RETVDQ: RETRIEVAL-AWARE VARIABLE-DEPTH RESIDUAL QUANTIZATION FOR VISUAL DOCUMENT RETRIEVAL

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-vector visual document retrieval (VDR) represents each page with multiple embeddings and scores it through MaxSim, which matches each query token to its most similar page embedding. This enables fine-grained retrieval but incurs a large storage cost. Existing compression mainly prunes or merges embeddings, but since different queries rely on different embeddings, removing any of them may discard needed evidence. Quantization keeps every embedding, yet existing quantizers only approximate the original vectors without preserving retrieval rankings, and assign the same precision to all embeddings regardless of their importance. We propose RetVDQ, a retrieval-aware variable-depth residual quantization framework that keeps every embedding and adapts its precision under a fixed budget. RetVDQ learns residual codebooks progressively so that every code prefix preserves the full-precision ranking, and then determines the depth of each embedding by the marginal retrieval utility of additional codes. Both stages are guided by synthetic queries generated offline from each page. The resulting index works with the frozen retriever and its original MaxSim scoring, requiring no model fine-tuning. Experiments on ViDoRe V1 and V2 show that RetVDQ consistently outperforms quantization baselines, and retains 99.3% of full-precision quality on average with only 3.2% of the original storage, far less than token reduction methods need for comparable quality.

1 INTRODUCTION

Visual document retrieval (VDR) aims to retrieve relevant pages from visually rich document collections. Unlike conventional methods (Xu et al., 2020; Appalaraju et al., 2021) that typically rely on OCR or text serialization, recent ColPali-style retrievers (Faysse et al., 2025) directly encode each page image into multiple contextualized document embeddings and match them against query-token embeddings through the MaxSim operation, where each query token selects its best-matching document embedding. This multi-vector representation preserves fine-grained evidence across text, tables, figures, and layout, making it well suited to visually rich documents. Yet this expressive representation incurs heavy storage overhead, as one page may require hundreds to thousands of document vectors, inflating the document-side index for large-scale corpora.

To reduce this storage cost, existing VDR compression methods mainly explore pruning and merging. Pruning removes embeddings (Ma et al., 2025) and merging combines them (Yan et al., 2026), both of which shrink the set of document embeddings that MaxSim selects from (see Fig. 1(a)). Since MaxSim scores a page by taking, for each query token, the maximum similarity over all document embeddings, an ideal compression scheme should reduce storage without altering this candidate set. Quantization (Jégou et al., 2011) meets this requirement by keeping every embedding and only reducing the bits used to store each, so the maximum is still taken over the same embeddings, with only their similarities perturbed by approximation. This makes quantization a natural fit for MaxSim-based VDR.

However, existing quantizers are largely uniform (see Fig. 1(b)), allocating the same number of bits to every document embedding. Under MaxSim, the contributions of document embeddings are highly uneven. Some carry the textual or visual evidence that query tokens actually select, while

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.

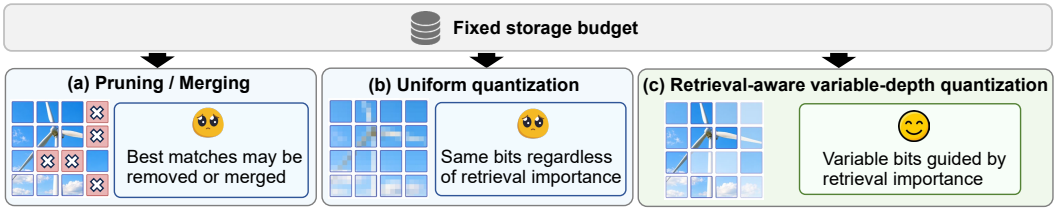


Figure 1: Comparison between existing compression approaches and our retrieval-aware method.

many others are rarely selected. Therefore, precision should instead be allocated across embeddings according to their retrieval importance. Residual quantization (RQ) (Chen et al., 2010) is well suited to this. It encodes an embedding as a sequence of codewords, each capturing the residual left by the previous ones, so that any prefix of its codes is itself a valid approximation. The number of retained codes therefore serves as a per-embedding quantization depth, enabling variable-depth quantization (see Fig. 1(c)). Yet applying standard RQ to this variable-depth setting faces two key challenges.

C1: Prefix-Level Retrieval Preservation. Standard RQ learns its residual codebooks solely by minimizing reconstruction error, without explicitly preserving retrieval behavior. Although some retrieval-aware quantizers (Yang et al., 2022; Salla et al., 2026) introduce ranking-preservation objectives based on the full-precision retriever, they supervise only the final quantized representation. This is insufficient for our variable-depth setting, where different embeddings may retain different numbers of residual codes and thus different prefixes may be used for retrieval. The core challenge is therefore to progressively learn residual codebooks such that every prefix preserves the retrieval behavior of the full-precision representation. Once this holds, the depth of each embedding can be chosen freely at inference time under a fixed storage budget, since any prefix is guaranteed to be a valid retrieval representation.

C2: Interaction-Aware Depth Allocation. With the learned residual codebooks, each embedding can be truncated at any depth, so a fixed storage budget can be distributed unevenly according to retrieval benefit. Since MaxSim keeps only the best-matching embedding for each query token, a refined embedding may take over or lose this match in the compressed index, so the gain of refining one embedding changes as the others are refined. The challenge is therefore to decide the depths of all embeddings in a page jointly, so that the budget best preserves the retrieval score of the page.

To address these challenges, we propose RetVDQ, a retrieval-aware variable-depth residual quantization framework for frozen ColPali-style retrievers. RetVDQ adopts a decoupled design that separates hierarchy learning from depth allocation: training is confined to the hierarchy-learning stage, yielding a reusable hierarchy and a training-free allocation across storage budgets. The hierarchy is learned by **retrieval-aware progressive codebook learning**, which trains codebooks from shallow to deep so that each prefix preserves the ranking behavior of the full-precision retriever. Given the learned hierarchy, **retrieval-aware residual-depth allocation** assigns a depth to each embedding under a shared page-level budget, spending more codes where they most improve retrieval. To support the training of both stages, we introduce **page-grounded retrieval supervision construction**.

Our main contributions are summarized below.

- We design a progressive codebook learning scheme in which each residual level repairs the ranking distortion of its prefix, making every code prefix retrieval-reliable.
- We develop a retrieval-aware allocation strategy that assigns codes by their marginal retrieval gain under MaxSim. To the best of our knowledge, this is the first work to vary quantization precision across embeddings in multi-vector visual document retrieval.
- Experiments on ViDoRe V1 and V2 show that RetVDQ consistently outperforms quantization baselines, retaining 99.3% of full-precision quality on average with only 3.2% of the original storage—far less than token reduction methods need for comparable quality.

