

PUMA: Post-Hoc Sparsification of Universal Multimodal Embeddings for Efficient Retrieval

Matteo Attimonelli^{1,2*}, Alessandro De Bellis^{1†}, Franco Maria Nardini³, Claudio Pomo¹
Cosimo Rulli³, Rossano Venturini⁴, Tommaso Di Noia¹

¹Politecnico di Bari, Italy ²Sapienza University of Rome, Italy

³ISTI-CNR, Pisa, Italy ⁴University of Pisa, Italy

{name.surname}@{poliba.it, isti.cnr.it, unipi.it}

Abstract

Universal multimodal embedders enable retrieval across text, image, and combined queries, but their dense representations incur high memory and inference costs. Post-hoc sparsification could reduce these costs but remains underexplored for multimodal retrieval. We introduce **PUMA**, a sparse autoencoder recipe that maps universal multimodal embeddings to compact sparse codes without retraining the backbone: a pretraining stage preserves dense dot-product geometry, after which the sparse encoder is fine-tuned for retrieval. We evaluate on five benchmarks covering text-to-image and composed image retrieval. On Qwen3-VL-Embedding-2B, PUMA is statistically indistinguishable from or improves over dense retrieval on four of five datasets. We further identify two failure modes of post-hoc sparsification: insufficient pre-TopK support and retrieval-misaligned active support. PUMA reduces vector storage by 8–16× (FP32) and is up to 25× faster than exact dense scoring on larger candidate pools, enabling efficient multimodal retrieval. Code is provided at: <https://github.com/sisinflab/PUMA>.

1 Introduction

Universal multimodal embedders offer a practical unified interface for retrieval across text, image, and mixed-modality queries (Lin et al., 2025; Zhang et al., 2025). A single backbone maps queries and candidates of any modality into one shared space, without separate retrieval heads or fusion modules. This generality comes at a cost: embeddings are dense, high-dimensional, and expensive to store and score over large candidate sets. Exact dense scoring over n candidates with d -dimensional embeddings costs $O(nd)$ per query,

and approximate dense retrieval still requires specialized vector indexes over large floating-point representations. Sparse retrieval offers a complementary alternative, storing vectors as small sets of components and scoring through feature overlap.

Prior works studied sparse text retrieval (Formal et al., 2021b; Wen et al., 2025; Formal et al., 2026; Guo et al., 2026; Nguyen et al., 2026), sparsity during multimodal contrastive pretraining (Qin et al., 2026), and post-hoc multimodal sparse autoencoders for representation analysis (Kaushik et al., 2026). What remains underexplored is post-hoc sparsification for retrieval serving: can a frozen universal multimodal embedder (i.e., a model that produces general purpose embeddings) be converted into a sparse retriever without retraining the backbone, while preserving quality across diverse tasks?

We propose **PUMA** (Post-hoc Universal Multimodal sparsifiCation), a post-hoc sparsification pipeline for frozen universal multimodal embedders. PUMA converts cached dense embeddings into sparse retrieval codes using a TopK sparse autoencoder. PUMA is first trained to preserve the dense dot-product geometry and is then fine-tuned with the sparse dot-product used at retrieval time. The resulting representation has exactly k active features per vector and can be scored as a sparse dot product. To the best of our knowledge, PUMA is the first work to study post-hoc sparsification of frozen universal multimodal embedders for retrieval serving.

We evaluate PUMA on five M-BEIR datasets (Wei et al., 2024), with candidate pools ranging from roughly 5K to 542K images. The tasks span two regimes: *text-to-image retrieval*, where text queries retrieve image candidates, and *composed image retrieval*, where a reference image and textual modification are fused into a single query representation. The latter is especially challenging as sparsification must preserve the modification semantics of this fused image–text query while

* Corresponding author: matteo.attimonelli@poliba.it.

† Work done while at Politecnico di Bari; currently Independent Researcher.

aligning it with target images.

On Qwen3-VL-Embedding-2B (Li et al., 2026), PUMA significantly improves over dense retrieval on CIRR and Fashion200K, is statistically indistinguishable on FashionIQ and MSCOCO, and trails dense on VisualNews. On CIRR, PUMA reaches 0.533 Hit@5 and 0.436 nDCG@10, compared with 0.522 and 0.424 for dense retrieval.

These gains are not explained by dimensionality reduction or domain-specific finetuning alone. In fact, PUMA outperforms dense TopK pruning, matched-memory PCA, a trained dense autoencoder, and encoder-only sparse training. Compared to a dense retriever, PUMA uses $8\text{--}16\times$ less per-vector storage and makes exact sparse search up to $25\times$ faster on larger candidate pools.

Our analysis identifies two ways in which post-hoc sparsification can fail. In some cases, the sparse encoder produces insufficient active features before TopK selection, so increasing k cannot help. In other cases, enough features are active, but they do not preserve the original dense ranking behavior. These failure modes suggest that future sparse multimodal retrievers must control both feature availability and retrieval alignment.

We make the following contributions:

1. We formulate post-hoc sparsification of frozen universal multimodal embedders as a retrieval problem, motivated by the storage and scoring costs of dense multimodal representations.
2. We propose **PUMA**, a retrieval-aware TopK sparse-autoencoder recipe that converts dense multimodal embeddings into compact sparse codes without retraining the backbone.
3. We evaluate PUMA on five datasets spanning text-to-image and composed image retrieval, showing competitive performance with dense baselines at substantially lower storage cost.
4. We compare against dense TopK pruning, matched-memory PCA, a dense autoencoder, and encoder-only sparse training, and analyze search time, statistical reliability, scaling, transfer, and k -selection failure modes.

Our results show that post-hoc sparsification offers a practical path to efficient multimodal retrieval, preserving the generality of universal embedders at a fraction of the storage and search cost.

2 Related Work

Universal multimodal retrieval. Universal multimodal retrieval aims to use a single embedding model across heterogeneous query–candidate pairs, including cross-modal and mixed-modality queries. M-BEIR (Wei et al., 2024) provides a unified benchmark across tasks and modalities, and MMEB (Jiang et al., 2025) broadens evaluation to new instruction-conditioned tasks. Recent dense embedders such as VLM2Vec (Jiang et al., 2025), MM-Embed (Lin et al., 2025), GME (Zhang et al., 2025), Qwen3-VL-Embedding (Li et al., 2026), and RZenEmbed (Jian et al., 2025) adapt multimodal LLMs or vision-language models into general-purpose embedding backbones; SERVAL (Nguyen et al., 2025) explores a complementary generate-and-encode route for visual document retrieval. PUMA is orthogonal to these works: rather than training a new model, we study whether pretrained dense representations can be sparsified post-hoc. We focus on universal multimodal embedders as they produce shared representations for mixed-modality inputs without modality-specific encoders or fusion modules.

Sparse retrieval and embedding compression.

