+}^{\mathrm{hard},(d)} \cup \mathcal{N}_{i^+}^{\mathrm{rand}},$$

where $\mathcal{N}_{i^+}^{\mathrm{rand}}$ contains n^{rand} uniformly sampled random negatives, providing broader catalog coverage in addition to locally confusable hard negatives. Each candidate is scored using using Eq. 2 as s_j^{full}. The catalog-level ranking loss is then defined with ranking temperature τ as

$$\mathcal{L}^{\mathrm{Rank},(d)} = -\log \frac{\exp(s_{i^+}^{\mathrm{full}}/\tau)}{\sum_{j \in \mathcal{S}_{i^+}^{(d)}} \exp(s_j^{\mathrm{full}}/\tau)},$$

Progressive Training. Given an RPQ-Tree of depth D, we divide training into D consecutive stages. The first stage serves as a warm-up and optimizes the standard cross-entropy objective. From stage 2 onward, hard negatives are progressively mined using the preceding tree levels, and the model is optimized with both code-level and item-level supervision:

$$\mathcal{L}^{(d)} = \begin{cases} \mathcal{L}^{\mathrm{CE}}, & d = 1, \\ \mathcal{L}^{\mathrm{WCE},(d)} + \lambda \mathcal{L}^{\mathrm{Rank},(d)}, & 2 \leq d \leq D, \end{cases}$$

where λ weights the ranking objective. As training proceeds to deeper stages, hard-negative mining progressively narrows the candidate region using finer-grained hierarchical information, thereby shifting the learning focus from broad item distinctions toward more localized discrimination.

4 Experiments

We conduct extensive experiments and aim to answer the following research questions.

- **RQ1:** Does Cuber outperform state-of-the-art generative recommendation methods?
- **RQ2:** How do different tree structures and hyperparameter settings affect Cuber?
- **RQ3:** How does each component of Cuber contribute to its overall performance?

4.1 Experimental Settings

Datasets. We adopt four subsets of the Amazon Reviews Datasets (McAuley et al., 2015): *Sports and Outdoors*, *Beauty*, *Toys and Games* and *CDs and Vinyl*; among them, CDs and Vinyl is substantially larger than the other three. Following previous studies (Hou et al., 2023), we treat each user review as an interaction and construct user interaction sequences based on review timestamps. We employ the leave-last-out protocol, using the most recent interaction of each user for testing and the second most recent one for validation. The statistics of the four datasets are summarized.

Baselines. We compare proposed Cuber with representative GR methods, which can be broadly categorized into two groups as follows: Item-ID-based, including Caser (Tang & Wang, 2018), GRU4Rec (Hidasi et al., 2016), HGN (Ma et al., 2019), BERT4Rec (Sun et al., 2019), SASRec (Kang & McAuley, 2018), FDSA (Zhang et al., 2019), and S^3-Rec (Zhou et al., 2020); and Semantic-ID-based methods, including VQ-Rec (Hou et al., 2023), RecJPQ (Petrov & Macdonald,

Table 2: Performance comparison. The best results are in bold, and second-best ones are underlined.

Model	Sports and Outdoors				Beauty				Toys and Games				CDs and Vinyl			
	R@5	N@5	R@10	N@10	R@5	N@5	R@10	N@10	R@5	N@5	R@10	N@10	R@5	N@5	R@10	N@10
Item ID-based																
Caser	0.0116	0.0072	0.0194	0.0097	0.0205	0.0131	0.0347	0.0176	0.0166	0.0107	0.0270	0.0141	0.0116	0.0073	0.0205	0.0101
GRU4Rec	0.0129	0.0086	0.0204	0.0110	0.0164	0.0099	0.0283	0.0137	0.0097	0.0059	0.0176	0.0084	0.0195	0.0120	0.0353	0.0171
HGN	0.0189	0.0120	0.0313	0.0159	0.0325	0.0206	0.0512	0.0266	0.0321	0.0221	0.0497	0.0277	0.0259	0.0153	0.0467	0.0220
BERT4Rec	0.0115	0.0075	0.0191	0.0099	0.0203	0.0124	0.0347	0.0170	0.0116	0.0071	0.0203	0.0099	0.0326	0.0201	0.0547	0.0271
SASRec	0.0233	0.0154	0.0350	0.0192	0.0387	0.0249	0.0605	0.0318	0.0463	0.0306	0.0675	0.0374	0.0351	0.0177	0.0619	0.0263
FDSA	0.0182	0.0122	0.0288	0.0156	0.0267	0.0163	0.0407	0.0208	0.0228	0.0140	0.0381	0.0189	0.0226	0.0137	0.0378	0.0186
S^3-Rec	0.0251	0.0161	0.0385	0.0204	0.0387	0.0244	0.0647	0.0327	0.0443	0.0294	0.0700	0.0376	0.0213	0.0130	0.0375	0.0182
Semantic ID-based																
VQ-Rec	0.0208	0.0144	0.0300	0.0173	0.0457	0.0317	0.0664	0.0383	0.0497	0.0346	0.0737	0.0423	0.0352	0.0238	0.0520	0.0292
RecJPQ	0.0141	0.0076	0.0220	0.0102	0.0311	0.0167	0.0482	0.0222	0.0331	0.0182	0.0484	0.0231	0.0075	0.0046	0.0138	0.0066
TIGER	0.0264	0.0181	0.0400	0.0225	0.0454	0.0321	0.0648	0.0384	0.0521	0.0371	0.0712	0.0432	0.0492	0.0329	0.0748	0.0411
HSTU	0.0258	0.0165	0.0414	0.0215	0.0469	0.0314	0.0704	0.0389	0.0433	0.0281	0.0669	0.0357	0.0417	0.0275	0.0638	0.0346
RPG	0.0286	0.0200	0.0419	0.0243	0.0505	0.0356	0.0734	0.0430	0.0547	0.0382	0.0787	0.0459	0.0481	0.0326	0.0708	0.0399
MHL	0.0356	0.0244	0.0509	0.0292	0.0569	0.0417	0.0784	0.0486	0.0640	0.0469	0.0883	0.0547	0.0341	0.0240	0.0486	0.0287
Cuber-A	0.0372	0.0254	**0.0542**	**0.0309**	0.0622	0.0423	**0.0918**	0.0518	0.0672	0.0462	**0.0966**	0.0557	**0.0667**	**0.0450**	**0.0971**	**0.0548**
Cuber-B	**0.0374**	**0.0256**	0.0538	**0.0309**	**0.0623**	**0.0431**	0.0908	**0.0523**	**0.0680**	**0.0479**	0.0958	**0.0569**	0.0658	0.0447	0.0960	0.0544

2024), TIGER (Rajput et al., 2023), HSTU (Zhai et al., 2024), RPG (Hou et al., 2025) and MHL (Wei et al., 2026). Cuber also belongs to this category.