2 Problem Formulation

A ColPali-style retriever (Faysse et al., 2025) encodes a query q into token embeddings $\{\mathbf{q}_i\}_{i=1}^{N_q}$ and a page D into M_D contextualized document embeddings $\mathbf{X}_D = \{\mathbf{x}_{D,j}\}_{j=1}^{M_D}$, each associated

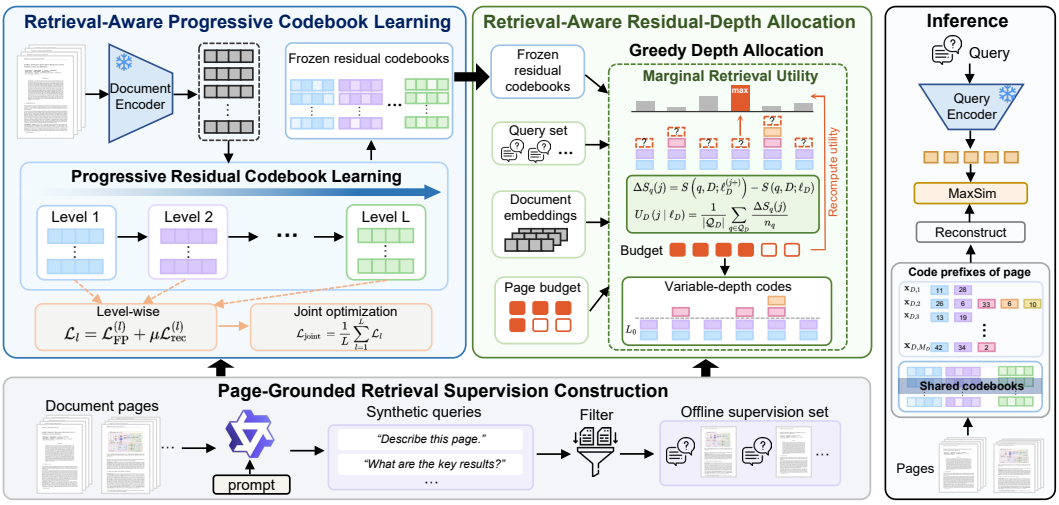


Figure 2: Overview of RetVDQ. Page-grounded retrieval supervision construction provides offline pseudo queries, which guide retrieval-aware progressive codebook learning and retrieval-aware residual-depth allocation. The compressed index is used for standard MaxSim scoring at inference.

with an image patch or a prompt token. Here N_q is the number of query token embeddings, and all embeddings are ℓ_2-normalized in $\mathbb{R}^d$. The relevance score is computed by late interaction:

$$S(q, D) = \sum_{i=1}^{N_q} \max_{1 \le j \le M_D} \mathbf{q}_i^\top \mathbf{x}_{D,j}. \tag{1}$$

Given a corpus $\mathcal{D}$ of pages, retrieval ranks every $D \in \mathcal{D}$ by $S(q, D)$. As pages of a given corpus are encoded offline and stored in the index, storing each page D at full precision costs $M_D \times d \times b_{\text{fp}}$ bits, where b_{fp} is the bit width of one scalar. In standard settings, a page is typically represented by hundreds to thousands of embedding tokens, so index storage scales rapidly with corpus size.

Therefore, given a frozen ColPali-style retriever and a corpus $\mathcal{D}$, we aim to reduce the storage of document embeddings. The embeddings $\mathbf{X}_D$ of each page $D \in \mathcal{D}$ are compressed into a compact representation, from which approximate embeddings $\widehat{\mathbf{X}}_D$ are reconstructed at retrieval time. Pages are then scored by Eq. (1) with $\widehat{\mathbf{X}}_D$ in place of $\mathbf{X}_D$, while the query encoder remains unchanged. We denote this score by $\widehat{S}(q, D)$ and the original score by $S_{\text{FP}}(q, D)$. Let B_{index} be the total number of bits stored for retrieval, covering the compact representations of all pages and any parameters shared across pages. Given a storage budget of B bits, the goal is:

$$\min \ \mathbb{E}_q\left[\Delta_q\left(\widehat{S}, S_{\text{FP}}\right)\right] \quad \text{s.t.} \quad B_{\text{index}} \le B, \tag{2}$$

where Δ_q measures how much the ranking of $\mathcal{D}$ under $\widehat{S}$ deviates from that under S_{FP} for query q.

3 Method

In this section, we present RetVDQ, which compresses the document index of a frozen ColPali-style retriever under the storage budget in Eq. (2). As illustrated in Figure 2, RetVDQ adopts a decoupled design that separates corpus-shared hierarchy learning from page-specific discrete allocation. This decoupling confines training to the hierarchy-learning stage, yielding a reusable hierarchy and a training-free allocation across storage budgets. Specifically, RetVDQ comprises three components: 1) page-grounded retrieval supervision construction, producing pseudo queries that supervise the subsequent two components; 2) retrieval-aware progressive codebook learning, deriving the residual codebooks via two stages: progressive residual codebook learning and joint codebook optimization; 3) retrieval-aware residual-depth allocation, which decides a per-embedding residual depth under a shared page-level budget, allocating more codes to embeddings whose retrieval score benefits most.

3.1 PAGE-GROUNDED RETRIEVAL SUPERVISION CONSTRUCTION

In MaxSim retrieval, the document embeddings that contribute to a page score depend on the query. We thus use queries to identify retrieval-sensitive embeddings and guide both codebook learning and residual-depth allocation. As test queries are unavailable when the index is constructed, we build a pseudo query set $\mathcal{Q}_D$ for each page D in the target corpus. Specifically, a frozen vision-language model G_ψ iteratively generates candidate queries conditioned on the page image and a fixed instruction prompt π, until N_{syn} valid pseudo queries are retained. The instruction asks for queries that are distinct, self-contained, and answerable from the visible content of D. We first filter candidates by format and uniqueness, and then retain only those for which the frozen full-precision retriever ranks D within its top-K_{f} results. This prevents semantically valid but retrieval-inconsistent queries from providing misleading supervision for document embedding quantization.

3.2 RETRIEVAL-AWARE PROGRESSIVE CODEBOOK LEARNING

Inspired by its progressive residual structure, which facilitates adaptive-depth representation, we adopt Residual Quantization (Chen et al., 2010) to reduce B_{index}. Specifically, RQ approximates each vector $\mathbf{x} \in \mathbb{R}^d$ with L codebooks $\mathbf{C}_1, \ldots, \mathbf{C}_L \in \mathbb{R}^{K \times d}$, each containing K codewords, where $\mathbf{C}_l[k]$ denotes the k-th codeword of $\mathbf{C}_l$. Starting from $\widehat{\mathbf{x}}^{(0)} = \mathbf{0}$, level l selects the codeword closest to the residual left by the preceding levels:

$$z_l = \arg\min_{k \in \{0,\ldots,K-1\}} \left\| \mathbf{x} - \widehat{\mathbf{x}}^{(l-1)} - \mathbf{C}_l[k] \right\|_2^2, \qquad \widehat{\mathbf{x}}^{(l)} = \widehat{\mathbf{x}}^{(l-1)} + \mathbf{C}_l[z_l], \qquad l = 1, \ldots, L. \tag{3}$$

Standard RQ learns codebooks solely from reconstruction error, without leveraging ranking signals from the full-precision retriever. Several retrieval-aware quantizers (Yang et al., 2022; Salla et al., 2026) instead incorporate ranking-preservation objectives based on the full-precision retriever to retain retrieval capability after quantization. However, they only supervise the overall quantized representation rather than individual prefixes. As any prefix in Eq. (3) may be retained after depth allocation, each prefix should be retrieval-effective. Moreover, residual prefixes are nested, as each codebook refines what the preceding ones leave behind. We exploit this structure by learning the codebooks progressively from shallow to deep. With the preceding prefix fixed, each new codebook is trained to repair the retrieval discrepancy left by this prefix.