Dense retrieval efficiency can be improved through quantization (Jégou et al., 2011), approximate search (Guo et al., 2020), or dense dimensionality reduction (Attimonelli et al., 2025), but these approaches keep dense representations or change the search procedure. Sparse retrieval instead changes the representation structure, storing feature–weight pairs that can reuse lexical-search infrastructure. This paradigm has been studied extensively for text, both in lexical spaces (Formal et al., 2021b,a) and in latent sparse codes, including CSR (Wen et al., 2025; Guo et al., 2026), MILCO (Nguyen et al., 2026), and SPLARE (Formal et al., 2026). Sparsification has also been explored in recommendation, including sparse contrastive supervision and sparse entity embeddings (Meehan and Pauwels, 2026; Vancura et al., 2026). PUMA builds on k -sparse autoencoders (Makhzani and Frey, 2014) and recent SAE work in representation analysis (Huben et al., 2024; Bricken et al., 2023; Gao et al., 2025). The closest multimodal comparisons differ in goal and intervention point: Sparse CLIP (Qin et al., 2026) optimizes sparsity during contrastive pretraining, whereas PUMA sparsifies a pretrained embedder post-hoc; MGS AE (Kaushik et al., 2026) studies post-hoc SAEs for interpretability, whereas PUMA

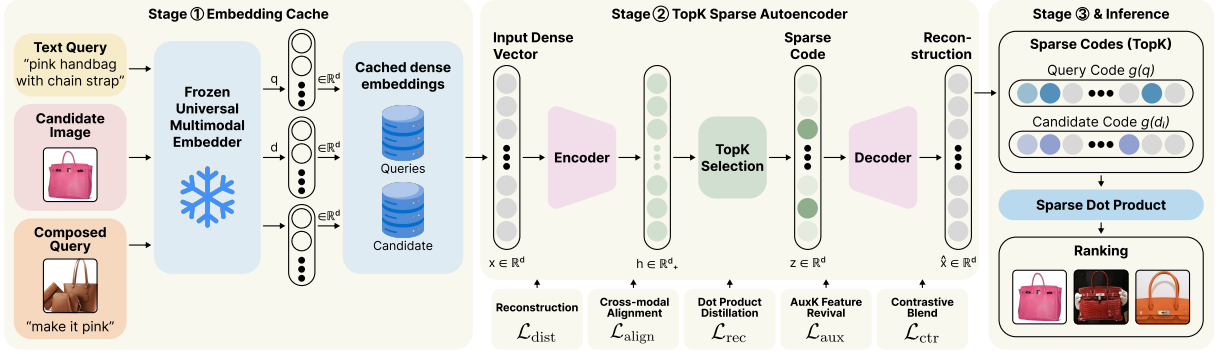


Figure 1: PUMA: (i) embeddings are cached using frozen embedders; (ii) a top- K SAE is trained to produce sparse representations; (iii) these representations are further tuned to optimize retrieval, and ranking is performed.

targets retrieval. We formulate post-hoc sparsification for frozen universal multimodal embedders, preserving dense retrieval geometry and evaluating retrieval quality, storage, and efficiency across text-to-image and composed image retrieval.

3 PUMA

Fig. 1 gives an overview of **PUMA**, a post-hoc sparsification approach for frozen multimodal embedders. PUMA converts dense embeddings from a frozen multimodal backbone into sparse retrieval codes. It learns an overcomplete dictionary of D latent features, with $D \gg d$: a TopK encoder maps a dense embedding $\mathbf{x} \in \mathbb{R}^d$ to a sparse code $\mathbf{z} \in \mathbb{R}^D$ with exactly k nonzero entries ($\|\mathbf{z}\|_0 = k$), and a linear decoder reconstructs $\hat{\mathbf{x}}$. Training has three stages. Stage ① caches dense embeddings from the frozen multimodal encoder. Stage ② pretrains the sparse autoencoder with reconstruction, retrieval-aware distillation, cross-modal alignment, feature revival for inactive dictionary atoms, and progressive k -annealing. Stage ③ fine-tunes the encoder with a contrastive loss on sparse codes, using the same sparse dot product used at retrieval time.

3.1 Problem Setting

Let $f(\cdot)$ be a frozen multimodal embedder that maps queries and candidates into a shared dense space. A query may be text only, image only, or composed from a reference image and a textual modification. We write $\mathbf{q} = f(\text{query})$ and $\mathbf{d} = f(\text{candidate})$, and dense retrieval ranks candidates by $\mathbf{q}^\top \mathbf{d}$. We learn a sparse encoder $g(\cdot)$ that maps any dense embedding to

$$\mathbf{z} = g(\mathbf{x}) \in \mathbb{R}^D, \quad \|\mathbf{z}\|_0 = k, \quad (1)$$

where D is the size of the learned sparse dictionary, $D \gg d$, and $k \ll D$ is the number of active

dictionary atoms per vector. Sparse retrieval ranks candidates by

$$s_{\text{sparse}}(\mathbf{q}, \mathbf{d}) = g(\mathbf{q})^\top g(\mathbf{d}). \quad (2)$$

The goal is to preserve dense retrieval rankings under a compact sparse representation.

3.2 TopK Sparse Autoencoder (SAE)

The sparse autoencoder computes, for each query and candidate embedding:

$$\mathbf{h} = \text{ReLU}(W_{\text{enc}}\mathbf{x} + \mathbf{b}_{\text{enc}}), \quad (3)$$

$$\mathbf{z} = g(\mathbf{x}) = \text{TopK}(\mathbf{h}, k), \quad (4)$$

$$\hat{\mathbf{x}} = W_{\text{dec}}\mathbf{z} + \mathbf{b}_{\text{dec}}. \quad (5)$$

Decoder (W_{dec}) columns are renormalized to unit norm after each optimization step to prevent scale drift. For our experiments, we set $D=16d$.

3.3 Stage ①: Embedding Cache

We run the frozen backbone f on the train, validation, and candidate-pool partitions of each task and store the resulting dense embeddings. All subsequent sparse training reads from this cache. This makes the training cost independent of the backbone and ensures that dense and sparse variants use identical underlying embeddings.

3.4 Stage ②: Retrieval-Aware Sparse Pretraining

A reconstruction objective alone is not directly aligned with ranking. Moreover, post-hoc sparse autoencoders over multimodal embeddings can learn modality-specific dictionaries, where different atoms fire for images and text, weakening cross-modal scoring (Kaushik et al., 2026). Stage 2 therefore trains on cached in-domain query–target pairs

with the objective:

$$\mathcal{L}_2 = \mathcal{L}_{\text{rec}} + \lambda_a \mathcal{L}_{\text{align}} + \lambda_d \mathcal{L}_{\text{dist}} + \lambda_x \mathcal{L}_{\text{aux}} + \lambda_c \mathcal{L}_{\text{ctr}}. \quad (6)$$

Reconstruction. We reconstruct both query and candidate embeddings with cosine reconstruction:

$$\mathcal{L}_{\text{rec}} = 1 - \cos(\mathbf{x}, \hat{\mathbf{x}}). \quad (7)$$

Cross-modal alignment. To make dictionary usage more balanced across modalities, we encourage paired cross-modal examples to activate overlapping atoms instead of assigning image and text embeddings to disjoint subsets of the D -dimensional sparse space. For an image/text pair $(\mathbf{x}_{\text{img}}, \mathbf{x}_{\text{txt}})$ with sparse codes $\mathbf{z}_{\text{img}}$ and $\mathbf{z}_{\text{txt}}$, we use

$$\mathcal{L}_{\text{align}} = \|\mathbf{m}_{\text{img}} - \mathbf{m}_{\text{txt}}\|_1 + \frac{1}{2} \sum_{j \in \mathcal{A}} ||z_{\text{img},j}| - |z_{\text{txt},j}||. \quad (8)$$

where $\mathbf{m}_{\text{img}}$ and $\mathbf{m}_{\text{txt}}$ denote the TopK support masks, and $\mathcal{A} = \{j : z_{\text{img},j} \neq 0 \wedge z_{\text{txt},j} \neq 0\}$ is the co-active support. The first term encourages shared support; the second aligns magnitudes on atoms that fire for both modalities. The support masks come from the hard TopK forward pass; gradients for the support-alignment term are passed to the selected pre-TopK activations using the same straight-through treatment as the TopK operation (Kaushik et al., 2026). For retrieval pairs, we apply this alignment to cross-modal query–target pairs: in text-to-image retrieval, this aligns the text-query code with the positive image-candidate code; in composed image retrieval, it aligns the composed image–text query code with the positive target-image code.