Metrics. Performance is measured by Recall@K and NDCG@K, with $K \in \{5, 10\}$. We select the one with the highest NDCG@10 on validation set and report its performance on test set.

Implementation Details. For a fair comparison, we follow the settings of RPG wherever applicable, with necessary adaptations to our scheme. We encode item metadata using Sentence-T5-base (Ni et al., 2022), producing 768-dimensional textual embeddings, and use 128-dimensional item co-occurrence embeddings to incorporate collaborative signals. The textual and co-occurrence embeddings are separately ℓ_2-normalized, concatenated, and projected to 144 dimensions via PCA.[1] Following RPG, the recommendation backbone is a 2-layer Transformer decoder with a hidden size of 448, an FFN dimension of 1024, and 4 attention heads. All experiments are conducted on NVIDIA RTX GPUs and Ascend NPUs. Additional implementation details are provided in Appendix B.

4.2 Overall Performance (RQ1)

Table 2 summarizes the overall performance of each baseline and Cuber, where Cuber-A and Cuber-B adopt different RPQ-Tree structures.[2] To avoid the randomization introduced by graph decoding(Hou et al., 2025), we report results under full catalog rating for RPG, MHL and Cuber, leaving detailed discussion on graph decoding to Sec. 4.3. Overall, SID-based methods generally achieve better performance than item ID-based methods, highlighting the benefit of incorporating semantic information into item representations. Among the SID-based methods, the two Cuber variants outperform the competing baselines in nearly all settings. On Sports, Beauty, and Toys, Cuber improves Recall@10 and NDCG@10 over RPG by an average of 25.2% and 23.6%, respectively. It also achieves gains of 10.5% and 5.3% over MHL, without the additional training overhead required by MHL. The advantage of Cuber is more pronounced on CDs, where it improves Recall@10 and NDCG@10 over TIGER, the strongest baseline on this dataset, by 29.1% and 32.8%, respectively. In contrast, MHL exhibits a noticeable performance degradation on CDs, even under-performing RPG. These results suggest that Cuber scales well to larger datasets.

4.3 Key Parameter Analysis (RQ2)

Scalability of SID length. The SID length determines the capacity of the representation space and is an important factor affecting recommendation performance. As in Fig.4.3, both Cuber-A and Cuber-B generally achieve their best performance with an SID length of 32 across the three datasets. The only exception is Cuber-A on Sports, where a length of 64 is marginally better. In comparison, RPG performs best with a length of 16 on Sports and Toys. This difference suggests that, compared with the OPQ-based encoding adopted by RPG, RPQ-Tree exploits longer SIDs more effectively, allowing additional codes to capture useful information for recommendation. However,

[1] We use 144 dimensions instead of 128 as in RPG to allow even partitioning across different configurations.

[2] Cuber-A uses $R(P_8(Q), P_{24}(Q))$ and Cuber-B uses $P_8(R(Q, P_3(Q)))$, similar to Fig. 1(b) and (c).

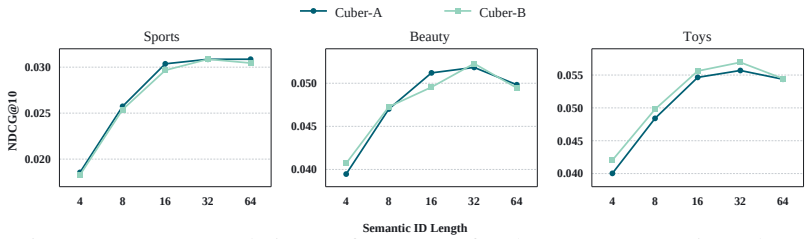

Figure 2: Recommendation performance of Cuber w.r.t. semantic ID length.

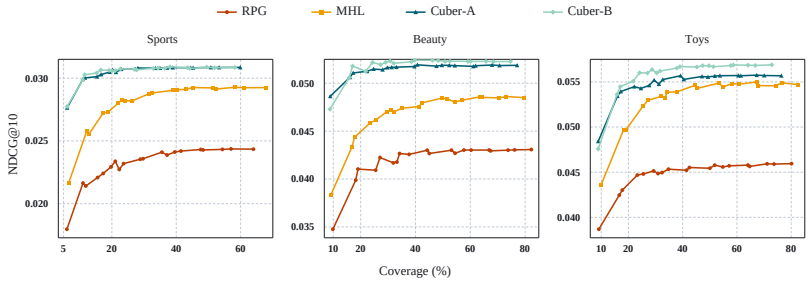

Figure 3: Recommendation performance of Cuber w.r.t. item coverage (step=3).

Table 3: Ablation Study.

Model	Cuber-A						Cuber-B					
	Sports		Beauty		Toys		Sports		Beauty		Toys	
	R@10	N@10	R@10	N@10	R@10	N@10	R@10	N@10	R@10	N@10	R@10	N@10
OPQ-32	.0420	.0227	.0787	.0450	.0815	.0464	.0420	.0227	.0787	.0450	.0815	.0464
RPQ-Tree	.0496	.0280	.0838	.0495	.0869	.0499	.0480	.0276	.0823	.0482	.0849	.0507
+ Init.	.0537	.0306	.0871	.0502	.0923	.0540	.0520	.0306	.0869	.0509	.0926	.0552
+ Train Obj.	.0542	.0309	.0918	.0518	.0966	.0557	.0538	.0309	.0908	.0523	.0958	.0569

further increasing the SID length to 64 leads to performance degradation for both Cuber variants. One possible explanation is that excessively fine-grained partitioning fragments semantically coherent information across subspaces, making the resulting representations more difficult for the recommendation models to learn. Based on these results, we set the SID length to 32.

Impact of Graph Decoding Parameters. Graph-based decoding (Hou et al., 2025) scores items within a subset of the full catalog to determine the final recommendation set. Its hyperparameters control the coverage of this subset and thereby affect the resulting recommendation performance. Fig. 3 shows NDCG@10 as a function of catalog coverage with the number of search steps fixed at 3. For all methods, performance generally improves with increasing coverage before gradually approaching a plateau. Notably, our method consistently outperforms RPG and MHL on all three datasets, indicating that its advantage is maintained under different graph-search budgets. See Appendix C for more explanations and results with one and two search steps.

Tree Structures and Training Parameters. Additional results for alternative RPQ-Tree structures and the effects of training hyperparameters are provided in Appendices E and F, respectively.