Progressive Residual Codebook Learning. In this part, at level l, we fix $C_1, \ldots, C_{l-1}$ and optimize only C_l toward retaining retrieval capacity. Notably, following standard RQ, we first initialize the residual hierarchy by learning $C_1, \ldots, C_L$ sequentially with k-means on the residuals left by the preceding levels. Then, at each quantization level, we align the depth-l and full-precision score margins using a positive-negative page pair. Specially, we have

$$\begin{cases} \mathcal{L}_{\text{FP}}^{(l)} = \dfrac{1}{|\mathcal{T}^{(l)}|} \displaystyle\sum_{(q,D^+,D^-) \in \mathcal{T}^{(l)}} \dfrac{\rho(M^{(l)} - M^{\text{FP}})}{N_q}, \\ M^{\text{FP}} = S_{\text{FP}}(q, D^+) - S_{\text{FP}}(q, D^-), \\ M^{(l)} = S_l(q, D^+) - S_l(q, D^-), \end{cases} \tag{4}$$

where $\mathcal{T}^{(l)}$ is the set of training triplets and ρ denotes the SmoothL1 loss, used to reduce the influence of extreme margin outliers. Here, M^{FP} denotes the full-precision score margin specifying the desired degree of separation between the positive and negative pages, while $M^{(l)}$ denotes the score margin obtained from the representation formed by the first l residual levels, $C_1, \ldots, C_l$. This soft alignment helps correct quantization-induced ranking distortions. As these margins scale with the number of query tokens, we normalize them by N_q, the number of query tokens, to balance supervision across queries of different lengths.

Since the preceding prefix is fixed during level-l optimization, $M^{(l-1)}$ is constant. Therefore, Eq. (4) can equivalently be viewed as fitting the margin change $M^{(l)} - M^{(l-1)}$ introduced by the new residual level to the remaining discrepancy $M^{\text{FP}} - M^{(l-1)}$. In this sense, each new residual level is trained to repair the retrieval distortion left by its preceding prefix.

Retrieval- and compression-aware hard-negative mining. To better support the above margin alignment, rather than simply randomly sampling negative pages, we construct a level-specific hard-negative set that covers both retrieval-hard negatives and compression-sensitive negatives. **(1) Retrieval-difficulty negatives** capture negatives that are intrinsically difficult to distinguish from the positive under a given retrieval representation, as indicated by their absolute ranking scores. This category comprises two complementary sources. *Prefix-hard negatives* are the top-H_0 non-relevant pages under S_{l-1}, which are difficult to distinguish under the current prefix and thus expose competitors that the new residual level needs to separate. *FP-hard negatives* are the top non-relevant page under S_{FP}, which identifies a challenging full-precision ranking relationship that should be preserved after quantization. **(2) Compression-discrepancy negatives** capture negatives whose retrieval behavior changes substantially under compression, as measured by the discrepancy between the full-precision and compressed representations. This category considers compression-induced degradation from two complementary perspectives. *Margin-based negatives* are the pages with the largest margin loss $M^{\mathrm{FP}} - M^{(l-1)}$, indicating substantial distortion of the full-precision score margin. *Ranking-based negatives* are the highest-ranked pages in the S_{l-1} top-κ that fall outside the S_{FP} top-κ, capturing compression-induced ranking loss. The resulting hard-negative set $\mathcal{N}_q^{(l)}$ provides the negative pages for $\mathcal{T}^{(l)}$.

Objective Function. Following Salla et al. (2026), we also regularize the codebooks with the reconstruction objective, to stabilize retrieval-aware codebook optimization:

$$\mathcal{L}_{\mathrm{rec}}^{(l)} = \frac{1}{|\mathcal{D}|} \sum_{D \in \mathcal{D}} \frac{1}{M_D} \sum_{j=1}^{M_D} \left\| \mathbf{x}_{D,j} - \widehat{\mathbf{x}}_{D,j}^{(l)} \right\|_2^2. \tag{5}$$

Ultimately, the progressive residual codebook learning objective at each level l can be written as:

$$\mathcal{L}_l = \mathcal{L}_{\mathrm{FP}}^{(l)} + \mu\, \mathcal{L}_{\mathrm{rec}}^{(l)}, \tag{6}$$

where μ is the trade-off hyper-parameter.

Joint Codebook Optimization. Progressive learning never revisits a codebook after later levels are added, so earlier codebooks cannot adapt to their effects on deeper prefixes. We therefore unfreeze all codebooks and jointly minimize the average objective over all prefixes:

$$\mathcal{L}_{\mathrm{joint}} = \frac{1}{L} \sum_{l=1}^{L} \mathcal{L}_l. \tag{7}$$

After optimization, all codebooks are then frozen, and the target corpus is encoded once with Eq. (3).

3.3 Retrieval-Aware Residual-Depth Allocation

With the codebooks learned in Sec. 3.2, each embedding can be represented at any depth from 1 to L. This section decides which depth each embedding keeps under the storage budget.

Depth Allocation Formulation. We allocate the retained residual code depth of each document embedding using a fixed base depth and an adaptive additional depth. Every embedding initially retains the first L_0 codes, which guarantees a baseline approximation so that no embedding is left too coarse to be matched. On top of this baseline, each page D receives a budget of B_D additional codes to distribute among its own embeddings. Formally, let $\boldsymbol{\ell}_D = (\ell_{D,1}, \ldots, \ell_{D,M_D})$ denote the depth map of page D, where $\ell_{D,j} \in \{L_0, \ldots, L\}$ specifies the retained depth of its j-th document embedding. The allocation is then subject to $\sum_{j=1}^{M_D} \left(\ell_{D,j} - L_0\right) \le B_D$.