Dot-product distillation. We distill the dense retriever’s scoring geometry into sparse space, matching dense and sparse scores for each query–candidate pair:

$$\mathcal{L}_{\text{dist}} = \left(g(\mathbf{q})^\top g(\mathbf{d}) - \mathbf{q}^\top \mathbf{d} \right)^2. \quad (9)$$

Auxiliary feature revival (AuxK). A TopK encoder gives gradient only to the k selected atoms, so the remaining $D - k$ tend to die. Following Gao et al. (2025); Guo et al. (2026), we revive them by asking the dead atoms to reconstruct the residual the main code missed. Let $\mathcal{G} \subset \{1, \dots, D\}$ be the set of atoms inactive for more than T steps, and form an auxiliary code on the dead support $\mathbf{z}_{\text{aux}} = \text{TopK}_{k_{\text{aux}}}(\text{softplus}(\mathbf{h}|_{\mathcal{G}}))$:

$$\mathcal{L}_{\text{aux}} = \|\mathbf{W}_{\text{dec}} \mathbf{z}_{\text{aux}} - \text{sg}[\mathbf{x} - \hat{\mathbf{x}}]\|_2^2, \quad (10)$$

where $\text{sg}[\cdot]$ is the stop-gradient operator and ensures that only dead atoms are updated by this term.

Stage ② contrastive blend. We include a small InfoNCE term on sparse codes during pretraining,

$$\mathcal{L}_{\text{ctr}} = -\log \frac{\exp(g(\mathbf{q}_i)^\top g(\mathbf{d}_i)/\tau)}{\sum_j \exp(g(\mathbf{q}_i)^\top g(\mathbf{d}_j)/\tau)}, \quad (11)$$

which begins aligning the sparse representation with ranking before retrieval fine-tuning.

Progressive k -annealing. We start from a larger support k_{init} and linearly anneal to the target support k_{final} . This gives the dictionary a denser training signal early and ends training at the sparsity used for retrieval (Guo et al., 2026).

3.5 Stage ③: Contrastive Retrieval Fine-Tuning

Stage ③ optimizes the sparse encoder for retrieval with the same dot product used at inference. We use InfoNCE with in-batch negatives and, when available, explicit hard negatives, while retaining reconstruction and distillation as regularizers:

$$\mathcal{L}_3 = \mathcal{L}_{\text{ctr}} + \alpha \mathcal{L}_{\text{rec}} + \beta \mathcal{L}_{\text{dist}}, \quad (12)$$

with $\alpha=0.1$ and $\beta=0.05$ in the main experiments. This stage shifts the model from reconstruction-oriented toward retrieval-oriented sparse scoring.

4 Experimental Setup

We evaluate on five M-BEIR datasets (Wei et al., 2024). Two are composed image retrieval tasks with image–text queries and image candidates: **CIRR** (Liu et al., 2021) and **FashionIQ** (Wu et al., 2021). Three are text-to-image retrieval tasks: **VisualNews**, **Fashion200K**, and **MSCOCO**. Together, their official training splits contain 485,482 queries and 977,506 positive query–candidate associations. Their test splits contain 4,170, 6,003, 19,995, 1,719, and 24,809 queries, respectively, with candidate pools ranging from 5,000 to 542,246 images. Full training, validation, test, and candidate statistics are reported in App. A. We focus on these two regimes as text-to-image retrieval is the standard cross-modal setting, whereas composed image retrieval tests whether a model can preserve visual context and a textual modification in one query representation.

Metrics. We report Hit@5, Recall@10, and nDCG@10. Hit@5 measures whether at least one relevant candidate appears in the top 5 and is the standard headline metric for CIRR and FashionIQ. Recall@10 measures coverage at a deeper cutoff, while nDCG@10 weights both relevance and rank position. We use nDCG@10 for validation.

Backbones. We focus on universal multimodal embedding models because they directly output shared embeddings for text, images, and image-text inputs, without requiring late fusion of modality-specific representations. They also represent the current strong-performing class of multimodal retrievers. Our main experiments use Qwen3-VL-Embedding-2B (Qwen-2B) (Li et al., 2026), which produces $d=2048$ dimensional embeddings. We also evaluate Qwen3-VL-Embedding-8B (Qwen-8B, $d=4096$) (Li et al., 2026) for within-family scaling and RZenEmbed (Jian et al., 2025) ($d=3584$) for cross-family transfer. Dense baselines and all sparse variants use identical prompts, cached embeddings, and candidate pools, to isolate sparsification effects.

Baselines. We compare PUMA against four baselines. *Full dense* retrieval ranks candidates with the original frozen embeddings. *Raw TopK dense* keeps the largest-magnitude dense coordinates with signed weights, representing a naïve sparsification baseline. *Matched-memory PCA* projects dense embeddings to dimension $2k$ using PCA fit on the candidate pool, approximately matching the storage of a sparse code with support k under FP32 (id, weight) accounting. *Trained dense AE* is a linear autoencoder with bottleneck dimension $2k$, trained on the same data with the same retrieval losses, optimization budget, and validation protocol as PUMA where applicable; sparsity-specific components such as AuxK, TopK support annealing, and support-mask alignment are omitted or replaced by dense analogs. *Encoder-only TopK* keeps the TopK sparse encoder and retrieval supervision but removes the decoder and reconstruction path, testing whether the autoencoding structure contributes beyond sparse retrieval supervision alone.

Implementation details. All sparse models are trained on cached dense embeddings for the official M-BEIR training query-positive pairs, with the backbone frozen, at expansion ratio $16\times$ (so $D=32,768$ for Qwen-2B and analogously for

larger backbones). We reserve a fixed 5% of training queries for checkpoint selection by nDCG@10; validation splits are used to select the operating point k , after which the test split is evaluated once. Dense embeddings are extracted with vLLM when supported by the backbone, and sparse training/evaluation is implemented in PyTorch. All experiments are run on a node equipped with a single NVIDIA A100 GPU. We use linear warmup with cosine learning-rate decay, decoder column normalization after each step, and progressive k -annealing to the target support. Storage, and search-time accounting, along with full hyperparameters and hardware details, are reported in Sec. 5.4 and App. F.

5 Results

We first evaluate PUMA on Qwen3-VL-Embedding-2B across all five M-BEIR tasks, then test whether the result transfers to Qwen3-VL-Embedding-8B and RZenEmbed. We next compare against a trained dense autoencoder, simpler dense and sparse compression baselines, analyze storage and search time, and ablate the main training components. Unless otherwise stated, significance is measured with 95% bootstrap intervals and 1000-trial paired approximate-randomization tests on nDCG@10.

5.1 Main Results on Qwen-2B

Tab. 1 reports results on the five M-BEIR tasks using Qwen3-VL-Embedding-2B. PUMA is competitive with dense retrieval across the benchmark family: it significantly improves over dense retrieval on CIRR and Fashion200K, is statistically indistinguishable on FashionIQ and MSCOCO, and trails dense only on VisualNews. On CIRR, PUMA improves nDCG@10 from 0.4235 to 0.4356 ($p \leq 0.001$), while also improving Hit@5 and Recall@10. On Fashion200K, the gain is larger, with nDCG@10 increasing from 0.0792 to 0.1144 ($p \leq 0.001$). On FashionIQ and MSCOCO, the differences are small and not statistically reliable. VisualNews is the only Qwen-2B setting where dense retrieval remains clearly ahead. Overall, PUMA matches or exceeds dense retrieval on four of five tasks by the primary criterion of competitive or better nDCG@10, while using substantially smaller sparse representations. Representative bootstrap intervals are reported in App. B.

