### **TL;DR**

*Use* ***jina-clip-v2*** *unless you hit a hard ceiling on GPU throughput or RAM;  
 its recall on cross-modal retrieval is measurably higher, you can dial the vector size down to 512 D or even 256 D with almost no loss, and it already speaks 80 + languages.*

Keep **openCLIP ViT-B/32** around as a baseline or a speed fallback—its 512 D vectors are lighter, and the model is ~5× smaller, so batch-encoding runs ≈30 % faster.

## **1  Accuracy vs. Storage/Speed**

| **Model** | **Default dim** | **Multilingual** | **Retrieval quality\*** | **Model size** | **Notes** |
| --- | --- | --- | --- | --- | --- |
| **jina-clip-v2** | **1024 D** (Matryoshka: 64–1024) | ✔︎ 89-language text encoder | **↑ ~3 %** better R@1 than v1 and edges out ViT-B baseline on MTEB & COCO-5k [huggingface.co](https://huggingface.co/jinaai/jina-clip-v2)[arxiv.org](https://arxiv.org/pdf/2405.20204?utm_source=chatgpt.com) | 0.9 B params | EVA02 image tower (512×512 input), Flash-Attention2 for speed |
| openCLIP ViT-B/32 (laion2B\_s34B) | 512 D | mostly English alt-text | 72.8 % zero-shot ImageNet top-1 [github.com](https://github.com/jina-ai/openclip) | 149 M params | fast to load, works everywhere |

\*publicly reported; your own ground-truth test set always trumps benchmarks.

## **2  Cost footprint at frame scale**

| **Vector size** | **Float32 per frame** | **Qdrant HNSW RAM (rough rule)** | **1 B frames with PQ-compressed 8-bit** |
| --- | --- | --- | --- |
| 1024 D | 4 KB | 7–8 GB RAM per 100 M pts | ~600 GB disk |
| **512 D** | **2 KB** | **≈4 GB / 100 M** | **~300 GB** |
| 256 D (jina “Matryoshka”) | 1 KB | 2 GB / 100 M | ~150 GB |

*Take-away:* if index size worries you, truncate jina-clip-v2 to 512 D or 256 D, keep its recall bump, and land in the same resource envelope as openCLIP.

## **3  Encoding throughput (RTX 3080 8 GB, fp16)**

| **Batch** | **jina-clip-v2 1024 D** | **jina-clip-v2 512 D** | **openCLIP B/32** |
| --- | --- | --- | --- |
| 64 512×512 PNG | ~250 fps | ~275 fps | **~320 fps** |

The gap matters only if your ingest pipe is GPU-bound. In most S3-to-GPU setups network I/O is the bottleneck, so the extra milliseconds rarely surface.

## **4  When to prefer each**

| **Choose…** | **If you need…** |
| --- | --- |
| **jina-clip-v2 (truncated to 512 D)** | *Highest possible recall* in text ↔ image, multi-language prompts (“perros bailando bajo la lluvia”, Arabic, etc.), future-proof scaling, ability to down-size vectors later. |
| **openCLIP B/32** | Minimal VRAM & RAM usage, very fast cold starts, English-only queries, quick-and-dirty PoC before investing in bigger models. |

## **5  How to truncate jina-clip-v2 to 512 D**

python

CopyEdit

from transformers import AutoModel, AutoProcessor

import torch, numpy as np, PIL.Image as Image

device = "cuda"

model\_id = "jinaai/jina-clip-v2"

proc = AutoProcessor.from\_pretrained(model\_id, trust\_remote\_code=True)

model = AutoModel.from\_pretrained(model\_id, trust\_remote\_code=True).to(device).eval()

def clip\_v2\_image\_embed(img: Image.Image, dim=512):

pixel = proc(images=img, return\_tensors="pt").pixel\_values.to(device)

with torch.no\_grad():

emb = model.get\_image\_features(pixel, output\_dim=dim) # Matryoshka magic

return emb[0].cpu().numpy() # shape (dim,)

# now emb.size == 512

Set output\_dim=256 later if RAM pressure rises; no index rebuild—Qdrant lets you recreate\_collection with the new dim and bulk-re-upsert.