Marginal Retrieval Utility. To decide which embeddings within a page should receive the extra codes, we compare the page score before and after adding one more code to a document embedding. Let $S(q, D; \boldsymbol{\ell}_D)$ be the score of Eq. (1) when each $\mathbf{x}_{D,j}$ is replaced by its depth-$\ell_{D,j}$ reconstruction $\widehat{\mathbf{x}}_{D,j}^{(\ell_{D,j})}$. Let $\boldsymbol{\ell}_D^{(j+)}$ be the depth map in which only $\ell_{D,j}$ is increased by one. The resulting score change for query q is

$$\Delta S_q(j) = S\big(q, D; \boldsymbol{\ell}_D^{(j+)}\big) - S\big(q, D; \boldsymbol{\ell}_D\big). \tag{8}$$

We then aggregate the query-specific score changes over the query set $\mathcal{Q}_D$ to quantify the overall marginal retrieval utility of embedding j:

$$U_D(j \mid \boldsymbol{\ell}_D) = \frac{1}{|\mathcal{Q}_D|} \sum_{q \in \mathcal{Q}_D} \frac{\Delta S_q(j)}{N_q}. \tag{9}$$

Greedy Depth Allocation. A simple strategy for token-wise depth allocation is to rank possible code additions by the above marginal retrieval utility and allocate the budget to those with the highest utilities until B_D is exhausted. However, the utility of one embedding depends on the current depths of the other embeddings. Adding a code to one embedding can therefore affect the marginal benefits of refining the others. We therefore allocate the extra codes greedily, one document embedding at a time. Starting from $\ell_{D,j} = L_0$ for all j, we assign each code to the embedding with the largest current utility and recompute the utilities after each allocation under the updated depth map. The process stops when all B_D extra codes are assigned.

4 Experiments

We first describe the experimental setup and then present and discuss the results.

4.1 Experimental Setting

Datasets and Metrics. We evaluate on ViDoRe V1 (Faysse et al., 2025), which contains 10 subsets and is the standard benchmark of prior VDR compression work (Ma et al., 2025; Yan et al., 2026), and ViDoRe V2 (Macé et al., 2025), which contains 4 subsets and more challenging long-form and cross-document queries. For V2 we use the official English queries. Following prior work (Faysse et al., 2025; Macé et al., 2025; Ma et al., 2025; Yan et al., 2026), we report nDCG@5 per subset and its macro average.

Baselines. We adopt two groups of baselines. As quantization remains unexplored for VDR, we adopt classical quantizers as the first group, including Product Quantization (PQ) (Jégou et al., 2011), Optimized Product Quantization (OPQ) (Ge et al., 2013), and Residual Quantization (RQ) (Chen et al., 2010). They learn codebooks by reconstruction and assign every embedding the same precision. Meanwhile, as token reduction is a prevailing approach to compressing VDR indexes, we additionally compare against representative token reduction methods, including Light-ColPali (Ma et al., 2025) and Prune-then-Merge (Yan et al., 2026), which reduce the number of stored embeddings rather than their precision.

Implementation Details. We use two frozen retrievers, ColPali (PaliGemma-3B) (Faysse et al., 2025) and ColQwen2.5 (Qwen2.5-VL-3B) (Bai et al., 2025b), both producing 128-d embeddings. For a fair comparison, all quantizers are evaluated under the same document-side code budget. Since RetVDQ assigns variable-depth representations while PQ, OPQ, and RQ use a fixed depth per token, we define the budget as the average number of code indices stored per token, denoted by $\bar{\ell}$. Following standard multi-codebook quantization settings (Huijben et al., 2024; Vallaeys et al., 2025), we set the codebook size to $K = 256$, so that each code index can be stored in one byte. Thus $\bar{\ell}$ denotes a budget in bytes per token (B/token), and we evaluate all methods at $\bar{\ell} \in \{2, \ldots, 7\}$ B/token. This budget is met exactly by PQ, OPQ, and RQ, which assign every token the same number of codes—$\bar{\ell}$ subvector codes for PQ and OPQ, and a fixed residual depth of $\bar{\ell}$ for RQ. RetVDQ instead maintains an $L = 8$ residual hierarchy and distributes the same total number of codes non-uniformly across the M_D tokens of a page, satisfying $\sum_{j=1}^{M_D} \ell_{D,j} = \bar{\ell} M_D$, where $\ell_{D,j}$ is its per-token residual depth. Full details are in Appendix C.

4.2 Main Results

Comparison with Quantization Methods. Table 1 shows the performance of quantization methods across different code budgets on both benchmarks and both retrieval backbones. Table 2 and Table 3 further report per-subset results of ViDoRe V1 and ViDoRe V2 at 2 B/token.

Table 1: Macro-averaged nDCG@5 on ViDoRe V1 and V2 across code budgets (B/token).

	Method	ViDoRe V1 (B/token)						ViDoRe V2 (B/token)					
		2	3	4	5	6	7	2	3	4	5	6	7
ColPali	*Full precision*			81.3						53.3			
	PQ	52.0	64.5	68.4	72.2	75.4	74.9	37.7	46.3	49.4	50.9	50.9	53.1
	OPQ	51.7	63.1	68.4	71.2	73.1	73.9	35.4	41.9	42.4	46.1	49.0	46.2
	RQ	64.7	73.3	76.3	77.5	78.3	79.0	45.2	48.3	50.6	50.8	51.9	51.9
	RetVDQ	**74.1**	**77.5**	**79.1**	**80.0**	**79.8**	**80.0**	**51.0**	**52.0**	**54.2**	**54.1**	**54.3**	**54.5**
ColQwen2.5	*Full precision*			89.2						61.4			
	PQ	63.7	74.6	80.0	83.0	84.7	85.6	44.9	50.6	51.8	56.0	56.0	56.4
	OPQ	68.5	77.5	82.5	84.3	85.6	85.9	47.2	55.4	56.4	57.7	58.1	58.8
	RQ	73.2	82.5	85.4	86.6	87.3	87.9	50.3	58.5	60.5	60.9	61.2	61.7
	RetVDQ	**82.7**	**86.1**	**87.4**	**87.6**	**88.3**	**88.3**	**56.7**	**60.6**	**62.0**	**63.0**	**63.3**	**63.1**

Table 2: nDCG@5 on all ten ViDoRe V1 subsets at 2 B/token with ColPali.

Method	Arxiv	DocVQA	InfoVQA	TabFQuAD	TATDQA	Shift	AI	Energy	Gov.	Health.	Avg.
PQ	50.7	31.4	54.9	73.1	36.0	39.7	58.8	57.9	60.6	56.6	52.0
OPQ	53.9	27.5	50.7	65.5	39.0	29.4	55.8	63.6	64.7	66.7	51.7
RQ	64.5	40.2	68.3	74.0	51.4	49.8	73.3	70.3	78.4	76.8	64.7
RetVDQ	**70.9**	**46.6**	**76.3**	**83.4**	**60.8**	**65.9**	**85.1**	**81.7**	**83.5**	**86.8**	**74.1**

Table 3: nDCG@5 on ViDoRe V2 at 2 B/token with ColPali.

Method	Bio.	Econ.	ESG	ESG-HL	Avg.
PQ	36.7	37.9	39.8	36.3	37.7
OPQ	39.6	30.7	37.1	34.3	35.4
RQ	46.8	49.5	41.0	43.6	45.2
RetVDQ	**55.0**	**51.7**	**47.3**	**49.9**	**51.0**

We highlight the following observations. **(1) RetVDQ consistently outperforms quantization baselines.** RetVDQ achieves the best result at every budget on both retrievers and both benchmarks. In three of the four settings, it surpasses RQ at 7 B/token using only 4 B/token, saving over 40% of the code storage.