4.4 Further Analysis

Ablation Study (RQ3). We conduct ablation studies for Cuber, as in Table 3. We start from the OPQ-32 configuration adopted by RPG, while using co-occurrence-augmented item representations for SID construction. From this baseline, we first replace OPQ with RPQ-Tree, which yields substantial improvements across all datasets and demonstrates the effectiveness of proposed hierarchical quantization structure. We then introduce the proposed initialization scheme, which also brings consistent and notable gains across datasets and metrics. Finally, adding the progressive training objective further improves performance, with more pronounced gains in Recall than in NDCG, consistent with the design of the catalog-level ranking loss, which directly promotes the target item.

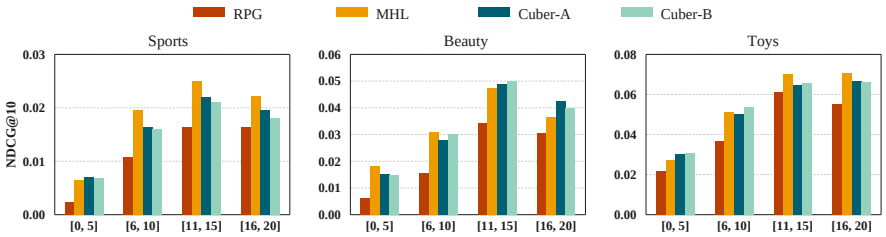

Figure 4: Cold-start recommendation performance comparison between RPG, MHL, and Cuber.

Cold-Start Scenario. Following RPG and MHL, we evaluate cold-start scenario, whose interaction frequencies fall into four ranges: $[0, 5]$, $[6, 10]$, $[11, 15]$, and $[16, 20]$, and report the performance separately for each range. As in Fig. 4, Cuber consistently outperforms RPG across all frequency groups, indicating improved robustness to sparse item interactions. Although Cuber achieves better overall performance than MHL on all three datasets, it outperforms MHL in 5 of the 12 cold-start settings, while MHL remains stronger in the remaining ones. This can be attributed to MHL's masked-history reconstruction objective, which provides additional contextual supervision for infrequent items beyond next-item prediction, making it advantageous in certain cold-item scenarios. Nonetheless, Cuber surpasses MHL in the sparsest frequency range (i.e., $[0, 5]$) on Sports and Toys.

5 Related Work

Sequential Recommendation. Sequential recommendation models temporally ordered user interactions to capture dynamic preferences and predict future behaviors. Early neural approaches mainly relied on recurrent, convolutional, or gating architectures to capture sequential dependencies and short-term transitions (Hidasi et al., 2016; Tang & Wang, 2018; Ma et al., 2019). Transformer-based models further improved sequence modeling through self-attention, using either causal next-item prediction or bidirectional masked-item reconstruction, with later variants further incorporating temporal intervals and heterogeneous item attributes (Kang & McAuley, 2018; Sun et al., 2019; Li et al., 2020; Zhang et al., 2019). To alleviate sparse supervision and improve representation quality, subsequent studies introduced auxiliary pre-training and contrastive objectives that exploit correlations among items, attributes, subsequences, and augmented sequence views (Zhou et al., 2020; Xie et al., 2022; Qiu et al., 2022). Despite these advances, conventional sequential recommenders typically rely on independently learned item ID embeddings, which neither explicitly capture inter-item semantics nor scale efficiently, as embedding parameters grow linearly with catalog size.

SID-based Generative Recommendation Generative recommendation (GR) reformulates next-item prediction as the generation of structured item identifiers. TIGER (Rajput et al., 2023) pioneered this paradigm by representing each item with a sequence of Semantic ID tokens through residual quantization (van den Oord et al., 2017; Zeghidour et al., 2022). Subsequent studies improve SID-based recommendation performance by incorporating linguistic and collaborative semantics (Zheng et al., 2024; Wang et al., 2024; Xiao et al., 2025; Ding et al., 2026), or jointly learning semantic IDs and recommendation models Liu et al. (2025); Fu et al. (2026). A recent line of work focuses on improving efficiency and expressiveness by drawing on alternative quantization structures (Hou et al., 2025; Huang et al., 2026), while Wei et al. (2026) further enhances the learning process by introducing richer supervision over user histories.

6 Conclusion

Generating semantic IDs in parallel suffers from poor reconstruction quality, while naively introducing additional residual refinement may incur extra inference cost. In this paper, we propose to flexibly combine PQ and RQ, enable the model to capture the hierarchical structure encoded in the SIDs, and progressively enhance fine-grained discrimination, while retaining single-forward inference. We then present Cuber, including a unified SID construction framework based on RPQ-Tree, a geometry-aware embedding initialization scheme, and a progressive training mechanism. Extensive experiments demonstrate that Cuber consistently outperforms competitive baselines across multiple key recommendation metrics.

AI USE STATEMENT

In this work, we used generative AI tools to propose or refine hypotheses, help develop theoretical models or conceptual frameworks, design or provide feedback on research methodology or experiments, implement methods and assist with translation. We have not used generative AI tools to generate synthetic data sets, clean and reformat dataset, support qualitative and thematic data analysis, nor interpret results. And formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs are not applicable to this work. Additionally, we used generative AI tools to create or modify scientific figures or images, creation of artifacts, draft parts of a research paper, edit a research paper to improve readability. We have reviewed all AI-assisted work. The LLM-generated test code was verified and tested for correctness by 2 authors. All manuscript content initially drafted with AI assistance was subsequently reviewed by all authors to ensure its accuracy and faithful communication of the intended information. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the datasets, preprocessing procedures, model configurations, training hyperparameters, and evaluation protocols in the main text 4.1 and appendix B. Anonymous source code is included in the supplementary material to facilitate reproduction of our results.

REFERENCES

Ben Chen, Xian Guo, Siyuan Wang, Zihan Liang, Yue Lv, Yufei Ma, Xinlong Xiao, Bowen Xue, Xuxin Zhang, Ying Yang, et al. Onesearch: A preliminary exploration of the unified end-to-end generative framework for e-commerce search. *arXiv preprint arXiv:2509.03236*, 2025.

Yongjian Chen, Tao Guan, and Cheng Wang. Approximate nearest neighbor search by residual vector quantization. *Sensors*, 10(12):11259–11273, 2010.

Chaoyue Ding, Jiahao Liu, Dongsheng Li, Shengkang Gu, Hansu Gu, Peng Zhang, Ning Gu, and Tun Lu. Unigcrec: Unified user-item quantization for generative cross-domain recommendation. In *Proc. of SIGKDD*, KDD '26, pp. 887–896, New York, NY, USA, 2026. ISBN 9798400722592. doi: 10.1145/3770855.3818132. URL https://doi.org/10.1145/3770855.3818132.