Table 1: Main results across three backbones, with the original dense models, a matched-memory trained autoencoder (TrAE) and PUMA. TrAE is trained with bottleneck dimension $2k$. **Bold** and underline denote the best and second-best method per (data, backbone, metric). PUMA and TrAE use $k=160$ for Qwen-2B/RZenEmbed on CIRR, FashionIQ, and VisualNews; $k=128$ for Fashion200K; and $k=144$ for MSCOCO. For Qwen-8B, $k=144$ on CIRR and $k=128$ elsewhere.

Data	Method	Qwen-2B			Qwen-8B			RZenEmbed		
		H@5	R@10	N@10	H@5	R@10	N@10	H@5	R@10	N@10
CIRR	Dense	<u>.5221</u>	<u>.6281</u>	<u>.4235</u>	.5570	.6577	.4961	.5930	.6914	.4860
	TrAE	.4784	.5849	.3885	.5233	.6210	.4354	.4962	.5963	.4087
	PUMA	.5329	.6449	.4356	<u>.5392</u>	<u>.6438</u>	<u>.4835</u>	<u>.5664</u>	<u>.6743</u>	<u>.4713</u>
FashionIQ	Dense	.2029	.2680	.1706	<u>.2321</u>	<u>.3027</u>	.1942	<u>.2342</u>	<u>.3061</u>	.1952
	TrAE	.1621	.2183	.1327	.1777	.2345	.1478	.1899	.2529	.1557
	PUMA	.2062	.2697	<u>.1681</u>	.2348	.3055	<u>.1922</u>	.2424	.3144	.1994
VisualNews	Dense	.2666	.3318	.2256	.2720	.3396	.2296	.4587	.5410	.3933
	TrAE	.2531	.3196	.2125	<u>.3000</u>	<u>.3772</u>	<u>.2523</u>	.4204	.5048	.3571
	PUMA	<u>.2590</u>	<u>.3243</u>	<u>.2168</u>	.3052	.3820	.2561	<u>.4215</u>	<u>.5069</u>	<u>.3588</u>
Fashion200K	Dense	.1198	.0879	.0792	.1646	.1215	.1048	.2054	.1521	.1339
	TrAE	<u>.1699</u>	<u>.1226</u>	<u>.1058</u>	<u>.1856</u>	<u>.1308</u>	<u>.1183</u>	<u>.2147</u>	<u>.1580</u>	<u>.1392</u>
	PUMA	.1763	.1355	.1144	.2071	.1595	.1318	.2478	.1813	.1579
MSCOCO	Dense	.8049	.8721	.7181	.8184	.8835	<u>.7349</u>	.8447	.9012	.7630
	TrAE	<u>.8085</u>	<u>.8778</u>	.7196	.8267	.8884	.7367	<u>.8301</u>	<u>.8903</u>	<u>.7445</u>
	PUMA	.8098	.8805	<u>.7191</u>	<u>.8220</u>	<u>.8871</u>	.7297	.8270	.8884	.7384

5.2 Scaling and Cross-Family Transfer

To test generalization, we use the same sparsification recipe on Qwen3-VL-Embedding-8B and RZenEmbed, changing only cached embeddings.

Within-family scaling: Qwen-8B. On Qwen3-VL-Embedding-8B (Tab. 1), PUMA improves over dense retrieval on the two larger text-to-image datasets, VisualNews and Fashion200K, by about $+0.027$ nDCG@10 in both cases ($p \leq 0.001$). It remains close to dense on MSCOCO and FashionIQ, but falls below dense on CIRR. Thus, scaling the dense backbone strengthens PUMA on text-to-image retrieval, but does not automatically make composed retrieval easier to sparsify.

Cross-family transfer: RZenEmbed. On RZenEmbed (Tab. 1), PUMA exceeds dense on FashionIQ and Fashion200K, with the Fashion200K gain statistically reliable ($p \leq 0.001$) and FashionIQ trending positive ($p=0.075$). It remains below dense on CIRR, VisualNews, and MSCOCO. This shows that transfer is real but uneven across backbones. More broadly, the RZenEmbed results suggest that the PUMA recipe is not specific to the Qwen family, but that its success depends on backbone geometry and task regime. In particular, the mixed transfer pattern is consistent with the two-way diagnosis developed below: some settings fail because support is not available before TopK,

while others fail because the available support is not retrieval-aligned enough.

Failure diagnostics. The two hardest settings fail for different reasons (Tab. 2). On Qwen-8B CIRR, the encoder produces only 166.5 positive pre-activations on average and realizes $k=160$ on only 81.8% of examples, so the requested support is not always available before TopK. On RZen VisualNews, support is available: the model produces 465.3 positive pre-activations on average and reaches $k=160$ on every example. Yet sparse nDCG@10 remains well below dense, indicating that the active features are not retrieval-aligned. Thus support availability is necessary but not sufficient; the active support must also preserve dense retrieval geometry. These diagnostics make the failure modes interpretable: post-hoc sparsification can fail either because the encoder does not produce enough pre-TopK support, or because the available support does not preserve retrieval rankings. Both diagnostics can be monitored during training and point to objectives that jointly improve dictionary coverage and retrieval alignment.

5.3 Alternative Compression Techniques

We compare PUMA with baselines that test whether its gains can be explained by simpler forms of compression or adaptation. Tab. 1 benchmarks PUMA with a learned dense compressor at the

Table 2: Support diagnostics for two hard settings. Pos-pre: mean positive pre-TopK activations; Reach: fraction realizing k exactly; Used: distinct active features; PUMA/Dn: PUMA/dense nDCG@10.

Setting	k	Pos-pre	Reach	Used	PUMA/Dn
8B CIRR	160	166.5	0.818	199	.484/.496
RZen VisN	160	465.3	1.000	497	.359/.393

same memory budget (TrAE). The trained dense autoencoder uses bottleneck dimension $2k$ and is trained on the same data with the same retrieval losses, optimization budget, and validation protocol as PUMA where applicable; sparsity-specific components such as AuxK, TopK annealing, and support-mask alignment are omitted or replaced by dense analogs. Across 15 model–dataset settings, PUMA has higher nDCG@10 in 12. TrAE is higher only on MSCOCO for Qwen-2B, Qwen-8B, and RZenEmbed. Thus adaptation alone does not explain PUMA’s gains: at the same per-vector budget, the sparse representation is usually the stronger retrieval representation.

We further compare PUMA with three compression baselines that isolate direct sparsification, dense compression, and decoder-backed reconstruction. Raw TopK dense coordinates test whether direct coordinate pruning is enough; matched-memory PCA tests whether a smaller dense representation suffices; and encoder-only TopK keeps the sparse encoder and retrieval supervision but removes the decoder, testing whether sparse supervision alone works without reconstruction. Tab. 3 compares these controls on three representative Qwen-2B settings. Raw TopK is substantially weaker: on CIRR, it reaches 0.3769 nDCG@10 versus PUMA’s 0.4356. Matched-memory PCA is stronger (0.4153) but remains below both dense and PUMA. The encoder-only sparse model reaches 0.3631 on CIRR, above Raw TopK but clearly below PUMA, showing that retrieval supervision alone does not replace the decoder/reconstruction path. The same pattern appears on FashionIQ and Fashion200K. Thus coordinate pruning, dense memory reduction, and sparse supervision without reconstruction are insufficient.

We additionally compare PUMA with two standard dense compression and indexing baselines on Qwen-8B embeddings: exact asymmetric-distance scoring with 128-byte Product Quantization codes (PQ-M128) (Jégou et al., 2011), and a IVF-PQ64×8 index (Jégou et al., 2011; Johnson

Table 3: Alternative compressors vs. PUMA vs. base uncompressed model (Dense) on three Qwen-2B settings. Raw TopK and PCA at $k=160$ (PCA dim = $2k = 320$); Enc-only and PUMA at $k=160$ except F200K at $k=128$. **Bold** = best per (data, metric); underline = second best.