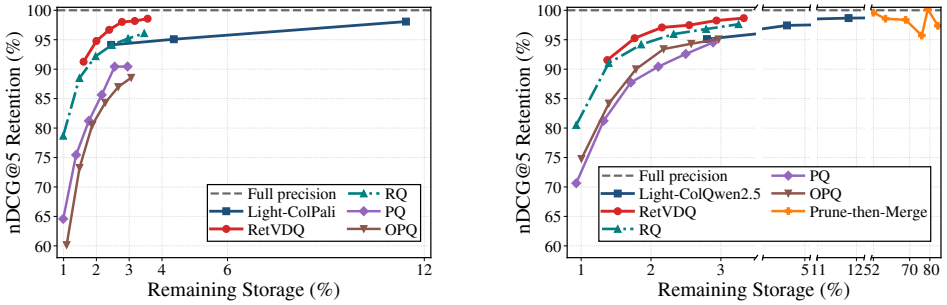
(2) The advantage is largest under tight budgets. The largest gain over RQ always appears at 2 B/token, reaching 9.4 and 9.5 points on V1 and 5.8 and 6.4 points on V2 for ColPali and ColQwen2.5, respectively. With few codes, reconstruction-oriented codebooks lose much of the information MaxSim relies on. **(3) The improvement is consistent across document types.** RetVDQ improves over RQ on all subsets in Tables 2 and 3, with the largest gains on the content-rich ShiftProject (16.1 points) and Artificial Intelligence (11.8 points), where relevant evidence spreads over many patches and allocation by retrieval utility is most beneficial. **(4) RetVDQ can exceed full precision on the harder ViDoRe V2.** On V1, where full precision is already strong, RetVDQ approaches it without surpassing it. On V2, where full precision is considerably weaker, RetVDQ exceeds the full-precision index at 4 B/token and above, by up to 1.2 and 1.9 points for ColPali and ColQwen2.5. We attribute this to RetVDQ's corpus-specific supervision: its codebooks and depth allocation are fitted to each target corpus with synthetic queries, mildly adapting to the corpus and correcting some ranking errors of the frozen retriever on this harder benchmark.

To translate these per-token code budgets into actual index compression, we aggregate RetVDQ over all 14 ViDoRe subsets (10 from V1 and 4 from V2) with ColPali. As shown in Table 4, at 4 B/token, RetVDQ retains 98.2% of full-precision nDCG@5 while using only 2.0% of the original document-index storage; at 7 B/token, retention reaches 99.3% with 3.2% storage. This shows that nearly all retrieval quality can be preserved within a small fraction of the full-precision index.

Comparison with Token Reduction Methods. Unlike RetVDQ, which keeps every embedding and encodes each with a few bytes, token reduction methods store fewer embeddings per page but keep each in full precision. As the two families operate at different storage ranges: token reduction typically retains a substantial fraction of the original index, whereas quantization can compress each embedding to only a few bytes, a comparison at a fixed budget is not meaningful. We thus compare

Table 4: nDCG@5 retention versus remaining storage on ViDoRe V1 and V2 with ColPali.

Code Budget(B/token)	**2**	**3**	**4**	**5**	**6**	**7**
nDCG@5 Retention(%)	92.0	96.0	98.2	99.0	99.2	99.3
Remaining Storage(%)	1.3	1.6	2.0	2.4	2.8	3.2



(a) ColPali backbone.

(b) ColQwen2.5 backbone.

Figure 3: nDCG@5 retention versus remaining storage on ViDoRe V1, averaged over the same six subsets reported by Light-ColPali. Results are taken from prior work where available.

their storage–quality tradeoff in Fig. 3, plotting nDCG@5 retention (compressed over full-precision nDCG@5) against remaining storage (compressed over full-precision storage). Specifically, we compare with Light-ColPali, Light-ColQwen2.5 (Ma et al., 2025), and Prune-then-Merge (Yan et al., 2026), using their published results. For consistent evaluation, we report results over the six ViDoRe V1 subsets (DocVQA, InfoVQA, TabFQuAD, ArxivQA, ShiftProject, and Healthcare) shared with Light-ColPali. As can be seen, on both backbones, RetVDQ reaches similar or higher retention with several times less storage, while Prune-then-Merge needs more than half of the original index. This suggests that adapting the precision of each embedding preserves MaxSim evidence more efficiently than removing embeddings.

4.3 Ablation Studies

We ablate the two key components of RetVDQ on three ViDoRe V1 subsets, TabFQuAD, TAT-DQA, and InfoVQA, which cover tables, financial reports, and infographics. Table 5 shows the macro-averaged ablation results across the three subsets. We group the variants into three blocks, corresponding to codebook learning, depth allocation, and their combination. Our allocation is denoted as utility-guided since it follows the marginal retrieval utility of Eq. (9). In the first block, (b) to (d) add level-wise supervision, joint optimization, or both to standard RQ (a) under uniform allocation, testing how each stage improves the codebooks. In the second block, random (e) and reconstruction-guided (f) allocation on the full codebooks test whether variable depth without any criterion, or with a reconstruction-based criterion, suffices to decide where precision matters for retrieval. In the last block, (g) applies our allocation to standard RQ, testing whether codebook learning still matters once utility-guided allocation is used.

We draw the following observations from Table 5. **(1) Retrieval-aware codebook learning improves retrieval fidelity.** Under uniform allocation, it improves over standard RQ by 3.2, 1.6, and 1.4 points at 2, 4, and 6 B/token (a vs. d). Level-wise supervision outperforms joint optimization alone at 2 B/token (b vs. c), and combining the two performs best, confirming the benefit of learning the codebooks progressively before joint optimization. **(2) Allocation should follow retrieval utility.** Random and reconstruction-guided allocation improve over uniform allocation only at 2 B/token, and fall below it at 4 and 6 B/token (d vs. e, f). Without a retrieval signal, they move codes away from the embeddings that MaxSim relies on. In contrast, utility-guided allocation improves over uniform allocation at every budget, by up to 5.7 points (d vs. Ours), showing that neither variable depth alone nor a reconstruction-based criterion can identify which embeddings matter for retrieval. **(3) The two components are complementary.** Utility-guided allocation improves standard RQ by 6.3, 1.7, and 1.2 points (a vs. g), and retrieval-aware codebooks further add 2.6, 1.2,

Table 5: Ablation study w.r.t. nDCG@5 under matched code budgets. Row (a) is standard RQ.