Junchen Fu, Xuri Ge, Alexandros Karatzoglou, Ioannis Arapakis, Suzan Verberne, Joemon M. Jose, and Zhaochun Ren. Differentiable semantic ID for generative recommendation. In *Proc. of SIGIR*, pp. 369–379. ACM, 2026.

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

Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. Recommendation as language processing (rlp): A unified pretrain, personalized prompt & predict paradigm (p5). In *Proc. of RecSys*, pp. 299–315, 2022.

Dan Guo, Chuanqing Li, and Lv Wu. Parametric and nonparametric residual vector quantization optimizations for ann search. *Neurocomputing*, 217:92–102, 2016.

Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In *Proc. of ICLR*, 2016.

Yupeng Hou, Zhankui He, Julian McAuley, and Wayne Xin Zhao. Learning vector-quantized item representation for transferable sequential recommenders. In *Proc. of WWW*, pp. 1162–1171, 2023.

Yupeng Hou, Jiacheng Li, Ashley Shin, Jinsung Jeon, Abhishek Santhanam, Wei Shao, Kaveh Hassani, Ning Yao, and Julian McAuley. Generating long semantic ids in parallel for recommendation. In *Proc. of SIGKDD*, pp. 956–966, 2025.

Wenyue Hua, Shuyuan Xu, Yingqiang Ge, and Yongfeng Zhang. How to index item ids for recommendation foundation models. In *Proc. of SIGIR-AP*, pp. 195–204, 2023.

Bin Huang, Xin Wang, Junwei Pan, Yongqi Zhou, Yifeng Zhou, Zhixiang Feng, Shudong Huang, Haijie Gu, and Wenwu Zhu. Asymmetric generative recommendation via kronecker residual bridge and multi-faceted hierarchical quantization, 2026. URL https://arxiv.org/abs/2605.14512.

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

Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *Proc. of ICDM*, pp. 197–206. IEEE, 2018.

Jiacheng Li, Yujie Wang, and Julian McAuley. Time interval aware self-attention for sequential recommendation. In *Proceedings of the 13th international conference on web search and data mining*, pp. 322–330, 2020.

Yu Liang, Zhongjin Zhang, Yuxuan Zhu, Kerui Zhang, Zhiluohan Guo, Wenhang Zhou, Zonqi Yang, Kangle Wu, Yabo Ni, Anxiang Zeng, et al. Rethinking generative recommender tokenizer: Recsys-native encoding and semantic quantization beyond llms. *arXiv preprint arXiv:2602.02338*, 2026.

Enze Liu, Bowen Zheng, Cheng Ling, Lantao Hu, Han Li, and Wayne Xin Zhao. Generative recommender with end-to-end learnable item tokenization. In *Proc. of SIGIR*, pp. 729–739, 2025.

Chen Ma, Peng Kang, and Xue Liu. Hierarchical gating networks for sequential recommendation. In *Proc. of SIGKDD*, pp. 825–833, 2019.

Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proc. of SIGIR*, pp. 43–52, 2015.

Jianmo Ni, Gustavo Hernandez Abrego, Noah Constant, Ji Ma, Keith Hall, Daniel Cer, and Yinfei Yang. Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models. In *Findings of the association for computational linguistics: ACL 2022*, pp. 1864–1874, 2022.

Aleksandr V Petrov and Craig Macdonald. Recjpq: training large-catalogue sequential recommenders. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pp. 538–547, 2024.

Ruihong Qiu, Zi Huang, Hongzhi Yin, and Zijian Wang. Contrastive learning for representation degeneration problem in sequential recommendation. In *Proceedings of the fifteenth ACM international conference on web search and data mining*, pp. 813–823, 2022.

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

Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proc. of CIKM*, pp. 1441–1450, 2019.

Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proc. of WSDM*, pp. 565–573, 2018.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Proc. of NeurIPS*, pp. 6309–6318, 2017. ISBN 9781510860964.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wenjie Wang, Honghui Bao, Xinyu Lin, Jizhi Zhang, Yongqi Li, Fuli Feng, See-Kiong Ng, and Tat-Seng Chua. Learnable item tokenization for generative recommendation. In *Proc. of CIKM*, pp. 2400–2409, 2024.

Kaiwen Wei, Xiaomian Kang, Jie Zhang, Yuming Yang, Li Jin, Zhenyang Li, Jiang Zhong, Richard He Bai, Junnan Zhu, et al. From past to path: Masked history learning for next-item prediction in generative recommendation. In *Proc. of ACL*, pp. 10421–10441, 2026.

Longtao Xiao, Haozhao Wang, Cheng Wang, Linfei Ji, Yifan Wang, Jieming Zhu, Zhenhua Dong, Rui Zhang, and Ruixuan Li. Unger: Generative recommendation with a unified code via semantic and collaborative integration. *ACM Trans. Inf. Syst.*, 44(2), 2025. ISSN 1046-8188. doi: 10.1145/3773771. URL https://doi.org/10.1145/3773771.

Xu Xie, Fei Sun, Zhaoyang Liu, Shiwen Wu, Jinyang Gao, Jiandong Zhang, Bolin Ding, and Bin Cui. Contrastive learning for sequential recommendation. In *2022 IEEE 38th international conference on data engineering (ICDE)*, pp. 1259–1273. IEEE, 2022.

Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi. Soundstream: An end-to-end neural audio codec. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:495–507, 2022. doi: 10.1109/TASLP.2021.3129994.

Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Jiayuan He, Yinghai Lu, and Yu Shi. Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations. In *Proc. of ICML*, volume 235 of *Proceedings of Machine Learning Research*, pp. 58484–58509, 2024.

Tingting Zhang, Pengpeng Zhao, Yanchi Liu, Victor S Sheng, Jiajie Xu, Deqing Wang, Guanfeng Liu, Xiaofang Zhou, et al. Feature-level deeper self-attention network for sequential recommendation. In *Proc. of IJCAI*, pp. 4320–4326, 2019.

Qihang Zhao, Zhongbo Sun, Xiaoyang Zheng, Xian Guo, Siyuan Wang, Zihan Liang, Mingcan Peng, Ben Chen, and Chenyi Lei. Coins: Semantic ids enhanced cold item representation for click-through rate prediction in e-commerce search. In *Proc. of WWW*, pp. 8529–8532, 2026.

Bowen Zheng, Yupeng Hou, Hongyu Lu, Yu Chen, Wayne Xin Zhao, Ming Chen, and Ji-Rong Wen. Adapting large language models by integrating collaborative semantics for recommendation. In *Proc. of ICDE*, pp. 1435–1448. IEEE, 2024.