Data	Method	Hit@5	R@10	nDCG@10
CIRR	Dense	<u>.5221</u>	<u>.6281</u>	<u>.4235</u>
	Raw TopK	.4655	.5688	.3769
	PCA	.5094	.6124	.4153
	Enc-only	.4518	.5614	.3631
	PUMA	.5329	.6449	.4356
FashionIQ	Dense	<u>.2029</u>	<u>.2680</u>	.1706
	Raw TopK	.1621	.2168	.1355
	PCA	.1891	.2504	.1574
	Enc-only	.1667	.2263	.1367
	PUMA	.2062	.2697	<u>.1681</u>
Fashion200K	Dense	.1198	.0879	.0792
	Raw TopK	.0762	.0621	.0513
	PCA	.0861	.0652	.0538
	Enc-only	<u>.1495</u>	<u>.1164</u>	<u>.0967</u>
	PUMA	.1763	.1355	.1144

et al., 2021). Tab. 4 reports the same three retrieval metrics as our other baseline comparisons. PUMA obtains higher Hit@5, Recall@10, and nDCG@10 than both alternatives on every dataset. Thus, the observed quality is not reproduced by replacing the sparse codes with standard dense PQ compression or IVF-PQ indexing at the evaluated configurations.

Together, these controls rule out simpler explanations for PUMA’s gains. Raw TopK shows that direct sparsification of dense coordinates is insufficient; matched-memory PCA shows that memory reduction alone is insufficient; the trained dense AE (TrAE) shows that supervised dense adaptation is usually insufficient; and encoder-only TopK shows that sparse retrieval supervision without a decoder is insufficient. The evidence instead supports the full recipe: sparse structure, decoder-backed reconstruction, and retrieval-aligned fine-tuning.

5.4 Efficiency: Storage and Search Time

PUMA reduces storage by representing each vector with only k feature ids and weights. Under FP32 accounting, dense embeddings cost $4d$ bytes, while sparse codes cost $8k$ bytes with 32-bit ids and FP32 weights. At our operating points ($k \in [128, 160]$), this gives an $8\times$ reduction for Qwen-2B ($d=2048$) and up to $16\times$ for Qwen-8B ($d=4096$).

The $8k$ accounting is conservative. In a fixed index, feature ids only need to address atoms that appear in the candidate pool. Remapping corpus-active atoms at index construction leaves sparse

Table 4: Qwen-8B comparison with PQ-M128 and IVF-PQ64 $\times$ 8. PUMA uses the operating points in Tab. 1. **Bold** denotes the best result per dataset and metric.

Data	Method	H@5	R@10	N@10
CIRR	PUMA	.5392	.6438	.4835
	PQ-M128	.3281	.4094	.2718
	IVF-PQ64	.4297	.5299	.3547
FashionIQ	PUMA	.2348	.3055	.1922
	PQ-M128	.1301	.1729	.1084
	IVF-PQ64	.1328	.1831	.1087
VisualNews	PUMA	.3052	.3820	.2561
	PQ-M128	.1323	.1705	.1095
	IVF-PQ64	.1133	.1546	.0933
Fashion200K	PUMA	.2071	.1595	.1318
	PQ-M128	.0739	.0528	.0442
	IVF-PQ64	.1018	.0778	.0618
MSCOCO	PUMA	.8220	.8871	.7297
	PQ-M128	.6939	.7881	.6094
	IVF-PQ64	.6747	.7554	.5870

vectors and retrieval scores unchanged, but reduces id width. In our diagnostics, at most 497 atoms are active, so 9-bit ids suffice; at $k=128$, this reduces PUMA storage from 1,024 to 656 bytes per vector. This remapping is optional, so the trend remains the deployment-agnostic 8–16 $\times$ reduction.

We also compare against quantized dense storage. FP16 and INT8 dense retrieval are essentially lossless in our measurements: on representative settings spanning CIRR, Fashion200K, and VisualNews at Qwen-2B and Qwen-8B, nDCG@10 changes by at most 0.001 between FP32, FP16, and INT8 dense retrieval. For example, CIRR-2B gives 0.4239/0.4241/0.4236 for FP32/FP16/INT8, and VisualNews-8B gives 0.2292/0.2292/0.2292. Thus FP16 gives a free 2 $\times$ dense-storage reduction and INT8 gives a free 4 $\times$ reduction. PUMA remains smaller even against INT8 dense: with 9-bit ids and FP32 sparse weights, PUMA uses 656 bytes per vector, compared with 2,048 bytes for INT8 dense at Qwen-2B and 4,096 bytes at Qwen-8B. Quantization is complementary to PUMA, since the sparse weights can also be quantized; thus, the FP32-weight accounting in Tab. 5 is conservative.

PUMA also improves exact search time on large candidate pools. On VisualNews 8B (542K candidates), exact dense search takes 423.8 s versus 17.3 s for sparse search (24.5 $\times$ faster). On Fashion200K 8B (202K candidates), the gap is 13.8 s versus 0.77 s (17.9 $\times$ faster). Dense search remains faster only on the much smaller MSCOCO pool, where sparse-kernel overhead dominates the

Table 5: Per-vector storage in bytes at $k=128$. Compact ids remap logical atoms to corpus-active ids of width $\lceil \log_2 D_{\text{active}} \rceil$ bits. The 9-bit row is a worst-case bound across our measured settings; the 8-bit row applies when $D_{\text{active}} \leq 256$ (e.g., Qwen-8B CIRR).

Representation	2B (B)	8B (B)	vs full dense
Full dense ($4d$, FP32)	8192	16384	1 $\times$
Full dense ($2d$, FP16)	4096	8192	2 $\times$
Full dense (d , INT8)	2048	4096	4 $\times$
Trained dense AE ($2k$, FP32)	1024	1024	8 / 16 $\times$
PUMA, 32-bit ids ($8k$)	1024	1024	8 / 16 $\times$
PUMA, 16-bit ids ($6k$)	768	768	10.7 / 21.3 $\times$
PUMA, 9-bit ids	656	656	12.5 / 25.0 $\times$
PUMA, 8-bit ids	—	640	— / 25.6 $\times$

low candidate count. These timings compare exact dense and exact sparse scoring to isolate the representation-level effect.

5.5 Ablation: Which Components Matter?

We use leave-one-out ablations on two Qwen-2B tasks: CIRR for composed retrieval and Fashion200K for text-to-image retrieval. Each run removes one loss or stage while keeping the remaining training protocol fixed. The ablation uses a fixed $k=128$ to compare variants under a common support; main results use the validation-selected operating points in Tab. 1.

Stage ② leave-one-out. We remove one Stage ② loss at a time while keeping the rest of PUMA fixed (Tab. 6). All three terms are load-bearing, but their importance differs by task. On CIRR, every single ablation drops PUMA below dense, with AuxK feature revival causing the largest loss. On Fashion200K, the Stage ② contrastive blend is most important, while cross-modal alignment and AuxK still provide clear gains. Thus the full recipe is needed for the composed-retrieval win, whereas Fashion200K remains above dense.

Stage ③ leave-one-out. The Stage ③ row evaluates the Stage ② checkpoint directly, without contrastive fine-tuning. On Qwen-2B, removing Stage ③ reduces nDCG@10 from .4333 to .4140 on CIRR and from .1144 to .1050 on Fashion200K; on Qwen-8B FashionIQ, it reduces nDCG@10 from .1922 to .1883. For Qwen-8B CIRR, at the matched $k=144$ operating point, the full model obtains .4793 and the no-Stage ③ variant obtains .4793; the difference is not statistically significant ($p=.115$). As diagnosed in Sec. 5.2, this setting is support-limited before TopK. Thus Stage ③ im-

Table 6: Leave-one-out ablation at Qwen-2B. Stage ② rows zero one term of Eq. (6); Stage ③ OFF skips contrastive fine-tuning. Other settings match full PUMA. n@10 is nDCG@10; Δ is relative to PUMA.