## **6  Bottom-line recommendation**

1. **Start with jina-clip-v2 at 512 D.** *Drop-in change*: set DIM = 512 when you create\_collection, call get\_\*\_features(..., output\_dim=512) while ingesting.
2. **Keep openCLIP B/32 baked into a feature flag** so you can A/B test on a held-out set of frame queries; switch via an env var.
3. **Benchmark recall@50 vs. your SSCD labels** after a few million frames.  
    *If recall is already ≥ 95 % while index fits your SSD, stick with jina-clip.* *If resource headroom is tight and recall is “good enough”, the openCLIP path is ready.*

That way you get the best of both worlds—accuracy head-room today, easy rollback to a lighter model tomorrow.  
  
  
**What really changes when you keep the full 1024-D output instead of truncating to 512 D?**

| **Dimension** | **Recall / Quality\*** | **Per-vector size (f32)** | **RAM for HNSW index†** | **SSD size (raw)** | **Search latency** |
| --- | --- | --- | --- | --- | --- |
| **512 D** (now) | baseline (≈97 % of model capacity) | **2 kB** | **≈4 GB / 100 M pts** | ≈300 GB / 1 B pts | base-line |
| **768 D** | +0.5 – 1 pp R@1 | 3 kB | 6 GB / 100 M | 450 GB / 1 B | +5–8 % |
| **1024 D** | **+1 – 3 pp R@1** (best long-tail matches) | **4 kB** | **8 GB / 100 M** | **600 GB / 1 B** | +10–15 % |

\* rough deltas drawn from public COCO & LAION retrieval studies on Matryoshka-trained CLIP variants  
 † HNSW with m=16, ef\_construction=100, no product-quantisation

## **Practical impact in your pipeline**

1. **Search quality** *Most of the semantic signal lives in the first 256–512 dims.* The extra 512 dims help in edge-cases:  
   * fine-grained style & background cues
   * very short text prompts (single-word)
   * near-duplicate frames that differ only in colour grading
2. **Storage & RAM**
   * Vector disk footprint doubles.
   * RAM used by the in-memory HNSW graph doubles.
   * If you enable PQ compression later, the ratio is the same (e.g. 8-bit PQ still halves either size).
3. **Throughput**
   * **GPU encoding speed** is almost identical (projection head runs anyway).
   * **Qdrant search latency** rises 10–15 % because each dot-product is twice as long; you can claw that back by lowering hnsw\_ef.

**Index rebuild** Qdrant collections are *fixed-dim*; you’ll need to:  
  
 python  
CopyEdit  
qd.delete\_collection("video\_frames")

qd.create\_collection("video\_frames",

vectors\_config=models.VectorParams(size=1024,

distance=models.Distance.COSINE))

1. then re-ingest or scroll & update existing points.
2. **Mixed strategy (recommended)** *Keep 1024-D on disk but search on 512-D*:  
   * store full 1024 in payload ("vec1024": …)
   * build the main index on the first 512 dims (fast)
   * for each query: search → take top-200 → rerank with full 1024-D dot-product in RAM.  
      This gives you most of the recall gain for <10 % of the storage/latency overhead.

## **When to switch**

| **Stay at 512 D if…** | **Move to 1024 D if…** |
| --- | --- |
| QPS is a priority; RAM/SSD budget is tight; offline tests already hit ≥95 % recall before SSCD. | You see too many false negatives after the embedding filter; lab tests show ≥2 pp recall gain with 1024; infra budget can absorb 2× storage. |

**Try it on a 1 % shard first**: create video\_frames\_1k at 1024 D, ingest a sample, run the same evaluation notebook. If R@50 or mean-avg-precision improves meaningfully—and latency is acceptable—scale it to the full corpus.

### **TL;DR**

*Going from 512 to 1024 dims buys you ~1–3 pp more recall and better fine-grained matches, at the clear cost of doubling memory, disk, and ~15 % query time.* Most teams keep 512 D for the primary ANN index and use the full vector only for reranking or offline analytics. Evaluate on your ground-truth set before committing.