Embeddings Research

##### **You said:**

I want to put up a video search service that find relevant video results based on the actual video visual content based on embeddings extraction. I’ve made some test with aws titan G1 but I want to go for an open source technology that I will later be able to deploy on any server.

Can you make a benchmark of open source alternative that I can use in production grade environments, for the long run for my service. I want it to be accurate, fast and cost efficient when it comes to computational consumption. Please compare these alternatives, list pros and cons and what dimensions of video are made searchable (elements, action, colors, faces, etc).

Make a sorting with highest recommandation at the top and lowest a the bottom and why

###### **ChatGPT said:**

To create the most relevant benchmark, could you please clarify the following:

1. What types of videos are you targeting (e.g., short-form social, movies, CCTV, etc.)?
2. What are the main use cases for your search (e.g., action detection, object search, aesthetic similarity, etc.)?
3. What’s your target scale (e.g., thousands, millions, or more videos)?
4. Do you require real-time indexing or is batch processing acceptable?
5. Are there specific hardware constraints or platforms you plan to deploy on (e.g., GPU availability, edge devices, Kubernetes)?

This will help tailor the benchmark for your production needs and ensure the results are actionable and cost-efficient.

##### **You said:**

1. Short form social and YouTube long form that include remixing (green screen, Picture in Picture, split screen, using the hook at the beginning of a video)

2. Element detection, action detection, something that allows to filter down results before a vision frame by frame comparison with sscd mixup classy

3. Millions even billions of videos

4. Batch processing and some real time for video to video search or image to video search

5. No specific hardware constraints, but because we deal with millions of videos, must be cost efficient

###### **ChatGPT said:**

Great, I’ll benchmark open-source video understanding and embedding extraction technologies suitable for large-scale visual content search. I’ll prioritize tools that support action and element detection, offer high accuracy and cost-efficiency, and can operate at scale with batch and near-real-time capabilities. I’ll compare each on accuracy, speed, computational cost, ease of deployment, supported features (e.g., actions, objects, colors), and long-term production viability.

I’ll get back to you with a ranked list including pros and cons for each alternative.

# **Benchmark of Open‑Source Video Embedding Techniques**

## **Introduction**

Building a production-grade visual video search service requires robust video embedding techniques and scalable retrieval infrastructure. Such a system must handle diverse content – from short-form social clips to long-form YouTube videos – including derivative media (e.g. green-screen overlays, picture-in-picture, remixes). The solution should support multiple search dimensions (e.g. actions, objects, colors, faces, scene layout) and allow both batch indexing at scale and real-time querying (video-to-video or image-to-video matching). Key considerations include accuracy of the embeddings (how well they capture visual similarity and relevant content), support for specific content filters (actions, objects, etc.), speed of feature extraction and search, computational cost (hardware/memory efficiency), ease of deployment/maintenance, and the strength of the open-source community behind each solution. Below, we compare the top open-source approaches for video embedding and search on these criteria. We then provide a ranked list (most to least recommended) with a summary of each solution’s pros, cons, and our justification for its ranking. A summary table and conclusion follow.

## **Evaluation Criteria**

Before diving into the alternatives, it’s important to clarify the criteria used for evaluation:

* Accuracy & Relevance of Embeddings: How well the method captures visual content such that truly similar videos (or frames) are close in embedding space. This includes robustness to transformations (edits, overlays) and semantic relevance (grouping related content).
* Searchable Dimensions (Actions, Objects, etc.): The ability to detect or encode specific facets like human actions, prominent objects/elements, dominant colors or scenes, faces, or layout. Solutions may explicitly provide these as separate features or implicitly encode them in the embedding.
* Speed of Processing: The computational efficiency of the pipeline – both inference speed (generating embeddings for frames/videos) and indexing/query speed (similarity search throughput). This is critical for scaling to millions of videos and supporting near real-time queries.
* Computational/Hardware Cost: Resource requirements to run the model at scale – e.g. GPU memory and FLOPs for inference, storage and RAM for indexes. More efficient models or indexing techniques reduce operational cost.
* Ease of Deployment & Maintainability: How straightforward it is to integrate and deploy the solution on-premise or on any servers (avoiding lock-in to a specific cloud). This includes availability of Docker images or libraries, code simplicity, and long-term support (bug fixes, updates).
* Community Support & Development Activity: The vibrancy of the open-source community – frequent updates, documentation, forums – which affects longevity and availability of help or improvements.

We apply these criteria to each solution below.

## **1. Towhee Pipeline + Milvus (Integrated Video Search Stack)**

Overview: *Towhee* is an open-source AI pipeline framework, and *Milvus* is an open-source vector database. Together, they provide a ready-made pipeline for video similarity search. Towhee offers built-in pipelines for video embedding and copy detection, which by default use a specialized model (“ISC”) geared towards video copy-detection

[towhee.io](https://towhee.io/tasks/detail/pipeline/video-copy-detection#:~:text=)

. Milvus handles large-scale vector indexing and similarity search efficiently. This combination is a complete end-to-end solution for content-based video retrieval, requiring minimal custom code. Pros:

* Complete Solution & Accuracy: Provides an end-to-end pipeline implementing best practices for video search. For example, the Towhee pipeline converts video frames to embeddings, stores them in Milvus, and applies a temporal consistency check to detect copied segments
* [towhee.io](https://towhee.io/tasks/detail/pipeline/video-copy-detection#:~:text=,its%20similar%20videos%27%20embedding%20loaded)
* [towhee.io](https://towhee.io/tasks/detail/pipeline/video-copy-detection#:~:text=,similar%20segments%20and%20similar%20scores)
* . The default embedding model (ISC) is specifically tuned for copy-detection, achieving high accuracy in catching near-duplicates or re-edited content. It produces compact 256-D embeddings designed for robust video fingerprinting (resistant to crops, overlays, etc.)
* [towhee.io](https://towhee.io/tasks/detail/pipeline/video-copy-detection#:~:text=)
* . This yields excellent relevance for finding the same or visually similar content across derivative videos.
* Multi-Modal Extensions: Towhee’s framework makes it easy to incorporate additional models for other searchable dimensions. It has operators for tasks like object detection and face recognition
* [towhee.io](https://towhee.io/tasks/operator#:~:text=Towhee%20,1%20models)
* , which can be added to the pipeline. This means actions or objects can be detected per video and used as filters or metadata in the search index (e.g. index videos by detected action labels or prominent objects alongside the embeddings). The integrated approach simplifies combining these signals, improving the relevance of search results for complex queries (e.g. filter by “contains a face and car” before similarity matching).
* Scalability & Speed: Milvus is built for billion-scale vector search with distributed indexing
* [researchgate.net](https://www.researchgate.net/post/Where-can-I-find-a-properly-arranged-database-for-content-based-video-retrieval#:~:text=Where%20can%20I%20find%20a,some%20benchmark%20database%20in)
* [researchgate.net](https://www.researchgate.net/post/Where-can-I-find-a-properly-arranged-database-for-content-based-video-retrieval#:~:text=I%20am%20working%20on%20content,some%20benchmark%20database%20in)
* . It supports efficient ANN (Approximate Nearest Neighbor) search with indexes like IVF and HNSW, enabling millisecond-level retrieval even with millions of video embeddings. The Towhee pipeline is designed to batch-process and index large video collections (it can stream data through Spark or in parallel). Thus, both offline batch indexing and online queries can scale horizontally. The retrieval step uses inner-product or Euclidean similarity in Milvus, which is optimized in C++ for speed.
* Deployment & Maintainability: The solution is cloud-agnostic and can be deployed on any VM or on-premise machine. Milvus offers Docker containers and is maintained by an active community (the Linux Foundation’s Milvus project). Towhee pipelines can be run in Python environments with minimal code (just load the video\_embedding pipeline and feed videos)
* [towhee.io](https://towhee.io/tasks/detail/pipeline/video-copy-detection#:~:text=We%20can%20use%20built,args%20to%20the%20pipeline%20config)
* . This high-level pipeline abstraction reduces maintenance – updates to models or indexing logic can come through Towhee updates without the user rewriting code. Both Milvus and Towhee are under active development by Zilliz (with frequent releases), ensuring long-term support.
* Community & Documentation: Milvus has a large user community and extensive documentation, since it’s a popular vector DB for AI search
* [researchgate.net](https://www.researchgate.net/post/Where-can-I-find-a-properly-arranged-database-for-content-based-video-retrieval#:~:text=Where%20can%20I%20find%20a,some%20benchmark%20database%20in)
* . Towhee, while newer, is backed by the same company and has growing community support. There are examples and guides (including those for video search and even copy-detection competitions) that users can leverage.

Cons:

* Moderate Complexity & Overhead: The integrated stack introduces multiple components (Towhee runtime, Milvus server, optional key-value store for metadata), which can be heavier to manage than a single-library solution. Operating Milvus at scale requires some database management know-how (e.g. configuring index types, managing cluster nodes). This added complexity might be overkill for smaller applications.
* Tailored for Copy-Detection by Default: The default ISC/SSCD embedding in Towhee is optimized for *duplicate detection* rather than broad semantic similarity. While extremely effective for finding copies, it might not distinguish high-level categories as well as a general-purpose model. If the use-case involves more semantic or “related content” search (not exact copies), one might prefer to swap in a different embedding model (Towhee does allow this). Using a different model (e.g. CLIP) in Towhee is possible but requires manual pipeline config changes.
* Action/Element Support Not Automatic: Although Towhee can incorporate other detectors, out-of-the-box the pipeline doesn’t automatically include action or object tagging – it focuses on visual similarity. Implementing the full multi-model ensemble (objects, faces, etc.) still requires additional configuration or custom pipelines. This flexibility is powerful, but it means the user must know which models to add and how to use the results (though Towhee provides operator libraries, it’s not a one-click solution for multi-dimensional search).
* Resource Consumption: Running the full pipeline at scale will require substantial compute. Feature extraction (ISC model or others) on many frames is GPU-intensive (though the model is fairly lightweight, you will still potentially process many frames per video). Milvus, while efficient, uses RAM for indexing vectors and will consume storage for large datasets. Ensuring low query latency at very large scale might necessitate multiple Milvus shards and replicas. Thus, cost-efficiency is good (since components are optimized in C/C++), but the absolute resource use is still significant for millions of videos.
* Smaller Community than Some Alternatives: Towhee is relatively new and niche compared to generic frameworks like PyTorch or to widely-used models like CLIP. While Milvus is well-established, Towhee’s user base is still growing. This means for very custom modifications or debugging, community help might be less abundant (though official support from maintainers is available).

Ranking Justification: We rank the Towhee+Milvus pipeline #1 because it provides a production-ready, scalable framework that directly addresses the project requirements. Its use of a state-of-the-art copy-detection embedding ensures high accuracy on the challenging derivative content scenario (e.g. finding green-screen or remixed copies), and the entire pipeline is open-source. The ease of deployment (pre-built pipeline) and the ability to extend it for actions/objects detection make it a strong foundation. While it has some complexity, it offloads much of the heavy lifting to well-maintained components (vector DB, etc.), which is a worthy trade-off for large-scale systems. Overall, it offers the best balance of accuracy, scalability, and deployability for a visual video search service.