Variant	Codebook learning		Depth allocation	B/token		
	Progressive	Joint		2	4	6
(a)	✗	✗	Uniform	64.6	73.0	74.9
(b)	✓	✗	Uniform	67.3	74.6	75.6
(c)	✗	✓	Uniform	66.8	74.6	75.6
(d)	✓	✓	Uniform	67.8	74.6	76.3
(e)	✓	✓	Random	70.2	73.7	75.5
(f)	✓	✓	Reconstruction-guided	70.6	74.3	75.8
(g)	✗	✗	Utility-guided	70.9	74.7	76.1
Ours	✓	✓	Utility-guided	**73.5**	**75.9**	**76.8**

and 0.7 points on top of it (g vs. Ours), so each component remains effective when the other is present. Overall, these results validate the effectiveness of both retrieval-aware codebook learning and retrieval-aware depth allocation, as well as the benefit of combining them.

4.4 Further Analysis of Prefix-Level Retrieval Fidelity

Progressive codebook learning aims to preserve retrieval behavior at each prefix that depth allocation may retain, rather than only at the full depth. To verify this, we measure how far the score margin of each prefix deviates from that of the full-precision retriever. For each query q, we fix a positive page and its top-ranked non-relevant page under S_{FP} as the negative, and measure the normalized margin error as $\mathcal{E}^{(l)} = \frac{1}{|\mathcal{Q}|}\sum_{q\in\mathcal{Q}} |M_q^{(l)} - M_{\mathrm{FP},q}|/N_q$, with margins defined as in Eq. (4). Lower values indicate that the depth-l prefix better preserves the full-precision ranking. Fig. 4 compares RetVDQ with standard RQ across residual prefix depths $l \in \{1,\dots,8\}$, averaged over TabFQuAD, TAT-DQA, and InfoVQA. RetVDQ achieves lower normalized margin error at every depth, showing that retrieval-aware learning improves all retained prefixes, not just the full-length representation. Specifically, the reduction relative to standard RQ grows from 3.3% at depth 1 to 13.7% at depth 8. This is reasonable as each residual level corrects part of the distortion accumulated by the preceding ones.

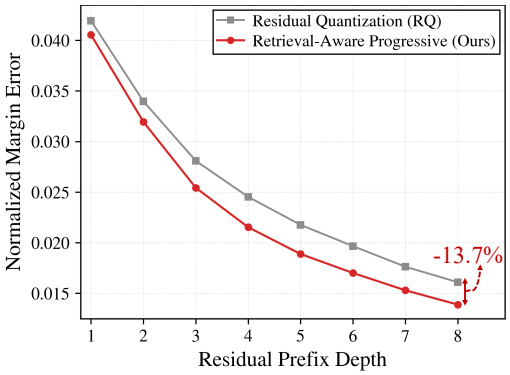


Figure 4: Normalized margin error w.r.t. the full-precision retriever across prefix depths.

5 Conclusion

We presented RetVDQ, a retrieval-aware variable-depth residual quantization framework for multi-vector visual document retrieval. Rather than removing embeddings, RetVDQ keeps every embedding and adapts its precision under a fixed budget, by learning residual prefixes that preserve the full-precision ranking and allocating depth by marginal MaxSim utility. Experiments on ViDoRe show that RetVDQ outperforms quantization baselines at every budget and achieves a better trade-off between retrieval quality and storage than token reduction.

Limitations RetVDQ relies on offline synthetic queries and corpus-specific codebooks, so its effectiveness depends on query quality, and substantial corpus shift may require retraining. Future work includes extending the depth range to zero, so that utility-guided allocation can also decide which embeddings to drop, and extending RetVDQ to jointly trained retrievers.

AI USE STATEMENT

We used generative AI tools for manuscript editing, code debugging, and assisting the implementation of data processing pipelines. For synthetic query construction, we used a vision-language model as described in Section 3.1; all generated queries were filtered and used only as offline retrieval probes for index construction. All AI-assisted outputs were reviewed and verified by the authors, and the authors take full responsibility for the final content, experiments, and conclusions of this work.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we provide detailed descriptions of the task formulation, compression framework, training objectives, and experimental protocols in the main paper and supplementary material. The appendix includes implementation details, synthetic query construction procedures, baseline configurations, and storage accounting. We will release the training and evaluation code, together with the necessary configuration files, to reproduce the reported experiments.

REFERENCES

Srikar Appalaraju, Bhavan Jasani, Bhargava Urala Kota, Yusheng Xie, and R. Manmatha. Docformer: End-to-end transformer for document understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 993–1003, 2021.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, et al. Qwen3-VL technical report, 2025a.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-VL technical report, 2025b.

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

Yan Fang, Jingtao Zhan, Yiqun Liu, Jiaxin Mao, Min Zhang, and Shaoping Ma. Joint optimization of multi-vector representation with product quantization. In *Natural Language Processing and Chinese Computing*, volume 13551 of *Lecture Notes in Computer Science*, pp. 631–642. Springer, 2022. NLPCC 2022.

Manuel Faysse, Hugues Sibille, Tony Wu, Bilel Omrani, Gautier Viaud, Céline Hudelot, and Pierre Colombo. ColPali: Efficient document retrieval with vision language models. In *International Conference on Learning Representations*, 2025.

Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2946–2953. IEEE Computer Society, 2013.

Iris A. M. Huijben, Matthijs Douze, Matthew J. Muckley, Ruud Van Sloun, and Jakob Verbeek. Residual quantization with implicit neural codebooks. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 20682–20699. PMLR, 2024.

Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

Yubo Ma, Jinsong Li, Yuhang Zang, Xiaobao Wu, Xiaoyi Dong, Pan Zhang, Yuhang Cao, Haodong Duan, Jiaqi Wang, Yixin Cao, and Aixin Sun. Towards storage-efficient visual document retrieval: An empirical study on reducing patch-level embeddings. In *Findings of the Association for Computational Linguistics: ACL*, pp. 19568–19580. Association for Computational Linguistics, 2025.

Quentin Macé, António Loison, and Manuel Faysse. ViDoRe benchmark v2: Raising the bar for visual retrieval, 2025.

Rohit Kumar Salla, Manoj Saravanan, and Ramya Manasa Amancherla. CrossQ: Task-aligned cross-token conditional quantization for late interaction retrieval. In *International Conference on Machine Learning*, 2026.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. ColBERTv2: Effective and efficient retrieval via lightweight late interaction. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3715–3734. Association for Computational Linguistics, 2022.

Théophane Vallaeys, Matthew J. Muckley, Jakob Verbeek, and Matthijs Douze. QINCo2: Vector compression and search with improved implicit neural codebooks. In *International Conference on Learning Representations*, 2025.

Shitao Xiao, Zheng Liu, Weihao Han, Jianjin Zhang, Defu Lian, Yeyun Gong, Qi Chen, Fan Yang, Hao Sun, Yingxia Shao, and Xing Xie. Distill-VQ: Learning retrieval oriented vector quantization by distilling knowledge from dense embeddings. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1513–1523. Association for Computing Machinery, 2022.

Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, and Ming Zhou. Layoutlm: Pre-training of text and layout for document image understanding. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1192–1200. Association for Computing Machinery, 2020.