Kun Zhou, Hui Wang, Wayne Xin Zhao, Yutao Zhu, Sirui Wang, Fuzheng Zhang, Zhongyuan Wang, and Ji-Rong Wen. S3-rec: Self-supervised learning for sequential recommendation with mutual information maximization. In *Proc. of CIKM*, pp. 1893–1902, 2020.

APPENDIX

A DETAILED ENCODING PROCEDURE

For the item embedding $\boldsymbol{e}_i$, the two trees in Fig. 1(b) and (c) produce SID $\boldsymbol{z}_i = (z_{i,1}, \ldots, z_{i,16})$ as follows. Consistent with Section 2, $Q_m(\cdot)$ returns the nearest codeword $\boldsymbol{q}_{i,m}$ in the m-th codebook, and $z_{i,m}$ denotes its index.

$R(P_4(Q), P_{12}(Q))$. The first product quantizer rotates $\boldsymbol{e}_i$ and partitions it into four 36-dimensional subvectors:

$$\begin{aligned} \boldsymbol{e'}^{c}_{i} &= \boldsymbol{R}^{c}\boldsymbol{e}_i = [\boldsymbol{e'}^{c}_{i,1}; \ldots; \boldsymbol{e'}^{c}_{i,4}], \\ \boldsymbol{q}_{i,m}, z_{i,m} &= Q_m(\boldsymbol{e'}^{c}_{i,m}), \qquad m = 1, \ldots, 4, \\ \hat{\boldsymbol{e}}^{c}_{i} &= (\boldsymbol{R}^{c})^{\top}[\boldsymbol{q}_{i,1}; \ldots; \boldsymbol{q}_{i,4}], \\ \boldsymbol{r}_{i,1} &= \boldsymbol{e}_i - \hat{\boldsymbol{e}}^{c}_{i}. \end{aligned}$$

The second product quantizer operates on this full-vector residual and produces the remaining twelve codewords:

$$\begin{aligned} \boldsymbol{e'}^{f}_{i} &= \boldsymbol{R}^{f}\boldsymbol{r}_{i,1} = [\boldsymbol{e'}^{f}_{i,1}; \ldots; \boldsymbol{e'}^{f}_{i,12}], \\ \boldsymbol{q}_{i,4+j}, z_{i,4+j} &= Q_{4+j}(\boldsymbol{e'}^{f}_{i,j}), \qquad j = 1, \ldots, 12, \\ \hat{\boldsymbol{e}}_i &= \hat{\boldsymbol{e}}^{c}_{i} + (\boldsymbol{R}^{f})^{\top}[\boldsymbol{q}_{i,5}; \ldots; \boldsymbol{q}_{i,16}]. \end{aligned}$$

Each subvector of ${e'}_i^f$ is 12-dimensional. The first stage yields $z_{i,1}, \ldots, z_{i,4}$, and quantization of $\boldsymbol{r}_{i,1}$ yields $z_{i,5}, \ldots, z_{i,16}$.

$P_4(R(Q, P_3(Q)))$**.** The outer product quantizer rotates $\boldsymbol{e}_i$ and partitions it into four 36-dimensional groups:

$$\boldsymbol{e'}_i = \boldsymbol{R}^o \boldsymbol{e}_i = [\boldsymbol{e'}_{i,1}; \ldots; \boldsymbol{e'}_{i,4}].$$

Each group has its own coarse codeword and residual. For $g = 1, \ldots, 4$,

$$\begin{aligned} \boldsymbol{r}_{i,g,0} &= \boldsymbol{e'}_{i,g}, \\ \boldsymbol{q}_{i,g}, z_{i,g} &= Q_g(\boldsymbol{r}_{i,g,0}), \\ \boldsymbol{r}_{i,g,1} &= \boldsymbol{r}_{i,g,0} - \boldsymbol{q}_{i,g}. \end{aligned}$$

A group-specific rotation splits $\boldsymbol{r}_{i,g,1}$ into three 12-dimensional subvectors. For each group $g = 1, \ldots, 4$, the residual $\boldsymbol{r}_{i,g,1}$ is rotated and split into three 12-dimensional subvectors:

$$\begin{aligned} {\boldsymbol{e'}}^f_{i,g} &= \boldsymbol{R}^1_g \boldsymbol{r}_{i,g,1} = [{\boldsymbol{e'}}^f_{i,g,1}; {\boldsymbol{e'}}^f_{i,g,2}; {\boldsymbol{e'}}^f_{i,g,3}], \\ \boldsymbol{q}_{i,3g+1+j}, z_{i,3g+1+j} &= Q_{3g+1+j}({\boldsymbol{e'}}^f_{i,g,j}), \qquad j = 1, 2, 3, \\ \hat{\boldsymbol{z}}_{i,g} &= \boldsymbol{q}_{i,g} + (\boldsymbol{R}^1_g)^\top [\boldsymbol{q}_{i,3g+2}; \boldsymbol{q}_{i,3g+3}; \boldsymbol{q}_{i,3g+4}], \\ \hat{\boldsymbol{e}}_i &= (\boldsymbol{R}^o)^\top [\hat{\boldsymbol{z}}_{i,1}; \ldots; \hat{\boldsymbol{z}}_{i,4}]. \end{aligned}$$

The coarse quantizers produce $z_{i,1}, \ldots, z_{i,4}$. The three residual quantizers in group g produce $z_{i,3g+2}$, $z_{i,3g+3}$, and $z_{i,3g+4}$; across four groups, these occupy positions 5–16.

B Additional Implementation Details

Dataset Preprocessing We use the 5-core subsets of Amazon Reviews 2014, in which each user and item has at least five interactions before data splitting. All review records are treated as implicit positive feedback without rating-based filtering. For each user, interactions are chronologically ordered and divided using a leave-one-out strategy: the most recent interaction is used for testing, the second most recent for validation, and all preceding interactions for training. The maximum user-history length is set to 50 across all datasets. For long training sequences, sliding windows are used to construct multiple next-item prediction instances, whereas validation and test instances retain only the 50 most recent interactions preceding the target item. Item textual representations are constructed by cleaning and concatenating available metadata fields, including title, price, brand, features, categories, and description. We primarily reuse the original implementation of RPG Hou et al. (2025) to perform the above data preprocessing.

Co-occurrence augmentation Since text encoders primarily capture item semantics from textual representations, it is common practice to augment these representations with collaborative information (Wang et al., 2024; Liang et al., 2026; Xiao et al., 2025; Ding et al., 2026). In Cuber, we capture such collaborative signals through a simple co-occurrence-based approach.