## **2. Self-Supervised Copy-Detection Models (SSCD & Variants)**

Overview: *Self-Supervised Descriptor for Copy Detection (SSCD)* is an open-source model from Facebook (Meta) Research for robust image/video copy detection

[github.com](https://github.com/facebookresearch/sscd-copy-detection#:~:text=A%20Self,SSCD)

. It was originally trained on images with heavy augmentations to act as a “fingerprint” descriptor for identifying duplicate or derived content

[github.com](https://github.com/facebookresearch/sscd-copy-detection#:~:text=This%20is%20the%20open,recently%20accepted%20to%20CVPR%202022)

. SSCD uses a ResNet-50 backbone and produces a 512-dimensional embedding per frame

[arxiv.org](https://arxiv.org/pdf/2306.09489#:~:text=SSCD%20descriptor,DISC21%20dataset%20with%20full%20augmentations)

. This model (and its improved variants) represents a top-tier approach for *frame-level video embeddings* aimed at finding overlapping content between videos. It served as the baseline in Meta’s 2023 Video Similarity Challenge (VSC), and many top teams built on it

[arxiv.org](https://arxiv.org/pdf/2306.09489#:~:text=for%20the%20baseline%20methods%20provided,video%20transforma%02tions%3A%20LINE%20CVL%20and)

[arxiv.org](https://arxiv.org/pdf/2306.09489#:~:text=6,supervision%20as%20in%20SSCD%20%5B33)

. Pros:

* High Accuracy for Duplicate Detection: SSCD and similar self-supervised models excel at fine-grained visual matching. They were specifically designed to detect if two images (or video frames) are derived from the same source, even under transformations
* [github.com](https://github.com/facebookresearch/sscd-copy-detection#:~:text=This%20is%20the%20open,recently%20accepted%20to%20CVPR%202022)
* . In the CVPR’23 benchmark, the SSCD-based baseline achieved a strong 60.5% µAP on the video copy detection task
* [arxiv.org](https://arxiv.org/pdf/2306.09489#:~:text=match%20at%20L752%20baseline%2060.5,from%20the%20temporal%20similarity%20graph)
* , and top challenge submissions further improved on this by using SSCD-style training on transformer backbones
* [arxiv.org](https://arxiv.org/pdf/2306.09489#:~:text=6,supervision%20as%20in%20SSCD%20%5B33)
* . This means the model is very effective at catching the types of remixed or edited video content described (overlays, crops, re-coloring, picture-in-picture) – far more robust to these distortions than standard image embeddings.
* Frame-Level Granularity: SSCD yields an embedding for each frame. This is useful to localize where a copy occurs and to handle partial matches (e.g. one video contains a short segment from another). By indexing frames (or short segments) individually, a search can pinpoint copied scenes with high precision. The approach of frame-level matching followed by temporal alignment is a proven strategy in video copy detection
* [towhee.io](https://towhee.io/tasks/detail/pipeline/video-copy-detection#:~:text=,its%20similar%20videos%27%20embedding%20loaded)
* [towhee.io](https://towhee.io/tasks/detail/pipeline/video-copy-detection#:~:text=,similar%20segments%20and%20similar%20scores)
* . SSCD provides the descriptors needed for this frame matching step.
* Efficiency and Size: Using a ResNet-50 base, SSCD is relatively lightweight. Inference on a single frame is on the order of tens of milliseconds on a GPU (roughly ~10ms per 224×224 image on modern GPUs, similar to ImageNet models). The descriptors are only 512 floats, which is modest in memory per frame. This means one can feasibly compute and store frame embeddings for a large collection (with appropriate ANN indexing) without an extreme memory penalty. If storage is a concern, dimensionality reduction (e.g. PCA) can compress embeddings further with minor loss in accuracy – some challenge teams concatenated and PCA-compressed descriptors
* [arxiv.org](https://arxiv.org/pdf/2306.09489#:~:text=for%20the%20baseline%20methods%20provided,video%20transforma%02tions%3A%20LINE%20CVL%20and)
* . The open-source code even provides TorchScript models
* [github.com](https://github.com/facebookresearch/sscd-copy-detection#:~:text=For%20convenience%2C%20we%20provide%20equivalent,files%20in%20a%20few%20formats)
* , making it easy to deploy efficiently in C++ or other environments without heavy Python overhead.
* Easy Deployment & Integration: The SSCD model weights and code are available under MIT license
* [github.com](https://github.com/facebookresearch/sscd-copy-detection#:~:text=)
* . Integrating it is straightforward with PyTorch – you can load the pretrained model and get frame embeddings with a few lines of code. It does not require specialized hardware beyond a standard GPU. Since it’s an image-model at heart, it can run on any inference server that supports CNNs (could even run on CPU for smaller scale). Its self-supervised nature means it was trained without proprietary data; the provided weights can be used freely, and one could fine-tune on specific video data if needed.
* Active but Focused Community: Being part of a recent research (CVPR 2022/2023), there is interest and knowledge about SSCD in the computer vision community. The VSC workshop paper indicates continued analysis of such models
* [arxiv.org](https://arxiv.org/pdf/2306.09489#:~:text=6,supervision%20as%20in%20SSCD%20%5B33)
* , and Meta released it specifically to encourage use-cases like copyright protection, moderation, etc. While not as broad as general vision models, the niche community (researchers and engineers working on copy detection) is quite active. Issues and discussions on the GitHub repo can provide guidance, and the methods are documented in academic papers
* [github.com](https://github.com/facebookresearch/sscd-copy-detection#:~:text=A%20Self,SSCD)
* .

Cons:

* Limited Semantic Understanding: SSCD’s strength is in low-level visual matching; it is not designed to understand high-level concepts or actions. Two videos of the same object or scene will be close in SSCD space, but two different videos both containing, say, dogs might *not* cluster unless the dogs and context look very similar. In other words, SSCD embeddings are excellent for “is this the same video or a modified version?” but not for “find all videos of X”. If the search requirement includes broad semantic similarity (e.g. find videos of a *category* or *action* rather than the exact same content), SSCD alone is less ideal. It doesn’t explicitly encode object categories, and might treat different scenes of the same action as far apart if the visual details differ. For those use-cases, a model like CLIP or an action-recognition model would perform better (meaning SSCD may need to be supplemented by other features).
* Framewise Indexing Overhead: Because it works at the frame level, using SSCD for large video collections can be resource-intensive in storage and search. A single 10-minute video at 1 fps sampling produces 600 frame vectors. For 1 million videos, that’s hundreds of millions of vectors to index/search. This can bloat index size and slow down search if not carefully managed (requiring techniques like hierarchical search or pruning). Some production systems mitigate this by first using cheaper global features to shortlist candidates. Without such measures, a pure SSCD approach might struggle with scalability to billions of frames – the brute-force solution is expensive (though approximate indexing and distributed search can address this).
* Narrow Community & Updates: Outside of the copy-detection/copyright domain, SSCD is not as widely adopted as generic models. This means fewer out-of-the-box tools and less forum discussion. The model is relatively “frozen” (no major new versions since CVPR 2022, aside from other teams’ experiments). If future improvements occur (e.g. a transformer-based copy-detection model as used by challenge winners), one might have to implement those from papers as they’re not yet in an off-the-shelf library. In contrast, more general models (CLIP, etc.) have numerous open variants and continuous improvement from the community.
* No Built-in Multi-Dimensional Features: SSCD by itself does not detect actions, objects, or faces in an interpretable way – it just gives an embedding. If you need actionable metadata like “which objects were detected in this video?”, SSCD offers no direct mechanism. You would still need to run a separate object detector or action classifier to get those labels. Thus, while SSCD can tell you two videos are similar, it won’t tell you *why* (no semantic labels). This can make it harder to integrate as part of a filtering mechanism without additional models in the pipeline.

Ranking Justification: We rank SSCD (and similar self-supervised copy-detection models) #2 for this use-case. For a service that explicitly needs to handle heavily edited copies and perform fine-grained visual matching, this approach is extremely powerful – likely more reliable than general-purpose embeddings in catching derivative content. Its high accuracy on the critical task of *copy localization*

[arxiv.org](https://arxiv.org/pdf/2306.09489#:~:text=6,supervision%20as%20in%20SSCD%20%5B33)

aligns well with the requirement of finding remixed/overlaid segments. Furthermore, it’s relatively easy to deploy and efficient given its robustness. We place it above more semantic-oriented models like CLIP because the challenge of near-duplicate video detection is front-and-center in the requirements, and SSCD was purpose-built for that. However, it sits below the Towhee+Milvus pipeline because, on its own, SSCD is a building block rather than a full scalable system – one would still need to build out the indexing, filtering, and multi-modal support. In fact, the Towhee pipeline leverages this very type of model under the hood. For someone willing to assemble a custom pipeline, SSCD provides the best-of-breed embedding for copy similarity; it’s our top recommendation if your focus is on *visual fidelity matching* above all else.

## **3. OpenCLIP / CLIP4Clip (CLIP-Based Visual Embeddings)**

Overview: *OpenAI’s CLIP* model (and the open-source OpenCLIP implementations) introduced a powerful way to embed images into a semantic vector space by training on 400 million image-text pairs

[anttihavanko.medium.com](https://anttihavanko.medium.com/building-image-search-with-openai-clip-5a1deaa7a6e2#:~:text=OpenAI%E2%80%99s%20Clip%20is%20a%20neural,a%20given%20text%20or%20image)

. Although CLIP is not inherently a video model, it has been widely adopted for visual similarity tasks. Using CLIP, one can encode video frames or keyframes and obtain embeddings that capture high-level content and context of the imagery. Extensions like CLIP4Clip train a video encoder to align with CLIP’s text/image space, producing a single embedding for an entire video

[towhee.io](https://towhee.io/video-text-embedding/clip4clip#:~:text=author%3A%20Chen%20Zhang)

[towhee.io](https://towhee.io/video-text-embedding/clip4clip#:~:text=Load%20an%20video%20from%20path,to%20generate%20an%20video%20embedding)

. This family of models is geared more towards semantic content understanding than exact pixel-level matching. It enables not only video-to-video similarity, but also text-to-video search (if needed), which adds versatility. Pros:

* Strong Semantic Relevance: CLIP embeddings excel at capturing the high-level concepts in an image (or frame). The model learned associations between images and natural language, so its vision encoder develops a rich representation of objects, their relationships, and context. For example, CLIP can embed an image such that all “beach sunset” scenes are near each other, regardless of specific pixel differences, because it has learned the concept of a beach sunset. This means using CLIP on video frames will group videos by visual themes and subjects. For a search service, this yields very relevant results for queries like “find videos of cats” or “a person cooking” – scenarios where we care about semantic similarity, not just exact duplicates. This complements copy-detection models well. In practice, many systems sample a few frames per video and average CLIP embeddings to get a video-level descriptor that is remarkably good at retrieving videos of similar content or genre.
* Multi-Modal Search Capability: A unique advantage of CLIP-based approaches is the ability to perform text-to-video search out of the box. Since CLIP embeds text and images in a joint space
* [anttihavanko.medium.com](https://anttihavanko.medium.com/building-image-search-with-openai-clip-5a1deaa7a6e2#:~:text=OpenAI%E2%80%99s%20Clip%20is%20a%20neural,a%20given%20text%20or%20image)
* , one can index videos by their frame embeddings and then query the index using an embedding of a text description. This enables semantic search (“find videos of a dog playing guitar”) without any manual tagging – the model itself has learned these associations. While our focus is video-to-video, having this capability can greatly enhance the service (users could find videos by keywords without manual metadata). Among open solutions, CLIP is unparalleled in this cross-modal aspect, and OpenCLIP has made a variety of pretrained weights freely available for use
* [huggingface.co](https://huggingface.co/docs/hub/en/open_clip#:~:text=OpenCLIP%20is%20an%20open,of%20OpenAI%E2%80%99s%20CLIP)
* .
* Fast Inference and Indexing: CLIP’s vision encoder (especially the ViT-B/32 or ViT-B/16 models commonly used) is reasonably fast. For instance, encoding an image with CLIP ViT-B/32 takes on the order of ~20ms on a V100 GPU
* [anttihavanko.medium.com](https://anttihavanko.medium.com/building-image-search-with-openai-clip-5a1deaa7a6e2#:~:text=Encoding%20a%20single%20image%20takes,the%20data%20to%20the%20GPU)
* , meaning ~50 images per second per GPU, and larger models like ViT-L/14 still achieve ~10+ images per second. This speed makes it feasible to embed enough frames for each video or even do on-the-fly query frame embedding in real-time. For indexing, because CLIP typically produces one vector per image or per video (if aggregated), the number of vectors is far fewer than frame-by-frame methods. If each video is represented by a single 512-D CLIP vector (e.g. via CLIP4Clip sampling 12 frames
* [towhee.io](https://towhee.io/video-text-embedding/clip4clip#:~:text=pipe.input%28%27video_path%27%29%20,.output%28%27video_path%27%2C%20%27flame_list%27%2C%20%27vec)
* ), one can index millions of videos using standard ANN libraries (Faiss, Annoy, etc.) quite efficiently. Memory footprint is low – 1 million CLIP vectors (~512 floats) is only ~2GB of data. Searches can be very fast (sub-100ms) even on CPU with proper ANN indexing
* [anttihavanko.medium.com](https://anttihavanko.medium.com/building-image-search-with-openai-clip-5a1deaa7a6e2#:~:text=to%20index%20them,many%20different%20functionalities%2C%20such%20as)
* . This makes CLIP-based embeddings cost-efficient for scale – you get a compact representation per video and rapid search times with minimal hardware.
* Ease of Deployment & Community: OpenCLIP (by LAION/ML Foundry) and CLIP models are widely used and easy to integrate. There are ready Python libraries (e.g. sentence-transformers or Hugging Face Transformers) that let you load a pretrained CLIP model in one line
* [anttihavanko.medium.com](https://anttihavanko.medium.com/creating-a-semantic-video-search-with-openais-clip-model-13ff14990fbd#:~:text=To%20create%20embeddings%20for%20the,interacting%20with%20the%20Clip%20model)
* [anttihavanko.medium.com](https://anttihavanko.medium.com/creating-a-semantic-video-search-with-openais-clip-model-13ff14990fbd#:~:text=match%20at%20L145%20def%20get_embeddings,for%20vector%20in%20vectors)
* . No special infrastructure is needed beyond a standard PyTorch environment. Many variations are available (different sizes, architectures) allowing choice between accuracy and speed. The community is large and active – CLIP has become a foundation model for vision-language tasks, with continuous improvements (e.g. LAION releasing new OpenCLIP checkpoints, or research like CLIP2Video, etc.). This means plenty of documentation, examples, and user-contributed tips for using CLIP in image/video search. Any issues encountered are likely to have been discussed on forums like GitHub or Reddit. Long-term maintainability is excellent: CLIP is now a staple model in many open-source AI ecosystems, so support will persist and new compatible models will emerge.
* Adaptability: Beyond just using CLIP as-is, its embeddings can be fine-tuned or adapted to specific domains if needed. There are known techniques to fine-tune CLIP on video datasets (while retaining its semantic knowledge) or to combine CLIP features with other modalities (e.g. audio) for more robust video embeddings. The open-source nature and popularity of CLIP means such extensions (like combining CLIP with optical flow for action sensitivity) might already exist in research, giving a path to further improve accuracy on video tasks if necessary.

Cons:

* Lower Robustness to Minor Edits: Because CLIP is trained on *overall semantic content*, it is less sensitive to subtle duplicates. For example, if a video scene is re-uploaded with a filter or slight crop, CLIP may not treat it as significantly closer than any other scene of the same general description. CLIP might focus on the conceptual similarity (e.g. “a person in front of a building”) rather than exact pixel alignment. In contrast to SSCD, which would almost certainly link the filtered copy to the original, CLIP could rank it lower if color or style changed (since style can affect the embedding). In short, CLIP’s strength is also its weakness for this task: it abstracts away details that might be crucial for identifying a derivative copy. For the specific need of catching green-screen or picture-in-picture reuses, CLIP could miss matches that a dedicated copy-detection model would catch, as it wasn’t trained specifically for that invariance.
* No Temporal Awareness: CLIP processes each frame independently (unless using a video extension like Clip4Clip, which still essentially pools frame features without modeling motion explicitly). It does *not* encode motion or temporal patterns. Thus, an action that is only distinguishable by motion (e.g. walking vs running) may not be differentiated in CLIP embeddings if individual frames look similar. Two videos might both contain a person, and CLIP might embed them as similar even if one person is dancing and the other is just standing, because the static visual elements overlap. For action-specific queries, CLIP is not the ideal choice; specialized video models or action detectors outperform it. This also means CLIP won’t inherently filter by dynamic events – you’d need an auxiliary method to handle that dimension (which reduces its one-model simplicity slightly if those are requirements).
* Lengthy Videos and Frame Sampling: Representing a long video with CLIP requires choosing a subset of frames (since encoding every frame is impractical and unnecessary). There is a risk that important content is missed if the sampling is sparse. For example, a one-hour video might need keyframe selection; CLIP won’t know about parts of the video you didn’t show it. Other methods might use learned pooling or segment representations to better cover a long video. Clip4Clip partially addresses this by uniform sampling (e.g. 12 frames)
* [towhee.io](https://towhee.io/video-text-embedding/clip4clip#:~:text=pipe.input%28%27video_path%27%29%20,.output%28%27video_path%27%2C%20%27flame_list%27%2C%20%27vec)
* , but depending on the video, 12 frames may not capture all distinct scenes or actions. Increasing the number of frames improves recall but also increases compute and index entries. There’s a trade-off here, and tuning it might require some experimentation. Essentially, using CLIP for videos demands a frame sampling strategy, adding a bit of complexity (whereas video-specific models or video pipelines might handle this inherently by temporal modeling or shot detection).
* Computational Cost for Large-Scale Inference: While CLIP is fast per frame, if one were to use it *in addition* to an existing pipeline (e.g. alongside SSCD or others), that is extra GPU workload. Running two large models doubles the compute cost. If CLIP is the sole model, the cost is moderate, but in a multi-model scenario the overhead could be significant. Also, the largest CLIP models (like ViT-L/14 or ViT-H/14 from OpenCLIP) are quite heavy (hundreds of millions of parameters). Using a smaller CLIP might sacrifice some accuracy for fine-grained details. So there is a balancing act in deployment: one might opt for a medium-sized CLIP for speed, which is generally fine, but it could miss some nuance a bigger model would catch.
* Potential Licensing for Pretrained Weights: OpenCLIP weights trained by LAION are licensed permissively, but OpenAI’s original CLIP weights have some usage restrictions (they are provided for research, not commercial use by default). However, this is a minor con since truly open alternatives exist (OpenCLIP, or CLIP checkpoints on Hugging Face with friendly licenses). It just requires a bit of attention to which model variant is chosen to ensure it’s free for your intended use.

Ranking Justification: We rank CLIP-based embeddings #3. They offer *excellent semantic search capabilities* and a well-proven, efficient pipeline for general visual similarity. In a video search service where users might look for thematically similar content (not just exact copies), CLIP provides more relevance than a purely copy-detection approach. Its huge community and ease of use are big pluses for long-term maintainability. We place it below SSCD because the specific requirement of handling heavily edited copies (green-screen, etc.) tilts the scales toward models purpose-built for that – CLIP can miss those fine details. However, it outranks the remaining solutions due to its strong balance of accuracy, speed, and community support. In practice, one might use CLIP *alongside* a copy-detector to get the best of both worlds. But as a standalone technology, CLIP is a top-tier open solution for video/image embedding and would be our go-to for semantic similarity and flexible search features.

## **4. Video Transformers (VideoMAE, TimeSformer, etc.)**

Overview: Video Transformer models like TimeSformer (Time-Space Transformer), ViViT, and VideoMAE represent the state-of-the-art in deep video understanding. These models extend the idea of Vision Transformers to the spatio-temporal domain, processing sequences of frames with self-attention. For instance, VideoMAE (Video Masked Autoencoder) is a self-supervised transformer pre-trained on video data; it has achieved state-of-the-art performance on multiple video classification benchmarks

[huggingface.co](https://huggingface.co/docs/transformers/en/model_doc/videomae#:~:text=%28MAE%29%20to%20video%2C%20claiming%20state,on%20several%20video%20classification%20benchmarks)

. Such models output embeddings that encapsulate both appearance and motion, making them very powerful for recognizing actions and complex events. They are typically open-sourced via academic code (often PyTorch or TensorFlow) and some have pretrained weights available on Hugging Face (e.g. VideoMAE, TimeSformer). In a retrieval context, one could use these models to extract a single embedding for a video or a segment, which would then be indexed for similarity search. Pros:

* Captures Temporal Dynamics (Actions): Unlike image-based methods, video transformers explicitly model the temporal dimension. They process a clip (e.g. 8, 16, or 32 frames) as input to produce an embedding. This means the resulting representation encodes motion patterns and actions – for example, distinguishing “walking” vs “running” or recognizing a “jump” action that spans multiple frames. This directly addresses the “actions” dimension of search. If a user wants to filter or search by an action, these models inherently provide an embedding that changes when the action changes. They have been shown to excel at action recognition tasks (e.g., VideoMAE tops many action classification leaderboards
* [huggingface.co](https://huggingface.co/docs/transformers/en/model_doc/videomae#:~:text=%28MAE%29%20to%20video%2C%20claiming%20state,on%20several%20video%20classification%20benchmarks)
* ). So, for content where the *dynamic* aspect is crucial (sports highlights, dance moves, etc.), video transformers provide much better features than static frame models.
* Rich, High-Dimensional Embeddings: The representations from these models are typically high-dimensional and very descriptive. They effectively summarize a chunk of video, including the scene, objects, and the motion occurring. For instance, a VideoMAE model might take a 16-frame clip and produce a feature vector that encodes the presence of certain objects (learned via attention to patches) *and* the fact that those objects are moving in a certain way. This can contribute to high retrieval accuracy for both semantic and copy-similarity queries: if two videos show the same sequence of actions and scenes, a good video transformer will place them close in feature space. Moreover, some models can be fine-tuned to directly optimize retrieval metrics. Even off-the-shelf, their classification performance implies strong discriminative power.
* Action and Context Integration: Because these models see multiple frames together, they handle cases like a static scene with different actions or identical actions in different scenes better than separate frame analysis. They implicitly combine context (what is in the scene) with the action (how it changes). For example, if searching for “a person throws a ball and then dances,” a sufficiently powerful video model could, in principle, encode that sequence, whereas frame-based methods would need to somehow chain separate detections. This holistic encoding of events can make certain fine-grained searches possible (if query is by example video or if one indexes sub-sequences of videos).
* Continual Improvement and Research Support: Video transformers are an active area of research. New architectures and pre-training methods (like VideoMAE V2, Masked Video modeling, etc.) are pushing the boundaries, often with open-source releases. The community, though mostly academic, frequently releases code and pretrained weights (as evidenced by VideoMAE’s code and models on GitHub/HuggingFace, TimeSformer by Facebook, etc.). This means if you adopt one of these models, you can expect improvements or variations to be available, and you can fine-tune these models on your own data to specialize them. The methods are in the open literature, so vendor lock-in is not a concern. For a long-term system, this is advantageous as you can swap in improved versions as they become available to boost accuracy.

Cons:

* Heavy Computational Cost: These video transformer models are *extremely resource-intensive*. A typical model might process 8–32 frames of size 224×224 with dozens of transformer layers. For example, VideoMAE base requires on the order of 7.4 – 8.3 TFLOPs per inference (per single video clip)
* [drone-vis.readthedocs.io](https://drone-vis.readthedocs.io/en/latest/action_recognition/video_masked_transformer.html#:~:text=Video%20Masked%20Autoencoders%20%28VideoMAE%29%20,)
* – roughly 2–3 orders of magnitude more than a single ResNet or CLIP forward pass. In practical terms, if CLIP can do ~50 images/sec on a GPU, a video transformer might only do a handful of clips per second on the same hardware. This poses a serious scalability challenge: indexing millions of videos with such a model could be prohibitively slow or require a large GPU cluster. Even at query time, using these models for real-time queries (video-to-video matching on the fly) may not be feasible unless the videos are very short or one has powerful accelerators. The high computational demand also means higher cloud costs or electricity usage, making them *less cost-efficient* for large-scale deployment.
* Memory and Storage Footprint: The embeddings from video models might be larger (some use 768 or 1024 dimensions for the feature vector). More significantly, if one chooses to index sub-segments of videos (to allow matching of a portion of a video), you might end up with multiple embeddings per video (e.g., one per every N-second clip). This can balloon the number of vectors, albeit not as badly as frame-level indexing but still more than one per video. Additionally, the model weights themselves are large (hundreds of MBs) and running them with long sequences can demand a lot of GPU memory (such models often require 16GB+ GPUs for inference). There’s also the engineering overhead of deploying such a heavy model – it might need model parallelism or careful optimization for throughput, which complicates deployment.
* Latency Issues for Real-Time: If your service needs *near real-time* search (e.g. a user uploads a clip and expects results in seconds), video transformers could be a bottleneck. Unless you precompute embeddings for the entire corpus (which you likely would), the query still needs to be encoded. Encoding even a 5-second query clip with a large transformer could take a second or more on GPU, which might be borderline for user experience if additional search time is included. In contrast, faster models (CLIP or SSCD) would encode that in a tiny fraction of a second. So the speed disadvantage might make these models unsuitable for interactive or high-QPS (queries per second) scenarios. They are better suited for *batch offline analysis* given current hardware.
* Implementation Complexity: Deploying these models is more complex than using a well-packaged image model. While libraries like Hugging Face Transformers are adding support (they have TimeSformer, VideoMAE classes, etc.), using them still requires careful preprocessing (extracting frames or decoding video into the model’s expected format) and possibly postprocessing (aggregating clip-level results). Not all video models are as plug-and-play; some might come from research code that isn’t optimized or documented for production use. Troubleshooting performance or memory issues can be non-trivial. Also, some of these models may not have been tested at the scale of millions of videos in a real system, so you may encounter unknown challenges (for example, how to best sample clips from each video to cover content – which is akin to the frame sampling issue, but now with expensive clips).
* Less Community Adoption in Industry: The cutting-edge nature means fewer companies or open-source projects have adopted these for actual video search deployments (as opposed to academic benchmarks). Community support is largely in the form of research papers and some reference implementations, rather than blog posts or Stack Overflow answers for deployment problems. If an issue arises (say, optimizing inference), you might have to rely on general knowledge of transformers or open an issue on the research repo and hope for a response. The support timeline is not as guaranteed as something like OpenCLIP which has many maintainers and users.

Ranking Justification: We rank video transformer models #4. There’s no doubt about their accuracy potential – they likely provide the richest embeddings and would shine in scenarios requiring detailed understanding of actions and complex scenes. However, their practicality in a production-grade service is questionable when weighed against cost and complexity. We place them below CLIP and SSCD because those offer *much better efficiency and ease for a relatively small trade-off in accuracy*. Essentially, a combination of lighter models can often cover similar ground (e.g. an object detector + CLIP for context + an action classifier) at a fraction of the compute cost of a giant unified model. For an application that needs to scale to billions of videos, the transformers are currently a stretch unless significant infrastructure is dedicated. They outrank the remaining solutions (like classical or full frameworks) in our list because they do meet the brief on action and element detection inherently and are at the cutting edge of what’s open-source. In a limited-scale or highly specialized setting, a VideoMAE or TimeSformer could be the top choice; but for broad use, we recommend them as an *optional addition* when specific complex video understanding is required, rather than the first-line approach.

## **5. Vitrivr (Cineast Multimedia Retrieval System)**

Overview: *Vitrivr* is an open-source full-stack content-based multimedia retrieval system (developed by researchers in Switzerland)

[github.com](https://github.com/vitrivr#:~:text=vitrivr)

. At its core is the Cineast engine, which extracts and indexes multiple types of features from videos (and other media)

[github.com](https://github.com/vitrivr#:~:text=match%20at%20L353%20Cineast%20is,textual%20descriptions%20and%20example%20objects)

. It uses a variety of descriptors – from simple ones like color histograms and edge sketches to deep learning features – to enable searching by example image, by sketch, by text, and more. Vitrivr comes with its own purpose-built vector database (Cottontail DB) for similarity search

[github.com](https://github.com/vitrivr#:~:text=1)

and a web UI. Essentially, it’s a comprehensive system aimed at academic and archival video search use-cases, providing many query modalities out-of-the-box. Pros:

* Multi-Feature, Multi-Modal Search: Vitrivr/Cineast’s biggest strength is that it doesn’t rely on a single embedding or model – it takes a *multi-feature fusion approach*. It can index videos by color layout, by motion patterns (via classical features), by audio (if enabled), by text (OCR or metadata), and by deep visual embeddings. It allows queries in the same vein: a user can sketch a shape or provide an example image, and the system will retrieve visually similar content
* [github.com](https://github.com/vitrivr#:~:text=match%20at%20L353%20Cineast%20is,textual%20descriptions%20and%20example%20objects)
* . It also supports semantic queries via concept detectors and text descriptions
* [github.com](https://github.com/vitrivr#:~:text=match%20at%20L353%20Cineast%20is,textual%20descriptions%20and%20example%20objects)
* . For example, it could detect “car” or “explosion” using trained classifiers and use those as search cues. This breadth of searchable dimensions is directly aligned with the requirement to support objects, colors, faces, layouts, etc. Rather than a monolithic embedding, Vitrivr maintains separate indices for different facets (color, motion, deep features, etc.) and can combine them for better results. This means it can do things like filter by “mostly red color” or “contains a face” because it explicitly extracts those attributes. Few open systems offer this level of out-of-the-box multi-modal search capability.
* Complete System (UI and DB included): Vitrivr is not just a model or library; it’s a full system complete with a database (CottontailDB) and even a graphical front-end. CottontailDB is a columnar vector database that supports both boolean (SQL-like) queries and vector similarity queries
* [github.com](https://github.com/vitrivr#:~:text=1)
* . This is useful for combining metadata filters with content similarity. The fact that a dedicated DB is included means you don’t have to integrate something like Faiss or Milvus separately – it’s already handling indexing and similarity search internally. Additionally, the provided web interface can significantly speed up development of a prototype or internal tool. Essentially, Vitrivr could be deployed as-is to end-users who want to search a video collection, which indicates a high level of maturity for an open project.
* Provenance in Academic Benchmarks: Cineast and Vitrivr have been used in competitive evaluations like the Video Browser Showdown (VBS) and TRECVID Ad-hoc Video Search by its developers
* [eurecom.fr](https://www.eurecom.fr/fr/publication/5714/download/data-publi-5714.pdf#:~:text=,VBS%29%202018)
* . These are forums that test retrieval systems on known datasets with live queries. The fact that it has been a contender in those shows that it’s capable of handling challenging search tasks under time constraints. It has been optimized through years of research to be responsive and effective in interactive search scenarios where a variety of query types are thrown at it. This gives some confidence in its accuracy and efficiency on diverse tasks – it may not always use the latest deep model, but its multi-feature approach can compensate (for example, if one method fails, another might catch the detail – a kind of ensemble effect).
* Open Source and Actively Maintained: Vitrivr has been around for several years and remains actively maintained (commits as of late 2024 in the repositories
* [github.com](https://github.com/vitrivr#:~:text=Python%20%201%20%200,Updated%20Nov%2028%2C%202024)
* ). The codebase (mostly Java/Kotlin for the engine and DB) is open under permissive licenses. There is documentation, including research papers and a wiki, which is fairly thorough due to its academic roots. Community-wise, while smaller than mainstream ML projects, there is a niche following and the developers themselves are quite responsive (given their interest in seeing it used). It’s a specialized tool, but within its niche it’s considered a reference system for content-based retrieval. Deploying it doesn’t incur any cost beyond infrastructure, and it doesn’t tie you to any cloud platform. It’s also extensible – you can add new feature modules if needed (for example, plugging in a new embedding model as a feature in Cineast, thanks to its modular architecture).