Variant	k	CIRR		Fashion200K	
		n@10	Δ	n@10	Δ
Dense	—	.4235		.0792	
PUMA (full)	128	.4333	—	.1144	—
<i>Stage ② (Eq. (6))</i>					
$-\mathcal{L}_{\text{align}}$	128	.4072	−.026	.1009	−.014
$-\mathcal{L}_{\text{aux}}$	128	.3975	−.036	.0995	−.015
$-\mathcal{L}_{\text{ctr}}^{(2)}$	128	.4061	−.027	.0857	−.029
<i>Stage ③:</i>					
Stage ③ OFF	128	.4140	−.019	.1050	−.009

proves the point estimate in these evaluated settings, although its CIRR gain at Qwen-8B is small and pre-TopK support availability remains an important diagnostic. Additional loss coefficient sweep are reported in App. F.

6 Conclusion

We presented **PUMA**, a post-hoc recipe that converts frozen universal multimodal embeddings into compact sparse retrieval codes. On Qwen3-VL-Embedding-2B, PUMA is competitive with dense retrieval across five M-BEIR tasks, with significant gains on CIRR and Fashion200K, near parity on FashionIQ and MSCOCO, and a remaining gap on VisualNews. The gains are not explained by dimensionality reduction alone: raw TopK, PCA, trained matched-memory dense autoencoders, and encoder-only sparse training all fall short in representative comparisons. The remaining challenge is to jointly ensure sufficient pre-TopK support and retrieval alignment of the active support, the two axes isolated by our failure diagnostics.

Limitations

This work opens several directions for future study. First, our main experiments focus on the Qwen embedding family, with RZenEmbed included as a cross-family check. The transfer results show that PUMA is not Qwen-specific, but they also suggest that sparsification behavior depends on the geometry of the frozen backbone. Evaluating additional universal multimodal embedders would clarify how broadly the recipe applies.

Second, our failure analysis diagnoses two bottlenecks but does not solve them. In particular, the Qwen-8B CIRR setting shows that stronger

dense backbones can become support-limited before TopK selection, while RZenEmbed VisualNews shows that available support is not always retrieval-aligned. Future work should develop objectives that jointly encourage sufficient dictionary coverage and retrieval-aligned active features.

Finally, PUMA is a post-hoc method and therefore inherits the representational properties of the frozen backbone. This is desirable for efficiency because the backbone is not retrained, but it also means that any domain imbalance or bias in the dense embedding space may propagate to the sparse codes. Deployment-sensitive settings should therefore audit the sparsified retriever alongside the original dense model.

Acknowledgments

This work has been carried out while *Matteo Attimonelli* was enrolled in the Italian National Doctorate on Artificial Intelligence run by Sapienza University of Rome in collaboration with *Politecnico Di Bari*. We acknowledge the CINECA award under the ISCRA initiative for the availability of high-performance computing resources and support. We acknowledge ISCRA for awarding this project access to the LEONARDO supercomputer, owned by the EuroHPC Joint Undertaking, hosted by CINECA (Italy). This publication was funded by the following projects: "New Kocs Era" (project code CDP001540; CUP B85H24002010007); "AURA – Augmented Unified Resource for Agents" (project code CDP001572); This research was conducted as part of the ENRESFOOD project, funded under the FutureFoodS partnership. The work received funding from the Austrian Science Fund (FWF; 10.55776/KIN4661525), the Dutch Ministry of Agriculture, Fisheries, Food Security and Nature (BO-43-219-026), the Italian Ministry of University and Research (MUR), and the German Federal Ministry of Research, Technology and Space (031B1717).

References

- Matteo Attimonelli, Alessandro De Bellis, Claudio Pomo, Dietmar Jannach, Eugenio Di Sciascio, and Tommaso Di Noia. 2025. Do we really need specialization? evaluating generalist text embeddings for zero-shot recommendation and search. In *RecSys*, pages 575–580. ACM.
- Trenton Bricken, Adly Templeton, Joshua Batson,

- Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, and 6 others. 2023. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*.
- Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2021a. SPLADE v2: Sparse lexical and expansion model for information retrieval. *CoRR*, abs/2109.10086.
- Thibault Formal, Maxime Louis, Hervé Déjean, and Stéphane Clinchant. 2026. Learning retrieval models with sparse autoencoders. In *ICLR*. OpenReview.net.
- Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021b. SPLADE: sparse lexical and expansion model for first stage ranking. In *SIGIR*, pages 2288–2292. ACM.
- Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. 2025. Scaling and evaluating sparse autoencoders. In *ICLR*. OpenReview.net.
- Lixuan Guo, Yifei Wang, Tiansheng Wen, Yifan Wang, Aosong Feng, Bo Chen, Stefanie Jegelka, and Chenyu You. 2026. Csr2: Unlocking ultra-sparse embeddings. In *ICLR*. OpenReview.net.
- Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating large-scale inference with anisotropic vector quantization. In *ICML*, Proceedings of Machine Learning Research, pages 3887–3896. PMLR.
- Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. 2024. Sparse autoencoders find highly interpretable features in language models. In *ICLR*. OpenReview.net.
- Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128.
- Weijian Jian, Yajun Zhang, Dawei Liang, Chunyu Xie, Yixiao He, Dawei Leng, and Yuhui Yin. 2025. Rzen-embed: Towards comprehensive multimodal retrieval. *CoRR*, abs/2510.27350.
- Ziyan Jiang, Rui Meng, Xinyi Yang, Semih Yavuz, Yingbo Zhou, and Wenhui Chen. 2025. Vlm2vec: Training vision-language models for massive multimodal embedding tasks. In *ICLR*. OpenReview.net.
- Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-scale similarity search with gpus. *IEEE Trans. Big Data*, 7(3):535–547.
- Chiraag Kaushik, Davis Barch, and Andrea Fanelli. 2026. Learning multimodal dictionary decompositions with group-sparse autoencoders. In *ICLR*. OpenReview.net.
- Mingxin Li, Yanzhao Zhang, Dingkun Long, Keqin Chen, Sibao Song, Shuai Bai, Zhibo Yang, Pengjun Xie, An Yang, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2026. Qwen3-vl-embedding and qwen3-vl-reranker: A unified framework for state-of-the-art multimodal retrieval and ranking. *CoRR*, abs/2601.04720.
- Sheng-Chieh Lin, Chankyu Lee, Mohammad Shoeybi, Jimmy Lin, Bryan Catanzaro, and Wei Ping. 2025. Mm-embed: Universal multimodal retrieval with multimodal LLMS. In *ICLR*. OpenReview.net.
- Zheyuan Liu, Cristian Rodriguez Opazo, Damien Teney, and Stephen Gould. 2021. Image retrieval on real-life images with pre-trained vision-and-language models. In *ICCV*, pages 2105–2114. IEEE.
- Alireza Makhzani and Brendan J. Frey. 2014. k-sparse autoencoders. In *ICLR (Poster)*.
- Gregor Meehan and Johan Pauwels. 2026. Sparse contrastive learning for content-based cold item recommendation. In *SIGIR*, pages 3994–3999. ACM.
- Thong Nguyen, Yibin Lei, Jia-Huei Ju, Eugene Yang, and Andrew Yates. 2026. Milco: Learned sparse retrieval across languages via a multilingual connector. In *ICLR*. OpenReview.net.
- Thong Nguyen, Yibin Lei, Jia-Huei Ju, and Andrew Yates. 2025. SERVAL: surprisingly effective zero-shot visual document retrieval powered by large vision and language models. In *EMNLP*, pages 30807–30822. Association for Computational Linguistics.
- Chuan Qin, Constantin Venhoeff, Sonia Joseph, Fanyi Xiao, and Stefan Scherer. 2026. Sparse CLIP: co-optimizing interpretability and performance in contrastive learning. In *ICLR*. OpenReview.net.
- Vojtech Vancura, Martin Spisák, Rodrigo Alves, and Ladislav Peska. 2026. Efficient learning of sparse representations from interactions. In *WWW*, pages 8577–8580. ACM.
- Cong Wei, Yang Chen, Haonan Chen, Hexiang Hu, Ge Zhang, Jie Fu, Alan Ritter, and Wenhui Chen. 2024. Uniir: Training and benchmarking universal multimodal information retrievers. In *ECCV (87)*, Lecture Notes in Computer Science, pages 387–404. Springer.
- Tiansheng Wen, Yifei Wang, Zequn Zeng, Zhong Peng, Yudi Su, Xinyang Liu, Bo Chen, Hongwei Liu, Stefanie Jegelka, and Chenyu You. 2025. Beyond matryoshka: Revisiting sparse coding for adaptive representation. In *ICML*, Proceedings of Machine Learning Research. PMLR / OpenReview.net.
- Hui Wu, Yupeng Gao, Xiaoxiao Guo, Ziad Al-Halah, Steven Rennie, Kristen Grauman, and Rogério Feris. 2021. Fashion IQ: A new dataset towards retrieving images by natural language feedback. In *CVPR*, pages 11307–11317. Computer Vision Foundation / IEEE.