Let N denote the number of items and let $A \in \mathbb{R}^{N \times N}$ be the item–item co-occurrence matrix. For two distinct items i and j occurring at positions p_i and p_j in the same training sequence, an edge is added when their positional distance does not exceed a predefined window size w. Its weight is defined as

$$A_{ij} \mathrel{+}= \exp\left(-\frac{|p_i - p_j|}{w}\right), \qquad 0 < |p_i - p_j| \le w.$$

We use $w = 2$ in our experiments. The edges are inserted symmetrically, i.e., $A_{ij} = A_{ji}$, and self-co-occurrence edges are excluded.

Since frequently interacted items may otherwise dominate the co-occurrence graph, we normalize the matrix according to item interaction frequencies. Let f_i be the number of occurrences of item i in the training sequences and let

$$D_f = \mathrm{diag}(f_1, f_2, \ldots, f_N).$$

The normalized co-occurrence matrix is then given by

$$\widetilde{A} = D_f^{-\frac{1}{2}} A D_f^{-\frac{1}{2}}.$$

We apply truncated singular value decomposition to $\widetilde{A}$ and retain the leading $d_c = 128$ components:

$$E^{\mathrm{co}} = \mathrm{TruncatedSVD}_{d_c}(\widetilde{A}), \qquad E^{\mathrm{co}} \in \mathbb{R}^{N \times d_c}.$$

Each row is subsequently normalized to unit length:

$$\bar{\boldsymbol{e}}_i^{\mathrm{co}} = \frac{\boldsymbol{e}_i^{\mathrm{co}}}{\|\boldsymbol{e}_i^{\mathrm{co}}\|_2},$$

where items without observed training co-occurrences are assigned zero vectors.

Meanwhile, the textual description of each item is encoded using Sentence-T5, producing a $d_t = 768$ dimensional embedding $\boldsymbol{e}_i^{\mathrm{text}}$. After row-wise ℓ_2 normalization, the textual and co-occurrence representations are concatenated:

$$\boldsymbol{h}_i = \left[\bar{\boldsymbol{e}}_i^{\mathrm{text}}; \bar{\boldsymbol{e}}_i^{\mathrm{co}}\right] \in \mathbb{R}^{d_t + d_c}.$$

Finally, a whitened principal component analysis transformation is fitted using only items observed in the training interactions and is then applied to the complete item catalog:

$$\boldsymbol{z}_i = \mathrm{PCA}_d(\boldsymbol{h}_i), \qquad d = 144.$$

The resulting representation $\boldsymbol{z}_i$ jointly captures textual semantics and collaborative co-occurrence information and is subsequently used as the input to the semantic quantization procedure. Both the co-occurrence statistics and the PCA transformation are estimated without accessing validation or test interactions, thereby avoiding information leakage.

Key Prameter Settings In Table 4, we provide the parameter settings we used for main experiments in Table 2.

Table 4: Key parameter settings for main experiments.

parameter	Sports	Beauty	Toys	CDs
learning rate	0.01	0.01	0.01	0.003
warmup steps	7500	7500	7500	10000
β	0.1	0.1	0.1	0.1
λ	0.8	0.9	1.0	0.2
$(\alpha_1^{\mathrm{score}}, \alpha_2^{\mathrm{score}})$	(0.35, 0.65)	(0.35, 0.65)	(0.35, 0.65)	(0.5,0.5)
$(\alpha_1^{\mathrm{ce}}, \alpha_2^{\mathrm{ce}})$	(0.5,0.5)	(0.5,0.5)	(0.5,0.5)	(0.5,0.5)
n^{hard}	256	256	256	256
n^{rand}	256	256	256	256
τ	2	2	2	1
#epochs of 1st stage	20	20	20	20

C Graph Decoding Analysis

In graph-based decoding (Hou et al., 2025), the search is initialized with `num_beams` randomly sampled items, where each item is connected to its `n_edges` most similar neighbors. At each of the `propagation_steps` propagation steps, the neighbors of the current beam items are collected and deduplicated to form the candidate set. These candidates are then scored by the recommendation model, and the `num_beams` highest-scoring items are retained as the beams for the next step. The three parameters jointly determine the portion of the catalog explored during decoding and can therefore affect recommendation performance. In order to prevent the main comparison from depending on method-specific graph-search budgets, we use full-catalog decoding for RPG, MHL, and Cuber in the main experiments, and separately examine the impact of graph-decoding parameters.

Specifically, we vary `num_beams` $\in \{10, 20, 30, 40, 50\}$, `n_edges` $\in \{50, 100, 150, 200\}$, and `propagation_steps` $\in \{1, 2, 3\}$, resulting in 60 decoding configurations for each method–dataset pair. The results with `propagation_steps` $= 3$ are reported in 3 from 4.3, while the remaining results with one and two propagation steps are provided here for completeness. Cuber

achieves better performance than RPG and MHL under all evaluated configurations. Moreover, for every method–dataset pair, the best two-step result is within 1% of the corresponding best result with three steps. This indicates that most of the benefit from propagation is realized within the first two steps, while an additional third step provides only marginal improvement in peak ranking quality.

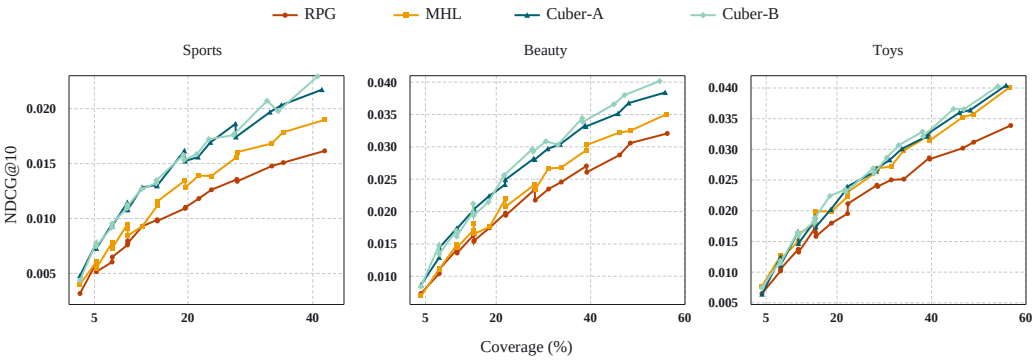

Figure 5: Recommendation performance of Cuber w.r.t. item coverage (step=1).