Yibo Yan, Mingdong Ou, Yi Cao, Xin Zou, Jiahao Huo, Shuliang Liu, James Kwok, and Xuming Hu. Sculpting the vector space: Towards efficient multi-vector visual document retrieval via prune-then-merge framework. In *Findings of the Association for Computational Linguistics: ACL*, pp. 24883–24925. Association for Computational Linguistics, 2026.

Eugene Yang, Andrew Yates, Dawn Lawrie, James Mayfield, Saron Samuel, and Rohan Jha. ColBERTSaR: Sparsified ColBERT index via product quantization. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 4305–4310. Association for Computing Machinery, 2026.

Yingrui Yang, Yifan Qiao, and Tao Yang. Compact token representations with contextual quantization for efficient document re-ranking. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 695–707. Association for Computational Linguistics, 2022.

ADDITIONAL MATERIALS

This appendix provides additional details and analyses that complement the main paper. We first discuss related work to further position RetVDQ with respect to existing VDR compression and quantization approaches. We then provide additional methodological and implementation details, including index construction, retrieval procedures, synthetic query supervision, and training configurations. Finally, we report extended experimental analyses, including storage accounting, offline index construction cost, additional benchmark comparisons, and results.

A RELATED WORK

A.1 VISUAL DOCUMENT RETRIEVAL

Vision-based document retrieval has recently emerged as an alternative to OCR-centric document retrieval pipelines. ColPali introduces a vision-language retriever that represents each document page with multiple contextualized embeddings and performs late interaction with query-token representations (Faysse et al., 2025). ColQwen2 and subsequent ColVision models retain this multi-vector retrieval paradigm while adopting stronger vision-language backbones (Faysse et al., 2025). Together, these models establish multi-vector late interaction as a strong architecture for visual document retrieval. Our work adopts this retrieval setting and focuses on compressing its document-side index.

A.2 EFFICIENT VISUAL DOCUMENT RETRIEVAL

Recent work improves the storage efficiency of multi-vector VDR mainly by reducing or aggregating document embeddings. Light-ColPali studies patch pruning and merging for ColPali-style retrievers (Ma et al., 2025). Prune-then-Merge further explores adaptive pruning, merging, (Yan et al., 2026). These approaches primarily target the number or organization of document representations retained in the index. Our work studies a complementary direction by compressing representation precision while retaining the document-embedding positions used by late interaction.

A.3 RETRIEVAL-AWARE VECTOR QUANTIZATION

Vector quantization has long been used to reduce embedding storage. Product Quantization and Optimized Product Quantization compress vectors through subspace decomposition (Jégou et al., 2011; Ge et al., 2013), while Residual Quantization represents a vector through successive quantization of its residuals (Chen et al., 2010). Some learned multi-codebook approaches further improve compression quality.

Retrieval-aware quantization incorporates ranking objectives into compression. Contextual Quantization learns compact token representations for document reranking (Yang et al., 2022), and CrossQ develops conditional quantization for late interaction, with an additional variant that fine-tunes part of the retriever (Salla et al., 2026). JMPQ jointly optimizes the query encoder and product quantization for multi-vector retrieval (Fang et al., 2022). These approaches establish both fixed-retriever compression and encoder adaptation as viable settings. Our work focuses on the former and studies a distinct design choice: learning deployable residual prefixes and allocating their retained depths across document embeddings under a storage budget.

Compression can also be designed for existing retrieval representations. ColBERTv2 applies residual compression without quantization-specific changes to encoder training (Santhanam et al., 2022), while ColBERTSaR constructs a compact approximation of MaxSim without retraining the retrieval model (Yang et al., 2026). Distill-VQ fixes document embeddings while learning the query encoder and quantizer, emphasizing compatibility with existing embeddings and the computational savings from excluding document encoding from training (Xiao et al., 2022).

B Additional Method Details

B.1 Index Construction and Retrieval

The final index stores the shared codebooks and the retained codes of each document embedding. The query sets and all intermediate results are discarded after compression. Since each embedding stores only a retained prefix of its residual codes, its depth is implicitly determined by the number of stored code indices, without requiring an additional depth map. The storage B_{index} in Eq. (2) is therefore

$$B_{\text{index}} = L \times K \times d \times b_{\text{fp}} + \sum_{D \in \mathcal{D}} \left(\lceil \log_2 K \rceil \sum_{j=1}^{M_D} \ell_{D,j} \right), \tag{10}$$

where the first term counts the shared codebooks, which are stored once for the entire corpus, and the second term counts the retained residual code indices of all document embeddings.

At retrieval time, the query is encoded by the frozen query encoder, and each document embedding is decoded from its retained codes as

$$\widehat{\mathbf{x}}_{D,j} = \sum_{l=1}^{\ell_{D,j}} \mathbf{C}_l[z_{D,j,l}], \tag{11}$$

which is used without renormalization. Pages are then ranked by Eq. (1) with $\mathbf{x}_{D,j}$ replaced by $\widehat{\mathbf{x}}_{D,j}$. Online retrieval therefore only requires decoding the retained residual codes before standard MaxSim scoring and does not modify the original retriever.

B.2 Synthetic Query Construction

The compression process requires query-dependent signals because both codebook learning and residual-depth allocation optimize retrieval behavior under MaxSim. However, official evaluation queries and relevance annotations are unavailable when the document index is constructed. We therefore build an offline page-grounded query set $\mathcal{Q}_D$ for each document page D using only information available before deployment.

The generated queries are used exclusively as retrieval probes during offline compression. They are not used for final evaluation, and all official benchmark queries and relevance labels remain unseen until the compressed index has been finalized. This separation ensures that the reported retrieval results measure generalization to unseen queries rather than memorization of benchmark queries.

B.2.1 Query Generation Pipeline

For each document page, we generate synthetic queries through a page-grounded visual question generation process. The pipeline consists of three steps:

1. **Page understanding.** A vision-language model receives the document page image and summarizes the key information contained in the page, including textual content, tables, figures, and layout structures.
2. **Query generation.** Conditioned on the page content, the model generates user-like retrieval queries whose answers can be found from the given page. The generated queries are designed to reflect realistic information needs rather than directly copying page text.
3. **Filtering and validation.** Generated queries are filtered according to length and quality constraints. The remaining queries form the offline probe set $\mathcal{Q}_D$ used for retrieval-aware codebook learning and depth allocation.

For all experiments, synthetic queries are generated before index construction. The same query-generation procedure is applied across different backbones, while each backbone independently learns its compressed index in its own embedding space.

B.2.2 Prompt Template for Synthetic Query Generation

To obtain retrieval supervision for document-side compression, we use the frozen Qwen3-VL-8B-Instruct model (Bai et al., 2025a) (`Qwen/Qwen3-VL-8B-Instruct`) to generate page-grounded query candidates, from which $N_{\text{syn}} = 8$ queries per page are retained. No additional system prompt is used. For each page, the model receives only the page image and a dataset-level corpus descriptor in a single user message. The descriptor is constructed from public dataset documentation and corpus metadata, without access to evaluation queries or relevance judgments.