Cons:

* Not Focused on Scalable Deep Learning: Vitrivr’s multi-feature approach, while comprehensive, includes many traditional descriptors that may not be as discriminative as modern deep embeddings. If not configured properly, the system might rely on color histograms or edge features that yield less relevant results compared to a learned embedding. While it does have deep learning components, it’s unclear if it uses the latest models (it historically supported CNN features like AlexNet or VGG-type descriptors). If it hasn’t integrated models like CLIP or SSCD as feature extractors, it could lag behind in pure accuracy for some queries. In a production setting where deep learning has set a high bar, the default accuracy might need boosting by adding newer models to the pipeline – which requires some work.
* Performance and Scalability Limits: Cineast and CottontailDB, being written in Java/Kotlin, might not match the raw performance of highly optimized C++ libraries (like Faiss) or specialized databases for billions of vectors. CottontailDB is relatively young and not as battle-tested as, say, Milvus or ElasticSearch in large deployments. For very large scale (millions+ videos), one might run into performance bottlenecks unless careful cluster scaling is done. The system may also be heavier in memory usage due to Java overhead. Its retrieval speed, when combining multiple feature types, can be slower because it might do multiple lookups and then fuse scores. In summary, it’s powerful for interactive search on moderate-scale (tens of thousands of videos, as in research evals), but scaling to industrial levels could require significant tuning and possibly hardware investment.
* Deployment Complexity: Setting up Vitrivr involves multiple components: a database (Cottontail, which in turn might need a Postgres for metadata), the Cineast server, and the front-end. The stack is not as simple as running a pip install. You’ll need to build or download the JARs, configure them (feature modules, DB settings), and ensure all parts (including potentially a separate web server for the UI) are working together. This is a higher bar for deployment and maintenance, especially if your team is more familiar with Python-based ML tools. Debugging issues in a Java application or the DB might also be more involved. Furthermore, maintaining this system long-term means keeping up with updates in a variety of technologies (Java server, DB, web UI). Compared to using a single library or a Python pipeline, the overhead here is greater.
* Niche Community: While maintained, the user community is relatively small and mostly academic. The number of practitioners using Vitrivr in industry is limited, so community support is not as robust as for mainstream projects. You may not find quick answers on Stack Overflow for configuration issues. Most guidance comes from official docs or papers, which might not cover every practical scenario. If the core team ever reduces their involvement, the pool of people to step in is limited (though open-source, it’s quite a specialized system). This could pose a risk for long-term maintainability – relying on a niche tool means you either commit to becoming an expert in it yourself or risk it becoming a black box.
* Generalist, Not Optimized for Specific Case: Because it aims to cover “everything” (audio, image, video, 3D, etc.), Vitrivr may not be as optimized for the specific case of *visual video copy search*. For example, its approach to video might emphasize multi-modal retrieval and not be tuned to, say, localize a copied segment within a video at high precision (though theoretically possible with frame features + temporal re-ranking, but one would have to implement that logic manually). If your primary goal is fast near-duplicate detection and not, say, querying by a hand-drawn sketch or an arbitrary example image, then the extra features of Vitrivr could be unnecessary baggage. In that sense, it might not outperform a simpler bespoke pipeline (like SSCD + Faiss + a bit of logic) for the core task, even while offering a broader feature set.

Ranking Justification: We rank Vitrivr #5. It’s a robust and impressively comprehensive open-source solution – great for exploring content-based video retrieval and quickly enabling a range of search functionalities. Its multi-feature approach aligns well with the requirement to support various search dimensions (objects, faces, colors, etc.), arguably more than any single-model solution. However, the complexity of deployment and potentially moderate scalability make it less appealing for a production scenario where efficiency and maintainability are paramount. We believe the accuracy (especially for copy-detection or fine-grained similarity) might also not reach the level of specialized modern models unless one plugs those into Vitrivr. It falls below the top approaches which are more direct, widely supported, and easily scaled. Nonetheless, for an organization that wants a turnkey system and is willing to invest in understanding it, Vitrivr is a viable option – hence we include it, albeit lower on the list. It outperforms a purely custom-from-scratch multi-model approach in terms of having an existing integration of features, but it is ranked below the purpose-built or massively community-supported solutions.

## **6. Custom Multi-Model Ensemble Pipeline**

Overview: The final approach is not a single product, but a strategy: building a bespoke pipeline that combines multiple open-source models to cover all desired facets. In practice, this could mean using an object detection model (like YOLOv5/YOLOv7 or Detectron2) to identify objects in key frames, a face recognition model (like FaceNet or ArcFace) to detect and embed faces, an action recognition model (like Facebook’s SlowFast or Google’s MoViNet) to get an “action” descriptor, and a global image similarity model (like CLIP or a ResNet) for overall visual context. Each video would then be indexed by multiple representations: e.g., a list of objects present, a list of face embeddings (for known face search or grouping), an action feature or label, and a visual embedding for appearance. At query time, one could filter or search by any combination of these (e.g. first find videos that have “car” and “explosion” tags via the detectors, then among those, use the visual embedding to fine-rank similarity to a query scene). Essentially, this approach cherry-picks the best open models for each task and integrates them. Pros:

* Maximum Accuracy and Flexibility: By using specialized models for each aspect, you can achieve very high accuracy in each “dimension.” Object detectors can provide precise identification of dozens of object categories; face recognition models can match identities or find all instances of the same person. Action classifiers are trained on datasets like Kinetics and can recognize hundreds of human actions. Combining these with a similarity model means your system can answer very specific queries that single embeddings would struggle with. For example, “find a video where a person (face detector) is running (action model) in front of a car (object detector)” – each part of that query can be handled by a dedicated model with high confidence. This ensemble approach mirrors how a human might search (filter by attributes, then compare visuals). It can yield extremely relevant results for complex queries and can ensure that retrieved videos meet multiple criteria (not just “looks similar” but “contains these specific elements”).
* Tailored to Use-Case: You have full control to choose models that best fit your content domain and performance needs. If your video domain is, say, sports videos, you can include a sports-specific action model or a pose estimation model for athletic moves. If it’s movies, you might include a place recognition model to match specific landmarks or a costume/clothing classifier. Because you’re assembling it, you can optimize for your context. This also extends to indexing: you can decide how to structure your indices (e.g., a separate index for each feature type, and a method to combine their scores). This kind of fine-tuning can lead to a very high-performing system for a *particular* application, potentially surpassing any single generalized model.
* Reusability of Open Models: All components of this pipeline are readily available as open-source with pre-trained weights. For example, YOLO models (for objects) are on GitHub with permissive licenses, MediaPipe offers face detection, ActionRecognition models are in TorchVision or PyTorchVideo, etc. Many of these run in real-time on modest hardware (some YOLO models can process 30+ FPS on a GPU). So you can leverage the huge communities and documentation of each of these projects. Each model can be independently updated or swapped out when new better ones appear. This gives a modular upgrade path: the pipeline isn’t locked to one monolithic model, which might degrade or become outdated. If a new state-of-the-art object detector comes out, you can integrate it to improve that aspect of search results.
* Granular Search Capabilities: This pipeline naturally produces metadata and auxiliary indices that enrich the search experience. For instance, object detection yields labels (“cat”, “car”, “building”) that can be indexed in a traditional inverted index or database – enabling *instantaneous filtering* by object presence before even touching the similarity search. Face embeddings allow implementing a “find more videos containing this person” feature by nearest-neighbor search in face space, separate from the general video similarity. Action labels let users explicitly search for certain actions. Colors or layouts (could be obtained via simple image analysis or by training a scene classifier) let you filter by scene type (e.g. indoor vs outdoor). In essence, this approach can fulfill the “filter search results by actions/elements prior to fine-grained matching” exactly as described, because you literally have separate outputs corresponding to those facets
* [zilliz.com](https://zilliz.com/ai-faq/what-is-contentbased-retrieval-in-video-search#:~:text=The%20approach%20typically%20involves%20the,based%20searches)
* . This can make the search service far more powerful and user-friendly (users can drill down by facets, much like e-commerce search with filters, but here filters are content-based).

Cons:

* High Engineering Complexity: Building and maintaining such a pipeline is non-trivial. You have to orchestrate multiple models, possibly in different frameworks. Data flows (video frames) must be efficiently shared to avoid duplicative decoding/computation (e.g., you might try to run object and face detection on the same frames in one pass). Combining results from heterogeneous models requires custom logic – how do you rank a video given multi-modal scores? That might involve learning a fusion model or manually weighting criteria. The system is inherently more complex than a single-model solution or even the Vitrivr system (which at least was a packaged integration). More components mean more potential points of failure. Upgrading one model might change the overall output distribution, necessitating re-tuning of the combination logic. In short, it’s high effort to develop and maintain, likely requiring a team with multi-domain expertise.
* Resource Intensive: Running several models means the inference cost multiplies. If you have one GPU model for objects, one for actions, one for faces, plus a global embedding model – that could be 4x the computation per video (if not more, depending on model sizes). Even if each is somewhat optimized, together they will consume significant GPU hours for indexing a large corpus. At query time, unless you precomputed everything (which you would for corpus, but for an ad-hoc query video you’d still need to run all detectors on the query), the latency will be the sum of all model latencies. This could easily become too slow for real-time use, so you might have to limit which models run per query or invest in heavy hardware. There’s also a storage cost: you’re storing multiple indices (one for objects, one for faces, one for actions, plus the main vector index). This increases storage and memory usage. While not as large as frame-level indexing, it’s still more than a single index.
* Integration Overhead: Many models means dealing with different model formats and dependencies. For example, YOLO might be in PyTorch or have a custom repo, a face model might require a specific library (like dlib or InsightFace), an action model could be TensorFlow or PyTorch. Harmonizing these into one pipeline can be a nightmare, especially in production where you want a single, efficient service. One may need to merge them into a single runtime (perhaps by converting some to ONNX or TensorRT for deployment). This is doable but time-consuming. Also, keeping all these components updated with their latest versions (for security or performance improvements) requires continuous effort. Essentially, you become the maintainer of a mini “ensemble system” platform.
* Lack of Unified Community Support: Unlike a singular solution where you can go to a specific forum for help (e.g., CLIP issues on OpenCLIP repo), here the “community” is fragmented by each component. Debugging a problem might involve multiple communities (object detection forum, etc.), or worse, the problem might be in the glue code you wrote to integrate them (meaning you’re on your own). There’s no out-of-the-box community for the *entire pipeline* because it’s custom. This also means if performance is suboptimal, you have to profile and optimize across different parts yourself; you won’t find a pre-written guide for your exact pipeline. This approach has been implemented by some large companies internally, but they have significant resources – for a typical team, it’s a heavy lift and could risk being brittle if not done carefully.
* Potential Redundancy and Diminishing Returns: Some of these models’ functionality overlaps. For instance, a robust video embedding might implicitly capture the presence of certain objects or actions. By explicitly computing them, you ensure you catch them, but it might turn out that for retrieval, the extra signals don’t always improve things dramatically relative to the effort. There’s a risk of over-engineering – the system becomes very complex for marginal gain, especially if the query patterns don’t fully utilize all those dimensions. If most searches are simple (e.g., find similar videos by example), a simpler pipeline could do nearly as well. The multi-model approach is most justified when queries are indeed multi-faceted and when precision needs to be extremely high (e.g., intelligence or forensic applications). If that level of specificity isn’t regularly needed, the maintenance cost might outweigh the benefits.