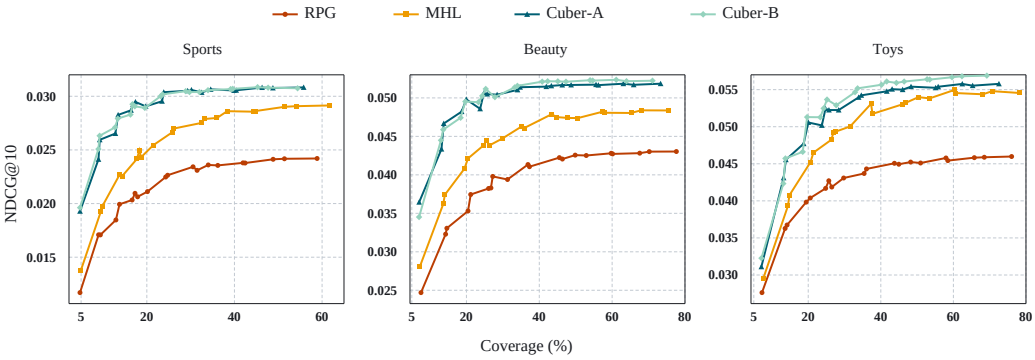

Figure 6: Recommendation performance of Cuber w.r.t. item coverage (step=2).

Note that recommendation performance is not strictly monotonic with catalog coverage for two reasons. First, coverage measures only the fraction of catalog items visited, rather than which items are explored; thus, different decoding configurations with similar coverage can produce different candidate sets and search trajectories. Second, even when the candidate set under one configuration is a strict superset of that under another, higher coverage does not necessarily improve ranking quality. If the target item is already included in the smaller candidate set, additional explored items with higher predicted scores may be ranked ahead of it in the larger set, thereby lowering its rank and potentially degrading NDCG. These effects jointly lead to local fluctuations in recommendation performance as coverage increases.

D Additional PCA-Dimension Results

To facilitate an even partition of the representation space across different code lengths and RPQ-Tree structures, Cuber adopts a 144-dimensional PCA representation, while RPG and MHL use 128 dimensions as specified in their original papers. To isolate the effect of PCA dimensionality, we additionally evaluate both methods with PCA-144 on the datasets where their original papers use 16-codeword encodings, allowing standard equal-width OPQ partitioning.

As shown in Table 5, increasing the PCA dimension from 128 to 144 does not yield a consistent improvement in recommendation performance. One possible explanation is that the additional

Table 5: Recommendation performance under different PCA dimensionalities for RPG and MHL.

PCA dim.	Sports (RPG)				Toys (RPG)			
	R@5	N@5	R@10	N@10	R@5	N@5	R@10	N@10
128	0.0286	0.0200	0.0419	0.0243	0.0547	0.0382	0.0787	0.0459
144	0.0281	0.0196	0.0415	0.0238	0.0541	0.0378	0.0778	0.0454

PCA dim.	Beauty (MHL)			
	R@5	N@5	R@10	N@10
128	0.0569	0.0417	0.0784	0.0486
144	0.0547	0.0405	0.0790	0.0483

Table 6: Recommendation performance and encoder training time of different tree structures on Sports and Outdoors. Best and second-best results are shown in bold and underlined, respectively.

Tree structure	R@5	N@5	R@10	N@10	Encoding time (s)
$P_{32}(Q)$ (OPQ-32)	0.0278	0.0181	0.0420	0.0227	15.24
$R(P_4(Q), P_{28}(Q))$	0.0364	0.0248	0.0530	0.0301	97.92
$R(P_8(Q), P_{24}(Q))$	0.0372	0.0254	<u>0.0542</u>	0.0309	<u>88.43</u>
$R(P_{12}(Q), P_{20}(Q))$	<u>0.0378</u>	0.0256	0.0537	0.0307	96.94
$R(P_{16}(Q), P_{16}(Q))$	**0.0387**	0.0258	**0.0545**	0.0309	102.21
$R(P_{20}(Q), P_{12}(Q))$	0.0373	0.0253	0.0528	0.0303	**84.29**
$R(P_{24}(Q), P_8(Q))$	0.0364	0.0246	0.0515	0.0295	101.20
$R(P_{28}(Q), P_4(Q))$	0.0335	0.0235	0.0460	0.0276	121.06
$P_8(R(Q, P_3(Q)))$	0.0374	0.0256	0.0538	0.0309	157.40
$P_8(R(P_2(Q), P_2(Q)))$	0.0358	0.0250	0.0507	0.0298	168.94
$P_8(R(P_3(Q), P_1(Q)))$	0.0351	0.0246	0.0471	0.0284	191.32
$P_4(R(P_2(Q), P_6(Q)))$	0.0375	<u>0.0261</u>	0.0538	**0.0313**	181.17
$P_4(R(P_4(Q), P_4(Q)))$	0.0363	0.0251	0.0522	0.0303	153.17
$P_4(R(P_6(Q), P_2(Q)))$	0.0344	0.0234	0.0496	0.0283	190.74
$P_2(R(P_4(Q), P_{12}(Q)))$	0.0363	0.0248	0.0515	0.0297	147.13
$R(P_8(Q), P_8(Q), P_{16}(Q))$	0.0376	0.0257	0.0537	0.0309	138.21
$R(P_4(Q), P_{12}(Q), P_{16}(Q))$	0.0376	0.0257	0.0532	0.0307	142.42
$P_8(R(Q, Q, P_2(Q)))$	0.0372	0.0255	0.0521	0.0303	194.31
$P_4(R(P_2(Q), P_2(Q), P_4(Q)))$	0.0377	**0.0262**	0.0528	<u>0.0310</u>	207.95

components retained by PCA-144 contain limited recommendation-relevant information and may introduce extra noise.

E Effect of Tree Structures

Experimental setup. Table E reports the recommendation performance and the encoder training time of Cuber with Sports under different RPQ-Tree structures. The first layer corresponds to the original OPQ-32 configuration; the second and third layers correspond to the two two-layer structures illustrated in Fig. E(b) and (c), respectively; and the fourth layer corresponds to the three-layer tree structure. We use the parameter configurations listed in Table 4 when applicable. For the three-layer structures, we set $(\alpha_1^{\text{score}}, \alpha_2^{\text{score}}, \alpha_3^{\text{score}}) = (0.35, 0.325, 0.325)$, $(\alpha_1^{\text{ce}}, \alpha_2^{\text{ce}}, \alpha_3^{\text{ce}}) = (0.5, 0.25, 0.25)$, and both the first and second training stages run for up to 20 epochs.