Xin Zhang, Yanzhao Zhang, Wen Xie, Mingxin Li, Ziqi Dai, Dingkun Long, Pengjun Xie, Meishan Zhang, Wenjie Li, and Min Zhang. 2025. Bridging modalities: Improving universal multimodal retrieval by multimodal large language models. In *CVPR*, pages 9274–9285. Computer Vision Foundation / IEEE.

A Additional Data Statistics

Tab. 7 reports the official split sizes used in our experiments. Train positives are the positive query–candidate associations in the training qrels. Candidates denotes the official test candidate pool and is therefore an evaluation statistic, not a count of training examples. Across the five tasks, the supervision pool contains 485,482 training queries and 977,506 positive associations.

B Bootstrap Confidence Intervals

Table 8: Representative 95% bootstrap intervals for nDCG@10 on Qwen-2B. Rows show a PUMA win, near parity, and a dense win.

Data	Dense	PUMA
CIRR	.4233 [.4105,.4349]	.4331 [.4214,.4446]
FashIQ	.1705 [.1624,.1785]	.1681 [.1602,.1760]
VisN	.2258 [.2209,.2308]	.2169 [.2122,.2218]

C Pooled-Adapter Generalization

The main experiments train a separate sparse adapter for each retrieval task. We therefore ask whether the frozen universal backbone can instead be served through one shared sparse adapter, and how much retrieval quality is lost relative to task-specific training, complementing Sec. 5.2.

Using cached Qwen-2B embeddings, we trained a single SAE on the five datasets together. The pooled adapter retains 92.7–99.0% of task-specific nDCG@10 on CIRR, FashionIQ, VisualNews, and MSCOCO, but only 61.4% on Fashion200K. A single adapter can therefore preserve most quality on four tasks, but task-specific adaptation remains important, particularly for Fashion200K. We consequently use task-specific adapters for the headline results.

Table 9: One pooled Qwen-2B adapter evaluated across all tasks. Task-specific PUMA uses the validation-selected operating point in Tab. 1; Pooled uses one checkpoint at $k=128$.

Data	Task-specific	Pooled	Retention
CIRR	.4356	.4097	94.1%
FashionIQ	.1681	.1558	92.7%
VisualNews	.2168	.2019	93.1%
Fashion200K	.1144	.0702	61.4%
MSCOCO	.7191	.7119	99.0%

D Support-Usage Diagnostics

Tab. 10 reports pre-activation statistics for our checkpoints. Encoder-only training and the final SAE are both well past support saturation on 2B (mean positive pre-activations $\gg k$, exact realization of the requested k on every example). On 8B, the SAE’s positive pre-activations drop sharply, explaining the support-limited regime described in Sec. 5.2.

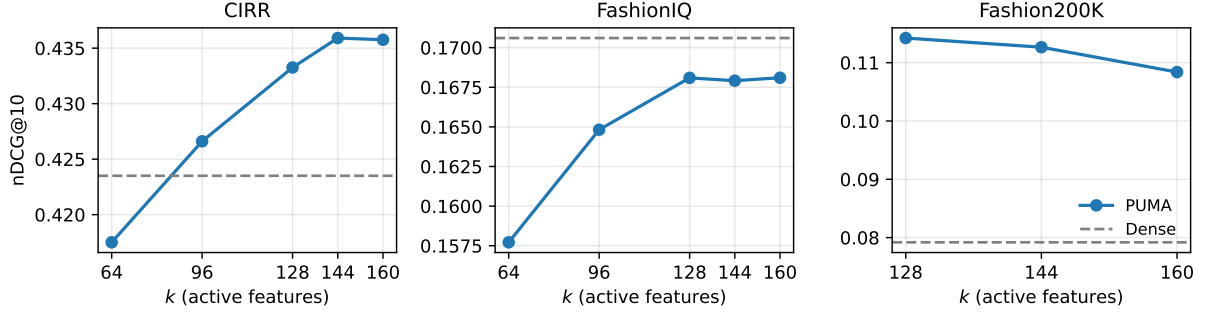
E Additional Ablation Study

Effect of the target sparsity k . Fig. 2 sweeps the inference-time support $k \in \{64, 96, 128, 144, 160\}$ at Qwen-2B on CIRR, FashionIQ, and Fashion200K. Quality saturates around $k = 128$: increasing k further yields diminishing returns, while $k = 64$ already reaches near-saturation on FashionIQ and Fashion200K. CIRR is the most k -sensitive of the three (the most stringent regime, as expected for composed queries), but its curve also flattens past $k = 128$. Combined with the storage analysis in Sec. 5.4, this places the practical operating point in the $k \in [96, 160]$ band: smaller k gives a tighter index, larger k recovers a tail of the dense ranking. Expansion ratio $D = 16d$ is held fixed across all experiments; varying D is left for future work.

Cross-modal alignment impact. The Stage ② LOO above shows that removing $\mathcal{L}_{\text{align}}$ costs retrieval quality, but does not show that the loss actually drives paired queries and candidates onto a common subset of dictionary atoms. To check this directly, we measure the Jaccard overlap of the TopK supports of paired and random query–candidate pairs on the trained PUMA-2B checkpoints (Tab. 11). On all three datasets the paired support overlap is substantially larger than the random baseline: +0.177 on CIRR, +0.134 on FashionIQ, and +0.088 on Fashion200K. The lift is largest on the two composed-retrieval datasets, which is where the alignment loss is supervised on genuinely cross-modal $(\mathbf{q}, \mathbf{d}^+)$ pairs. Across all three datasets fewer than 500 atoms ever fire, mirroring the concentration observed in Tab. 2, so the alignment behavior holds within an active dictionary that is two orders of magnitude smaller than the trainable $D = 16d$.

Table 7: M-BEIR split and supervision statistics.