Ranking Justification: We rank the custom multi-model ensemble #6 (last). While conceptually it can offer the best performance on every axis, it scores lowest on practicality. The question emphasizes a production-grade service and cost-efficiency – this approach is the least cost-efficient and hardest to maintain. It’s essentially what one would do if no single open-source solution met the requirements; but in our case, we have strong single-model or integrated solutions above that get pretty close to the needs without such complexity. The custom approach is powerful but high-risk/high-cost. We see it as a path to consider only if the top-ranked solutions fail to meet certain critical requirements, or if one has a dedicated team to invest in this long-term. It’s telling that even industrial efforts often use a simpler pipeline first and only add such complexity as needed. Therefore, we consider it the least recommended initially – though it remains an option for those who need absolute cutting-edge tailored performance and are willing to handle the engineering challenges.Having evaluated each alternative, we present the following summary of how they compare across key criteria:

## **Summary Recommendation Table**

| **Solution (Rank)** | **Accuracy & Relevance** | **Content Dimensions** | **Speed & Scalability** | **Compute Cost** | **Deployment & Maintenance** | **Community Support** |
| --- | --- | --- | --- | --- | --- | --- |
| 1. Towhee + Milvus Pipeline | Excellent for duplicate detection (uses SOTA model); good semantic coverage if configured with additional models  [towhee.io](https://towhee.io/tasks/detail/pipeline/video-copy-detection#:~:text=)  [towhee.io](https://towhee.io/tasks/detail/pipeline/video-copy-detection#:~:text=,its%20similar%20videos%27%20embedding%20loaded)  . | Multi-feature capable (visual embedding + easy integration of object/action detectors) – flexible to cover actions, objects, etc.  [towhee.io](https://towhee.io/tasks/operator#:~:text=Towhee%20,1%20models)  . | High scalability via Milvus ANN DB (billions of vectors)  [researchgate.net](https://www.researchgate.net/post/Where-can-I-find-a-properly-arranged-database-for-content-based-video-retrieval#:~:text=Where%20can%20I%20find%20a,some%20benchmark%20database%20in)  ; real-time queries feasible. Moderate pre-processing speed (frame sampling + embedding per video). | Moderate – uses one main deep model by default (plus vector DB on CPU). Can increase if extra models added. Overall efficient given robustness. | Turnkey pipeline, relatively easy to deploy (Python + Docker for DB). Some complexity in running two components, but well-documented and actively maintained. | Growing – Milvus is widely used, Towhee is newer but supported by its creators. Active development and community examples. |
| 2. SSCD (Copy-Detection Model) | Top-tier for fine-grained visual similarity – very robust to edits  [github.com](https://github.com/facebookresearch/sscd-copy-detection#:~:text=This%20is%20the%20open,recently%20accepted%20to%20CVPR%202022)  [arxiv.org](https://arxiv.org/pdf/2306.09489#:~:text=SSCD%20descriptor,DISC21%20dataset%20with%20full%20augmentations)  . Best for detecting re-used or overlapping content. Not as semantic for unrelated content. | Focused on overall visual content; does *not* explicitly separate objects or actions (needs supplementary detectors for those). Good for “any visual overlap”, less for specific category search. | Fast per-frame inference (ResNet-based) and small descriptors. But indexing every frame for large corpora can be heavy – requires ANN and possibly multi-stage search for scale. | Low/Moderate – Single CNN model, 512-D vectors. Storage can grow with frame-level index. Compute cost scales with frames processed (can be high for long videos). | Easy to integrate model into custom pipeline. But requires building surrounding indexing & search logic. Maintenance of model itself is simple; overall system upkeep is custom. | Niche but active in vision research. Official open release by Meta  [github.com](https://github.com/facebookresearch/sscd-copy-detection#:~:text=A%20Self,SSCD)  , used in competitions – some support available via GitHub/papers  [arxiv.org](https://arxiv.org/pdf/2306.09489#:~:text=6,supervision%20as%20in%20SSCD%20%5B33)  . Smaller user community than CLIP. |
| 3. OpenCLIP / CLIP4Clip | High semantic relevance – groups videos by concepts and context  [anttihavanko.medium.com](https://anttihavanko.medium.com/building-image-search-with-openai-clip-5a1deaa7a6e2#:~:text=OpenAI%E2%80%99s%20Clip%20is%20a%20neural,a%20given%20text%20or%20image)  . Great for finding related content, though may miss subtle duplicates. | Encodes objects/scenes implicitly. No explicit action modeling (treats frames independently). Can combine with text queries naturally. Faces/ colors not explicit but semantic cues present. | Very fast embedding generation (tens of ms/frame)  [anttihavanko.medium.com](https://anttihavanko.medium.com/building-image-search-with-openai-clip-5a1deaa7a6e2#:~:text=Encoding%20a%20single%20image%20takes,the%20data%20to%20the%20GPU)  ; one vector per video approach scales easily (ANN search is lightweight). Suitable for million+ videos indexing  [anttihavanko.medium.com](https://anttihavanko.medium.com/building-image-search-with-openai-clip-5a1deaa7a6e2#:~:text=to%20index%20them,many%20different%20functionalities%2C%20such%20as)  . | Low – Efficient transformer inference, and few vectors per video to store (memory-friendly). Larger CLIP models increase cost but still far below heavy video models. | Extremely easy – many libraries and pre-trained weights available. Deploy on any platform (PyTorch, ONNX). Long-term maintenance is trivial (widely adopted model, lots of tooling). | Huge community. Well-documented, numerous examples and continual improvements (OpenCLIP releases, etc.). Strong support for model and integration (huggingface, forums). |
| 4. Video Transformers (VideoMAE, TimeSformer) | Very high potential accuracy – captures motion + appearance; excels in action recognition  [huggingface.co](https://huggingface.co/docs/transformers/en/model_doc/videomae#:~:text=%28MAE%29%20to%20video%2C%20claiming%20state,on%20several%20video%20classification%20benchmarks)  . Likely to retrieve fine-grained matches including dynamic events. | Naturally covers actions and temporal patterns. Also encodes objects/scenes per clip. One embedding combines multiple aspects. No separate output per facet, but the info is in the feature. | Slow inference (orders of magnitude heavier than image models)  [drone-vis.readthedocs.io](https://drone-vis.readthedocs.io/en/latest/action_recognition/video_masked_transformer.html#:~:text=Video%20Masked%20Autoencoders%20%28VideoMAE%29%20,)  . Not real-time friendly. Indexing at scale is difficult without massive compute. Search requires ANN but fewer vectors (clip-level). | High – Large GPU memory and compute needed. Running these models for millions of videos is costly. Possibly need model parallel or accelerator usage. Not cost-efficient for general scaling. | Complex – models not as plug-and-play, some available via research code or HF. Deployment requires optimizing transformer for inference. More things to go wrong (performance tuning needed). | Moderate in research circles, low in industry adoption so far. Support mainly from authors’ repos and academic community. Less user-friendly resources for deployment issues. |
| 5. Vitrivr (Cineast) | Good overall retrieval effectiveness by combining features. Not SOTA in deep features, but multi-feature fusion can yield decent accuracy across varied queries. Excels if query matches one of the indexed feature types (e.g. color or shape). | Very rich: indexes multiple facets (color, edge, deep features, text)  [github.com](https://github.com/vitrivr#:~:text=match%20at%20L353%20Cineast%20is,textual%20descriptions%20and%20example%20objects)  . Can detect certain objects/concepts via pre-trained classifiers. Faces/actions support depends on plugins (could be added). Broad but each individual feature might be simplistic. | Designed for interactive use on moderate scale. Vector DB (Cottontail) supports hundreds of millions of entries, but performance may degrade at extreme scale. Merging scores from many features can slow queries. Not as optimized as specialized ANN libraries for single-feature bulk search. | Moderate to High – runs many algorithms (some CPU-bound). JVM-based DB might require substantial RAM. Efficient for what it does, but overall more resource use than a single-model approach that offloads to GPU. | Heavy deployment: multi-component (Java services, DB, front-end). Steep learning curve to configure. Maintenance involves keeping Java and associated stack updated. Less flexible to modify internals unless familiar with codebase. | Small, specialized community. Maintainers are active (academic group) and there are users in multimedia research. However, support is limited to project’s channels; not much third-party ecosystem. |
| 6. Custom Multi-Model Ensemble | Potentially the highest recall/precision if done right – uses best model for each aspect (objects, faces, actions, etc.). Can outperform others in specific queries by ensuring each relevant feature is captured  [zilliz.com](https://zilliz.com/ai-faq/what-is-contentbased-retrieval-in-video-search#:~:text=The%20approach%20typically%20involves%20the,based%20searches)  . However, “potential” is only realized with significant tuning. | Explicit support for all dimensions: separate outputs for actions, objects, faces, color, layout, etc. You can answer very detailed queries and apply fine-grained filters. Ultimate flexibility – you decide what to detect and index. | Slower overall: runs multiple models per video (increases processing time linearly with number of models). Harder to scale – needs parallelism or sacrificing some features per query to keep latency acceptable. Indexing multiple feature sets increases engineering complexity for scaling (different indices for different features). | Very high – multiple deep models running means high GPU hours and memory usage. Storing and updating several indices (vectors, tags) is also resource-intensive. Lacks the efficiency of a unified model, as many computations are redundant (each model running over same video frames). | Most complex to deploy and maintain: essentially building a custom system from scratch. Must manage various model dependencies and ensure they work in unison. Maintenance is on you – updating models or code for each part, keeping the pipeline orchestrated. A lot that can break or require adjustment. | Fragmented – you rely on each model’s community for that part (YOLO’s community for object detection, etc.). No single forum for the whole pipeline. This approach is used in some bespoke systems, but shared knowledge is limited. Requires in-house expertise to troubleshoot and improve. |

*(Note: Ratings in the table are qualitative and relative to each other. They assume an implementation of each solution in its recommended configuration.)*

## **Conclusion**

After thorough comparison, we recommend using an integrated approach built on open-source components, with an emphasis on robust visual embeddings for copy detection augmented by additional content filters as needed. Specifically, the Towhee + Milvus pipeline (Rank #1) stands out as the best starting point – it leverages a state-of-the-art embedding (similar to SSCD) for fine-grained similarity

[towhee.io](https://towhee.io/tasks/detail/pipeline/video-copy-detection#:~:text=)

, and provides a scalable search backend

[towhee.io](https://towhee.io/tasks/detail/pipeline/video-copy-detection#:~:text=,its%20similar%20videos%27%20embedding%20loaded)