Discussion. The results show that allocating most quantizers to the first stage generally degrades performance, while balanced or residual-heavy allocations are more effective. The balanced structure $R(P_{16}(Q), P_{16}(Q))$ achieves the highest recall, whereas $P_4(R(P_2(Q), P_6(Q)))$ obtains the highest NDCG. Three-stage structures remain competitive but do not consistently outperform the simpler two-stage alternatives for this dataset.

Encoder training time also varies across tree structures. Compared with OPQ-32, more complex RPQ-tree structures increase the encoder training time, Structures of the form $R(P_a(Q), P_b(Q))$ are generally more efficient, with Cuber-A requiring only 88.43 seconds, close to the minimum of 84.29 seconds. Nested and three-stage structures are typically slower due to their greater structural complexity, but this additional cost is incurred only during offline SID construction.

Since the differences among most structures are modest, the main comparison adopts Cuber-A, $R(P_8(Q), P_{24}(Q))$, and Cuber-B, $P_8(R(Q, P_3(Q)))$, as representative configurations for comparison with other methods. We further conjecture that recursively deeper trees may become more beneficial on larger datasets, where additional hierarchy could better organize a substantially larger item space.

F Hyperparameter Sensitivity

We follow the implementation settings in Appendix B. To examine the two loss coefficients separately, we first vary the catalog-loss weight λ from 0 to 1 while holding all other hyperparameters fixed. We then fix λ to 0.8, 0.9, and 1.0 for Sports, Beauty, and Toys, respectively, and vary the weighted cross-entropy coefficient β from 0 to 0.20. Each configuration is evaluated using full-catalog ranking, and we report Recall@10 and NDCG@10.

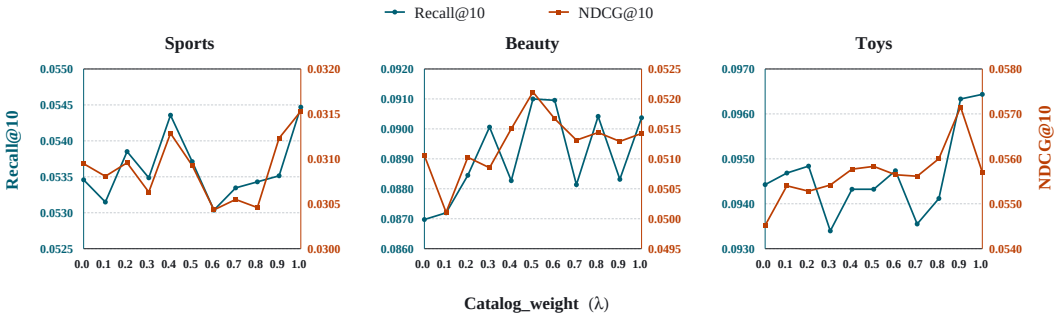

Figure 7: Effect of the catalog loss weight λ on Recall@10 and NDCG@10 across the three datasets.

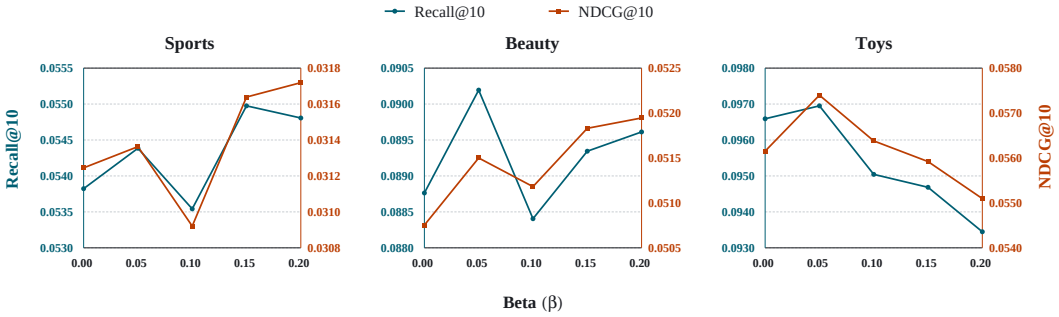

Figure 8: Effect of β on Recall@10 and NDCG@10 for $\beta \in [0.00, 0.20]$ across the three datasets.

Fig. 7 shows that the best observed catalog loss weight varies by dataset: both metrics peak at $\lambda = 1.0$ on Sports and $\lambda = 0.5$ on Beauty, while on Toys Recall@10 peaks at 1.0 and NDCG@10 at 0.9. Fig. 8 shows a non-monotonic response to β. Compared with $\beta = 0$, $\beta = 0.05$ improves both Recall@10 and NDCG@10 on all three datasets. The best observed β values for Recall@10 and NDCG@10 are 0.15 and 0.20 on Sports, 0.05 and 0.20 on Beauty, and 0.05 for both metrics on Toys.

G Additional observations of RQ and PQ-based methods

To better understand how SID construction affects prediction difficulty, we further analyze codeword-level accuracy using Top-1 and Top-5 metrics.

These results reveal different prediction patterns for RQ and OPQ codes. As shown in table 7, in RQ-4, the Top-1 accuracy drops sharply from 17.06% at the first position to 2.41% at the fourth

position, suggesting that later codewords are substantially harder to predict. In contrast, the four OPQ-4 positions have relatively uniform Top-1 accuracies, ranging from 7.89% to 9.23%.

Table 8 shows that increasing the OPQ code length also makes individual codewords harder to predict: the average Top-1 accuracy decreases from 8.44% for OPQ-4 to 4.71% for OPQ-16. As shown in table 9, in the hybrid OPQ-4 + OPQ-12 code, the first four positions retain a Top-1 accuracy of 8.02%, while the last twelve reach 3.05%. Nonetheless, Cuber still improves the final recommendation metrics through its reasonable design.

Table 7: The prediction accuracy of each codeword in RQ-4 and OPQ-4 SIDs

SID code position	RQ-4		OPQ-4	
	Top 1 Acc	Top 5 Acc	Top 1 Acc	Top 5 Acc
1	17.06%	38.79%	7.89%	20.48%
2	6.31%	15.51%	8.71%	20.95%
3	2.91%	8.26%	9.23%	21.51%
4	2.41%	7.23%	7.92%	21.11%

Table 8: Average prediction accuracy in different OPQ methods

method	average Top 1 Acc	average Top 5 Acc
OPQ-4	8.44%	21.02%
OPQ-8	6.02%	16.03%
OPQ-12	4.91%	13.51%
OPQ-16	4.71%	12.99%

Table 9: Prediction Accuracy for a hybrid OPQ-4 + OPQ-12 code.

SID code position	average Top 1 Acc	average Top 5 Acc
First 4	8.02%	20.50%
Last 12	3.05%	9.26%
All	4.29%	12.08%