The prompt encourages the model to generate realistic search intents while avoiding answer leakage, trivial surface matching, and references to the observed page. The generated candidates contain structured fields including a broad intent, a retrieval query, an answer, and a query type. Only the `specific_query` field is used as retrieval supervision.

The core prompt template is shown below.

Prompt Template

```
You are constructing candidate search queries for offline training
of a visual document retriever and a retrieval-aware document
compression model.
Given the single page image, generate exactly N distinct candidate
records.  Simulate users searching a large document corpus without
knowing the page content.
Generate standalone and natural queries corresponding to realistic
information needs.  Cover diverse answer-bearing evidence, including
entities, values, dates, definitions, categories, conditions,
comparisons, trends, tables, figures, and conclusions when
available.
Do not refer to the page, document, image, table, figure, or content
shown.  Do not reveal the answer in the query.  Do not copy complete
sentences from the source.  Avoid generic requests such as title,
summary, main topic, or overall findings.
For each candidate, output a JSON record:
{
  "broad_intent": "...",
  "specific_query": "...",
  "answer": "...",
  "query_type": "..."
}
Return JSON only.
```

For all experiments, we sample with `temperature`=0.7, `top_p`=0.8, `top_k`=20, and `seed`=3407. Each page initially produces 24 candidates. After rule-based filtering, vision-language validation, and full-precision retrieval-consistency checking, we retain 8 queries per page as final retrieval probes. The official evaluation queries and relevance judgments are never exposed during query generation or filtering.

C Experimental Details

C.1 Benchmarks and metric.

ViDoRe V1 contains ten subsets covering heterogeneous layouts and provides a common basis for comparison with existing compression methods (Ma et al., 2025; Yan et al., 2026). ViDoRe V2 complements V1 with more demanding long-form and cross-document queries. We use the English queries of V2 to isolate document-side compression from multilingual query transfer. nDCG@5 measures both whether relevant pages enter the top five and how highly they are ranked, and naturally supports the multi-positive relevance labels of V2.

C.2 EMBEDDING EXTRACTION.

Under our preprocessing, ColPali produces 1,030 document positions per page, while ColQwen2.5 uses dynamic resolution and yields a page-dependent number of positions. Document embeddings are encoded once and cached, so that all methods operate on identical full-precision representations.

C.3 BASELINE CONFIGURATIONS.

All quantizers use $K = 256$ codewords per codebook, so each code index occupies one byte. Under a budget of $\bar{\ell}$ B/token, PQ and OPQ split each 128-dimensional embedding into $\bar{\ell}$ subspaces and quantize each with its own codebook, RQ encodes every embedding with the first $\bar{\ell}$ residual levels, and RetVDQ retains an average depth of $\bar{\ell}$ per page.

C.4 QUANTIZER TRAINING.

For each dataset and retriever, all quantizers are trained on the same set of document embeddings drawn with page-balanced sampling. Table 6 lists the hyperparameters used for RetVDQ. Unless otherwise specified, ColPali and ColQwen2.5 use the same optimization configuration. For ColQwen2.5, synthetic queries are filtered in the ColQwen2.5 embedding space so that retrieval supervision matches the target backbone. After codebook training, the learned codebooks are frozen before residual-depth allocation. All experiments are run on NVIDIA L40S GPUs.

Table 6: Hyperparameters of RetVDQ.

Hyperparameter	**Value**
Residual levels L	8
Codewords per level K	256
Base depth L_0	1
Synthetic queries per page N_{syn}	8
Query filtering rank K_f	100
Prefix-hard negatives H_0	8 per query per prefix
Ranking-based negative pool κ	5
Reconstruction weight μ	0.1 (progressive), 0.01 (joint)
Optimizer and learning rate	AdamW, lr $= 3 \times 10^{-4}$, weight decay $= 0$
Training epochs (progressive / joint)	5 query exposures per level / 5 epochs
Batch size	8 queries
Reconstruction replay tokens	4,096 document tokens per update
Gradient clipping	1.0
Random seed	13

C.5 STORAGE ACCOUNTING

This section details the storage accounting used for the storage–quality comparison in Fig. 3, including the conversion from the nominal B/token budgets defined in Sec. 4.1 to remaining-storage percentages. We account only for the document-side retrieval index.

Since both retrievers produce 128-dimensional document embeddings, the full-precision index is stored in FP16 and therefore requires $128 \times 2 = 256$ bytes per document embedding. Query embeddings are generated online during retrieval and are not included in document-index storage.

For quantized indexes, we include both the stored document-side code indices and the shared codebooks. For RetVDQ, the total index storage is computed according to Eq. (10). The B/token values reported in the main paper denote the nominal code-index budget, i.e., the average number of stored one-byte code indices per document embedding. When converting this nominal budget to actual index storage, we additionally include the shared codebook storage.

We report the remaining storage relative to the corresponding full-precision document index:

$$\text{Remaining Storage } (\%) = \frac{B_{\text{index}}^{\text{compressed}}}{B_{\text{index}}^{\text{FP16}}} \times 100. \tag{12}$$

Table 7: Offline index construction cost on a representative ViDoRe V1 workload. Times are measured from cached document embeddings and exclude document encoding by the retriever.

Component	Time
Synthetic query generation	1.84 h
RQ codebook learning	1.76 min
RQ encoding	0.03 min
RetVDQ progressive learning	123.54 min
RetVDQ joint optimization	7.88 min
RetVDQ depth allocation	1.84 min
RetVDQ encoding	0.03 min

C.6 Offline Index Construction Cost

RetVDQ introduces additional offline optimization stages beyond standard residual quantization, including retrieval-aware codebook learning and residual-depth allocation. We therefore analyze the offline index construction cost to quantify the computational overhead introduced by these components.

We measure the construction time on a representative ViDoRe V1 workload using 500 document pages from InfoVQA. Each page contains 1,030 document positions encoded by the official ColPali embedding space. All measurements start from cached document embeddings and exclude the cost of the ColPali forward pass. The RQ baseline uses $K = 256$ codewords per residual level with $L = 8$ residual levels, trained with 500K document tokens. Results are summarized in Table 7.

The major additional cost of RetVDQ comes from retrieval-aware codebook optimization. Specifically, progressive learning optimizes residual levels sequentially to preserve retrieval behavior at different prefixes, while joint optimization further calibrates the complete residual hierarchy. In contrast, the depth allocation stage introduces only a small overhead (less than two minutes for 500 pages), and the final encoding cost is comparable to standard RQ because both methods store residual code indices with the same representation format.

The synthetic query construction stage is reported separately from compression cost because it provides offline retrieval supervision rather than directly compressing the index. During deployment, the generated queries and intermediate optimization artifacts are discarded after index construction. Therefore, online retrieval remains unchanged and only requires decoding the retained residual codes followed by the original MaxSim scoring procedure.

Although RetVDQ requires additional offline computation compared with reconstruction-oriented RQ, this cost is incurred only once during index construction. This trade-off allows RetVDQ to optimize residual depth according to retrieval utility while keeping the original frozen late-interaction retrieval pipeline unchanged.