. This addresses the core requirement of finding visually similar video segments even under heavy transformations. Towhee’s flexibility means that action recognition or object detection modules can be added to the pipeline to further filter results, fulfilling the multi-dimensional search criteria. In practice, this could mean using Towhee/Milvus for an initial deployment, and progressively enhancing it (e.g., integrate a CLIP-based embedding for semantic search, or add a face recognition stage) as needed, effectively moving slightly towards the multi-model ensemble but in a controlled, maintainable way. The SSCD model (Rank #2), or its close variants, is highly recommended to include (and indeed Towhee uses one by default) for its proven strength in copy detection

[arxiv.org](https://arxiv.org/pdf/2306.09489#:~:text=SSCD%20descriptor,DISC21%20dataset%20with%20full%20augmentations)

. If one were building a system from scratch without Towhee, we would suggest using SSCD or a similar self-supervised ResNet/ViT embedding as the backbone for frame-level matching, combined with a vector index like FAISS or Milvus for scale. It ensures that the service can catch near-duplicate content – a critical capability given the prevalence of remixed media. CLIP (Rank #3) is our next recommendation to cover the semantic gaps. While CLIP alone wasn’t top-ranked for this specific task, incorporating it into the system (for example, to allow text-to-video search, or to re-rank results by semantic similarity) can greatly enhance usability and broaden the search functionality. Its ease of use and low cost make it a low-risk addition that can yield significant benefits in retrieving “related” content, not just exact copies. We advise caution with the video transformer models (Rank #4) – they are promising, especially for action-heavy content, but the infrastructure and cost burden is substantial. They could be considered for off-line analysis or perhaps indexing of particularly important content where deeper understanding is needed, but for most production scenarios, the marginal accuracy gain may not justify the complexity. It might be more practical to use a smaller action recognition model to plug the action-detection gap alongside lighter embeddings. The Vitrivr system (Rank #5) is a strong open-source option for those who want a ready-made solution and are less concerned with having the absolute latest in deep learning. It shines in scenarios where a variety of query types are expected. However, given the rapid progress in deep learning-based retrieval, one might use Vitrivr as a complement (e.g., using it to handle some modalities like audio or OCR search, while using a custom pipeline for the core visual similarity) or as an inspiration for designing a multi-modal search interface. If used, one should plan for dedicated time to configure and possibly extend it with newer models to meet high accuracy expectations. Finally, the custom multi-model ensemble (Rank #6), while not our first recommendation due to its complexity, remains an approach to keep in mind. If down the road the search service needs to answer extremely specific queries with high precision (and if resources allow), gradually evolving the system by adding specialized models (object, face, etc.) could yield the best results. The key would be to do this in a measured way – ensure each added model demonstrably improves search outcomes and justify the maintenance cost. In summary, the recommended strategy is to start with a solid foundation using open-source embedding models geared towards visual similarity (SSCD/CLIP) coupled with a scalable vector search engine, optionally harnessed via frameworks like Towhee for quicker deployment. This covers the fundamental ability to search videos by visual content. Then, iteratively layer in specific detectors or additional embeddings to handle actions, objects, and other facets – thereby meeting the full spectrum of requirements. Each of the top-ranked solutions contributes to this vision: Towhee+Milvus provides the framework and scale; SSCD ensures copy-detection accuracy; CLIP adds semantic range. Lower-ranked options like advanced video transformers or full multimedia systems can be drawn upon as needed for further enhancement. By combining these open-source technologies, one can implement a production-grade video search service that is accurate, versatile, and maintainable without reliance on proprietary services

[zilliz.com](https://zilliz.com/ai-faq/what-is-contentbased-retrieval-in-video-search#:~:text=The%20approach%20typically%20involves%20the,based%20searches)

. The open-source ecosystem in 2025 is rich enough to achieve this, and our analysis shows that the best results will likely come from intelligently fusing multiple tools, with the listed rankings guiding which to prioritize first.

Favicon

Favicon

Favicon

Favicon

Favicon

Sources

##### **You said:**

Is aws titan G1 able to recognize action ?

###### **ChatGPT said:**

No, AWS Titan G1 is not designed to recognize actions.

### **Here's what Titan G1 is optimized for:**

* It's a general-purpose embedding model, focused on semantic understanding of images, text, and image-text pairs.
* It's most similar in nature to CLIP (Contrastive Language-Image Pretraining), and is used for tasks like:
  + Image or text similarity
  + Image classification
  + Search and retrieval based on semantic content
  + Multimodal understanding (image ↔ text)

### **What it does not do:**

* Titan G1 does not have temporal modeling — it doesn’t process video sequences.
* It processes individual images or frames, and therefore lacks the capacity to analyze motion or actions, which require an understanding of change over time.
* It does not output action labels (like “running”, “jumping”, “dancing”) nor is it trained on datasets like Kinetics or Something-Something that are typical for action recognition tasks.

### **If you're looking for action detection, you should look into:**

* VideoMAE, TimeSformer, SlowFast – all of which are trained on motion and temporal dynamics.
* Or augment Titan G1 / CLIP-like models with an action recognition model that handles short video clips (e.g. 8–32 frames) rather than static images.

✅ Summary: Titan G1 = great for semantic image/video frame similarity, but ❌ not for action recognition. For action-based filtering or search, you need a model trained specifically on temporal data.

##### **You said:**

Our strategy is to use CLIP for cheap and fast search and then, only if needed run sscd on the results found through clip based search. This filtered approach allows use to have a good pre filtering before running sscd that is more costly.

What do you think of this approach?

Are embeddings produced by CLIP searchable with aws open search service with knn?

###### **ChatGPT said:**

Your strategy is smart, cost-effective, and scalable — and it's a well-established pattern in large-scale similarity search. Here's a breakdown of why it's solid and what to watch for:

## **✅ Your Strategy: CLIP Pre-Filter + SSCD Re-Rank**

### **Strengths:**

1. Cost-Efficient Filtering
   * CLIP is fast to compute, with lightweight inference (esp. ViT-B/32 or ViT-B/16).
   * Embedding each video with CLIP (e.g. one vector per video or sampled keyframes) keeps your index small and fast to search, especially with ANN (Approximate Nearest Neighbor) techniques like HNSW.
2. Semantic Relevance First
   * CLIP returns content- and conceptually-similar results (e.g., “a guy in front of a screen”) — which often surfaces *visually* relevant but not exact matches.
   * You can rank/filter broadly before diving into frame-level fine-grained SSCD analysis.
3. SSCD on Demand = Cost Control
   * By running SSCD only on top-K candidates, you're dramatically reducing its GPU cost and compute time.
   * SSCD is excellent at identifying remixes, picture-in-picture edits, green screen overlays — so it’s the perfect *second-pass precision layer*.
4. Scalability
   * You can index millions of CLIP embeddings in a KNN-compatible vector store like OpenSearch, Faiss, or Milvus.
   * SSCD inference happens *only if needed*, letting you stay lean at scale.

## **🔎 Can CLIP Embeddings Be Used with AWS OpenSearch KNN?**

### **✅ Yes, CLIP embeddings can be used with OpenSearch's k-NN plugin.**

### **Details:**

* OpenSearch supports vector search via the k-NN plugin, compatible with dense vector fields (like CLIP's 512-dimensional float arrays).
* Compatible index types: HNSW, IVF, etc.
* You store CLIP embeddings as dense\_vector fields, then query using cosine similarity, L2, or dot product — all of which work well with CLIP.

#### **Example OpenSearch setup:**

json

Copy

Edit

PUT my-index { "settings": { "index": { "knn": true } }, "mappings": { "properties": { "clip\_vector": { "type": "knn\_vector", "dimension": 512 } } } }

Then query with:

json

Copy

Edit

POST my-index/\_search { "size": 10, "query": { "knn": { "clip\_vector": { "vector": [ ... your CLIP embedding ... ], "k": 10 } } } }

✅ Best distance metric for CLIP: cosine similarity, or dot product if normalized.

## **🧠 Things to Watch Out For**

* Normalize CLIP embeddings: CLIP vectors are typically L2-normalized. If using cosine or dot product, ensure your index/query vectors are also normalized.
* Frame Sampling Strategy: CLIP works best if your frame sampling captures the video’s diversity. You can aggregate by averaging, concatenating, or using attention pooling methods.
* Latency: SSCD re-ranking can be slow if you re-encode frames at query time. Precompute SSCD descriptors for your dataset and only encode the query video/frame at runtime.
* Index Growth: If you go with frame-level CLIP (instead of per-video), monitor index size — it can grow fast. Stick to 1–3 embeddings per video unless fine-grained search is a must.

## **🧩 Optional Enhancements**

* Add object tags or scene labels with fast detectors (YOLOv8, etc.) to enhance filtering.
* Use CLIPScore or cross-modal CLIP ranking (image-text) if you later add text queries.
* Explore hybrid search (e.g., metadata + embedding in OpenSearch) to filter before even running ANN search.

### **✅ Final Take**

Your approach is *excellent*: fast + cheap filtering with CLIP, accurate + precise verification with SSCD. Yes, OpenSearch is fully compatible with CLIP vectors, and your setup should scale well across millions of indexed items.

Let me know if you want example code to ingest CLIP vectors into OpenSearch or optimize hybrid search.

##### **You said:**

I’ll want to expand my infrastructure on Google cloud in the next 2 years. Should I use aws open search or go with an open source solution ?

###### **ChatGPT said:**

If your long-term plan includes expanding to Google Cloud, then using AWS OpenSearch Service (managed) now will lock you into AWS, making your migration harder and costlier later.

Here's a strategic breakdown:

## **✅ Go with an Open-Source Solution You Can Host Anywhere**

### **Why:**

1. Cloud-Agnostic = Future-Proof
   * Open-source vector search tools like Milvus, Weaviate, Qdrant, or even OpenSearch self-hosted can run on any cloud provider, including Google Cloud, AWS, or even on-prem.
   * You can start on AWS and move to GCP later without re-architecting your system.
2. Avoid Vendor Lock-In
   * AWS OpenSearch Service (managed) is convenient but:
     + Tied to AWS IAM, VPCs, and billing models
     + Often lags behind the open-source OpenSearch community in terms of updates
     + Migration later will require data export, config rewrites, and likely downtime
3. More Control & Features with Open-Source
   * Want to use GPU acceleration, custom scoring, or tight integration with other OSS tools?
   * Open-source versions (e.g. Milvus or Qdrant) are often more feature-rich and give you access to internals and plugins.
   * You can even optimize costs using spot instances, autoscaling, or serverless setups on your own terms.
4. Better Long-Term Economics
   * Managed OpenSearch pricing can become prohibitively expensive at scale.
   * With OSS, you can:
     + Use GKE or ECS for orchestration
     + Scale compute/storage independently
     + Optimize infra spending across clouds

## **🔁 Recommended Strategy**

### **Start with:**

* Milvus, Qdrant, or OpenSearch self-hosted:
  + All have KNN + ANN support, high-scale readiness, and GCP-compatible Helm charts and Terraform modules.
  + Choose depending on your familiarity and feature needs:
    - Milvus: Highly scalable, great for large-scale similarity search.
    - Qdrant: Easy to use, excellent API, lightweight and fast.
    - OpenSearch (OSS): Familiar if you're already on Elasticsearch/OpenSearch stack.