Dataset	Train queries	Train positives	Val queries	Test queries	Candidates
CIRR	26,116	26,676	2,000	4,170	21,551
FashionIQ	16,176	16,266	1,700	6,003	74,381
VisualNews	199,903	200,000	19,996	19,995	542,246
Fashion200K	30,000	63,945	1,746	1,719	201,824
MSCOCO	213,287	670,619	24,819	24,809	5,000
Total	485,482	977,506	50,261	56,696	—

Figure 2: PUMA nDCG@10 at Qwen-2B as the target sparsity k varies, on CIRR, FashionIQ, and Fashion200K. Dashed line: dense baseline.Table 10: Support-usage diagnostics ($k=128$). Pos-pre: mean positive pre-activations before TopK. Active: mean surviving features at native support. Reach: fraction of examples realizing k exactly.

Data	Setting	Pos-pre	Active	Reach
CIRR	Enc-only 2B	3774.4	128.0	1.000
CIRR	PUMA 2B	363.8	128.0	1.000
CIRR	PUMA 8B	167.5	128.0	1.000

Table 11: Cross-modal TopK-support Jaccard on Qwen-2B PUMA ($k=128$, $D=32,768$). Paired: (query, positive candidate) pairs. Random: query paired with a uniformly sampled candidate. Lift is paired — random.

Dataset	Pairs	Paired J	Random J	Lift	$ D_{\text{active}} $
CIRR	4170	0.351	0.174	+0.177	481
FashionIQ	6003	0.314	0.180	+0.134	424
Fashion200K	1719	0.211	0.123	+0.088	447

Ultra-sparse regime ($k \leq 64$). Although our headline operating points sit in $k \in [64, 160]$, two questions remain: (i) how steeply does PUMA degrade as k shrinks toward the ultra-sparse regime, and (ii) how cheap can the index be made before retrieval collapses? Tab. 12 extends the sweep with $k \in \{16, 32, 48, 64\}$ on the same three 2B datasets. The curve is smooth: nDCG@10 at $k=16$ already captures 74% of the dense score on

CIRR (0.313/0.4235) and 69% on Fashion200K (0.054/0.0792). By $k=48$ PUMA reaches 96% of dense on CIRR (0.407) and *exceeds* dense on Fashion200K (0.110 vs 0.079, where dense is weak). The implication is a per-corpus storage knob: with 9-bit ids, $k=16$ corresponds to ~ 82 bytes per vector ($\sim 100\times$ smaller than FP32 dense at 2B), trading $\sim 25\%$ of dense quality on the hardest of the three tasks; $k=48$ at ~ 246 bytes ($\sim 33\times$) preserves it within 4%. The ultra-sparse rows therefore extend the sparsity-vs-quality frontier of Fig. 2 rather than break it.

F Additional Training Details

We train SAEs for two stages on cached query/candidate embeddings from each M-BEIR task. Stage ② (sparse pretraining) optimizes the objective in Eq. (6) with cosine reconstruction, retrieval-similarity supervision, AuxK feature revival, and a small Stage ② contrastive blend; Stage ③ (retrieval fine-tuning) emphasizes InfoNCE on sparse codes with reconstruction and distillation kept as small regularizers. Both stages use AdamW with linear warmup and cosine decay, decoder column-norm renormalization after each step, and progressive k -annealing toward the target sparsity. For Qwen3-VL-Embedding-2B we use $D=32,768$; for 8B we use the analogously expanded $D=16d$. A fixed 5% of official train-

Table 12: Ultra-sparse PUMA at Qwen-2B (test split). Rows extend Fig. 2 below the $k=64$ point. All numbers measured from the same checkpoints used in Tab. 1, with k varied only at inference.

Setting	k	Hit@5	R@10	nDCG@10
CIRR	dense	.5221	.6281	.4235
	16	.3784	.4874	.3133
	32	.4540	.5622	.3743
	48	.4887	.6038	.4070
	64	.4990	.6185	.4177
FashionIQ	dense	.2029	.2680	.1706
	16	.1129	.1566	.0914
	32	.1609	.2152	.1297
	48	.1821	.2401	.1478
	64	.1922	.2580	.1580
Fashion200K	dense	.1198	.0879	.0792
	16	.0861	.0799	.0543
	32	.1449	.1326	.0900
	48	.1629	.1556	.1099
	64	.1763	.1654	.1151

ing queries, sampled with the same seed for all runs, is held out for checkpoint selection by sparse nDCG@10.

Loss weights. Tab. 13 lists all loss weights. One coefficient configuration is reused unchanged across all datasets and backbones, without per-dataset tuning. The weights have distinct roles: $\lambda_a=\lambda_d$ controls alignment and dot-product distillation, λ_x controls AuxK feature revival, λ_c blends Stage ② InfoNCE, and α and β retain reconstruction and distillation during Stage ③. The leave-one-out results in Tab. 6 establish that these components contribute to retrieval quality, while the sensitivity analysis below tests whether the reported behavior requires a narrow coefficient setting.

Table 13: PUMA loss weights, held fixed across all datasets and backbones.

Weight	Value	Source / role
$\lambda_a=\lambda_d$	0.3	Alignment / dot-product distillation
λ_x	1/32	AuxK default (Gao et al., 2025)
λ_c	0.05	Stage ② InfoNCE blend (small regularizer)
α	0.1	Stage ③ reconstruction regularizer
β	0.05	Stage ③ distillation regularizer
τ	0.05	InfoNCE temperature

Hyperparameter-sensitivity analysis. This analysis complements Sec. 5.5 by testing broad coefficient perturbations on Qwen-2B CIRR and Fashion200K, representing composed image retrieval and text-to-image retrieval, respectively. The global defaults in Tab. 13 were fixed before

Table 14: Qwen-2B coefficient sensitivity at $k=128$. Entries are the nDCG@10 range across the indicated perturbations.

Stage	Perturbations	CIRR	Fashion200K
Dense reference	—	.4235	.0792
Stage ② auxiliaries	0, 0.5, $2\times$	.4107–.4272	.0871–.1090
Stage ③ regularizers	0, 0.5, 2, $4\times$	.4302–.4318	.1240–.1254

the sweep and reused for both datasets.

We run nine Stage ② variants, perturbing one auxiliary coefficient at a time: alignment, AuxK revival, or the Stage ② contrastive blend. Each coefficient is set to zero, $0.5\times$, or $2\times$ its global value, while all other coefficients and optimization settings are fixed. Each resulting checkpoint then undergoes the unchanged Stage ③ fine-tuning procedure and is evaluated at $k=128$. This one-at-a-time design isolates each component rather than jointly rescaling the Stage ② objective.

For Stage ③, all runs start from the same Stage ② checkpoint to isolate fine-tuning sensitivity. We jointly set the reconstruction and distillation regularizers (α, β) to zero or scale both by $0.5\times$, $2\times$, and $4\times$, leaving InfoNCE and the optimization protocol unchanged. Tab. 14 reports the minimum and maximum test nDCG@10 within each sweep, rather than selecting the best perturbation.

Stage ② is more sensitive because its auxiliary terms shape the sparse representation, with nDCG@10 ranging from .4107–.4272 on CIRR and .0871–.1090 on Fashion200K. Even so, the method remains stable under zeroing and twofold perturbations, and all Fashion200K variants remain above the .0792 dense reference. Stage ③ is less sensitive, with narrower ranges of .4302–.4318 and .1240–.1254. Together with the globally reused defaults, these results show that PUMA’s behavior is not due to per-dataset coefficient tuning or a narrowly selected optimum.

Optimization. AdamW with weight decay 10^{-2} . Learning rates: 3×10^{-4} (Stage ②), 1×10^{-4} (Stage ③), each with 1,000 / 500 linear warmup steps followed by cosine decay. Stage ② runs 40,000 steps; Stage ③ runs 5,000 steps. Batch size 64 throughout. Validation-based checkpoint selection uses the fixed 5% holdout of training queries described above.