### **Benefits:**

* Portable across cloud providers
* Easier cost management
* Better debugging and observability control
* No surprises from AWS-specific service limits or pricing

## **👎 When AWS OpenSearch Service Might Still Make Sense**

Only consider it if:

* You’re 100% committed to AWS for at least 2–3 years
* You need zero infra overhead right now and are willing to pay for it
* You’re already deeply integrated into other AWS-managed services (S3, Lambda, IAM, etc.)

Even then: deploy your OpenSearch logic in a way that’s portable, using interfaces that could be switched later (e.g., abstract indexing and search functions behind APIs).

### **✅ Final Recommendation**

Go open source. Use a cloud-agnostic vector DB like Milvus, Qdrant, or self-hosted OpenSearch.  
You’ll retain flexibility, reduce long-term costs, and be ready to scale on GCP (or multi-cloud) when needed.

Let me know if you want a quick comparison table between Milvus, Qdrant, and OpenSearch OSS.

##### **You said:**

« cosine similarity, L2, or dot product » what are the difference between these ?

###### **ChatGPT said:**

Great question — these three similarity metrics are at the heart of how vector search works. They measure how “close” two vectors are in embedding space, but they do it differently, and choosing the right one can significantly affect your search results.

## **🔁 The 3 Metrics at a Glance**

| **Metric** | **Measures** | **Sensitive to Vector Magnitude?** | **Best For** |
| --- | --- | --- | --- |
| Cosine similarity | Angle between two vectors | ❌ No | Semantic similarity (e.g., CLIP) |
| L2 distance (Euclidean) | Straight-line distance | ✅ Yes | Physical similarity, spatial data |
| Dot product | Magnitude of alignment + size | ✅ Yes | Fast similarity (if normalized) |

## **🧠 1. Cosine Similarity**

* Formula:
* cosine\_sim(A,B)=A⋅B∥A∥∥B∥
* cosine\_sim(*A*,*B*)=
* ∥*A*∥∥*B*∥
* *A*⋅*B*
* ​
* Interpretation: Looks at the angle between two vectors.
* Range: –1 to 1 (but usually 0 to 1 for embeddings).
* Magnitude: Ignored — only the direction matters.
* Why it's good:
  + Works well for semantic embeddings like CLIP, where you care about "meaning" similarity, not pixel-for-pixel precision.
  + If two videos are *about* the same thing (e.g., a man playing guitar), even if lighting/scale differs, the angle will still be small → high similarity.

✅ Use it with CLIP or text/image embeddings.

## **🧠 2. L2 Distance (Euclidean)**

* Formula:
* L2(A,B)=∑(Ai−Bi)2
* L2(*A*,*B*)=
* ∑(*A*
* *i*
* ​
* −*B*
* *i*
* ​
* )
* 2
* ​
* Interpretation: The straight-line distance between points in space.
* Range: 0 to ∞ (smaller = more similar).
* Magnitude: Matters a lot — two vectors pointing in the same direction but with different lengths will be far apart.
* Why it's good:
  + Best when vectors are not normalized.
  + Good for geometry, image similarity with exact pixel-like structure.

✅ Use it when magnitude is meaningful — e.g., for raw feature maps or embeddings where larger values encode stronger presence of features.

## **🧠 3. Dot Product**

* Formula:
* A⋅B=∑Ai×Bi
* *A*⋅*B*=∑*A*
* *i*
* ​
* ×*B*
* *i*
* ​
* Interpretation: Mix of alignment and magnitude.
* Range: –∞ to ∞ (higher = more similar).
* Magnitude: Yes — larger vectors = larger dot product.
* Why it's good:
  + Can be faster to compute than cosine (since no normalization step).
  + If your vectors are already L2-normalized, dot product ≈ cosine similarity.

✅ Use dot product if you're optimizing for speed and you've normalized all vectors ahead of time.

## **🧪 Which One Should You Use?**

### **For CLIP or text/image embeddings (semantic content):**

✅ Cosine similarity or dot product (with normalized vectors)

### **For raw visual descriptors (e.g., SSCD or ResNet features):**

✅ Use L2 if you're not normalizing them  
⚠️ But if you normalize SSCD vectors, cosine or dot product are also fine.

### **For embedding-based search with OpenSearch:**

* OpenSearch knn\_vector supports:
  + "l2" (default)
  + "cosine"
  + "dot\_product"
* Choose based on whether your vectors are normalized:
  + Normalized → use dot\_product or cosine
  + Not normalized → use L2

Let me know if you want example queries or a test dataset to see how they behave differently in search results.

##### **You said:**

What does normalize mean in this context ?

###### **ChatGPT said:**

Great follow-up. In this context, "normalize" means adjusting a vector so that its length (or norm) equals 1 — without changing its direction.

## **📐 What Does It Mean to Normalize a Vector?**

A vector like this:

ini

Copy

Edit

A = [3, 4]

Has a length (a.k.a. L2 norm) of:

∥A∥=32+42=5

∥*A*∥=

3

2

+4

2

​

=5

### **To normalize it:**

You divide each component by the length:

Anormalized=[35,45]=[0.6,0.8]

*A*

normalized

​

=[

5

3

​

,

5

4

​

]=[0.6,0.8]

Now the vector still points in the same direction, but:

∥Anormalized∥=0.62+0.82=1

∥*A*

normalized

​

∥=

0.6

2

+0.8

2

​

=1

## **✅ Why Normalize for Similarity Search?**

### **1. Cosine Similarity is based on vector angle**

* If two vectors are both normalized to length 1, then:
  + Cosine similarity = dot product (simpler math)
  + You don't need to calculate norms during search
  + Faster, less error-prone

### **2. Normalized embeddings = better distance consistency**

* Vectors with different magnitudes may skew L2 or dot product comparisons if not normalized.
* Normalizing ensures distance depends only on direction, not on "intensity" or "scale" of the vector.

## **🧠 When to Normalize Your Embeddings**

| **Case** | **Normalize?** | **Why** |
| --- | --- | --- |
| CLIP embeddings | ✅ Yes | Trained to be compared with cosine/dot |
| Text/image similarity | ✅ Yes | Semantics in direction, not magnitude |
| SSCD / visual copy detection | ⚠️ Optional | Sometimes yes (e.g. for cosine); sometimes no for L2 |
| Audio or signal features | ⚠️ Depends | Magnitude may encode volume/presence |

## **🔧 How to Normalize (in code)**

In Python (NumPy):

python

Copy

Edit

import numpy as np v = np.array([3.0, 4.0]) v\_normalized = v / np.linalg.norm(v)

In PyTorch:

python

Copy

Edit

import torch v = torch.tensor([3.0, 4.0]) v\_normalized = torch.nn.functional.normalize(v, p=2, dim=0)

## **🔚 Summary**

Normalization = make a vector’s length equal to 1 by dividing it by its norm.

✅ Use normalization when using cosine similarity or dot product to avoid letting magnitude distort similarity.

Let me know if you want to test what happens with and without normalization in OpenSearch or Faiss!

##### **You said:**

Can I make a video to video cosine search with clip?

###### **ChatGPT said:**

Yes, you absolutely can perform video-to-video cosine similarity search using CLIP — it's a common and effective method for semantic video retrieval.

Here’s how to do it 👇

## **✅ How to Use CLIP for Video-to-Video Cosine Search**

Since CLIP is an image encoder, not a video encoder, the trick is to represent each video as a single (or few) CLIP embedding(s). Here's how.

### **🔁 Step-by-Step: Building a CLIP-Based Video Embedding Index**

#### **1. Extract Keyframes from Each Video**

* Sample frames at regular intervals (e.g. 1 frame every second, or 12 frames total).
* Goal: capture the semantic diversity of the video without extracting every frame.

#### **2. Generate CLIP Embeddings per Frame**

python

Copy

Edit

import clip import torch from PIL import Image model, preprocess = clip.load("ViT-B/32", device="cuda") # For each frame (as PIL image) image\_input = preprocess(frame).unsqueeze(0).to("cuda") with torch.no\_grad(): embedding = model.encode\_image(image\_input) embedding = embedding / embedding.norm(dim=-1, keepdim=True) # normalize!

#### **3. Aggregate Frame Embeddings into a Video Embedding**

Choose one of the following:

* Average: simple and effective
* Max-pooling: more sensitive to dominant visual themes
* Weighted (e.g. attention-based pooling if you want to get fancy)

python

Copy

Edit

video\_embedding = torch.stack(frame\_embeddings).mean(dim=0) video\_embedding = video\_embedding / video\_embedding.norm()

Now you have one vector per video.

### **🔍 4. Store in Vector Index (e.g. OpenSearch, Faiss, Qdrant)**

Store all video embeddings with metadata (video ID, title, etc.) in a vector database that supports cosine similarity (or dot product on normalized vectors).

### **🔁 5. At Query Time (Video-to-Video)**

* Extract frames from the query video the same way.
* Compute and normalize the query video’s embedding.
* Search your index using cosine similarity (or dot product).

Most vector databases let you do this in milliseconds over millions of videos.

## **✅ Benefits of CLIP for Video-to-Video Search**

| **Feature** | **Description** |
| --- | --- |
| Fast | Frame-level embedding is cheap to compute on GPU (~20ms/frame). |
| Semantic-aware | Finds *related* videos, not just duplicates (e.g. same action, similar setting). |
| Cross-modal extendable | You can later support text-to-video queries with the same index. |
| Open-source & scalable | Works with Faiss, OpenSearch, Qdrant, Weaviate, etc. |

## **⚠️ Limitations to Keep in Mind**

| **Limitation** | **Workaround** |
| --- | --- |
| No motion understanding | Use CLIP + action recognition model (e.g. SlowFast) if needed |
| Might miss fine-grained copies (green screen edits) | Run SSCD or another copy-detection model after CLIP pre-filter |
| Requires smart frame sampling | Uniform sampling or scene change detection can improve results |

### **🧠 Bonus Tip**

If you want a more CLIP-native approach to comparing entire videos:

* Try CLIP4Clip (a transformer that aggregates CLIP over video sequences)
* Or use ViViT / VideoMAE if you want real motion-aware video embeddings (heavier, less scalable)

## **✅ Final Verdict:**

Yes, you can — and should — use CLIP for video-to-video cosine similarity search.  
It’s fast, scalable, and gives surprisingly good semantic matching.

Let me know if you want example code to build such a pipeline or compare it with SSCD post-filtering.

Tools

ChatGPT can make mistakes. Check important info. See Cookie Preferences.