Scalable Image Search Using CLIP: Architecture Evolution, GPU Acceleration, and Practical Evaluation of a Command-Line Tool

Yurij Mikhalevich
QA Wolf
Melbourne, Australia
email: yurij@mikhalevi.ch

Abstract - **This paper presents an extended study of `rclip`, a practical, scalable, command-line image search engine that leverages Contrastive Language-Image Pre-Training to enable natural language queries over unlabeled image collections. Extending the original conference paper, this work describes the substantial architectural evolution of `rclip`, including its migration from OpenAI's `clip` library to the open-source `open_clip` library, the introduction of graphics processing unit acceleration via Apple's Metal Performance Shaders backend, configurable batched indexing for improved throughput, and a text-only model loading optimization that significantly reduces query latency when indexing is not required. The paper also reports corrected benchmark results for the ImageNet-1k dataset: the original conference paper reported a top-1 accuracy of 31.17% due to a bug in the benchmark code that silently discarded 488 of the 1000 ImageNet classes; the corrected measurement yields 58.78% top-1 accuracy with `ViT-B-32-quickgelu`, the `open_clip` variant that uses the original OpenAI Contrastive Language-Image Pre-Training model weights. Metal Performance Shaders acceleration further reduces indexing time for 1.28 million images on an Apple M1 Max from 4 hours 52 minutes to 2 hours 16 minutes, a 2.15× speedup. Additional improvements include expanded image format support, an improved file system traversal mechanism, and an evolved database schema. The results demonstrate that semantic image search powered by vision-language models is now practical for everyday use on consumer hardware, through an accessible command-line interface, without cloud infrastructure or specialized servers.**

Keywords - image search; image indexing; CLIP; natural language processing; computer vision.

I. Introduction

This paper is an extended version of the original conference paper [1], which introduced `rclip`, a command-line tool for searching unlabeled image collections using natural language, and demonstrated its practical viability. It substantially extends that work with new material on `rclip`'s evolution, updated experimental results, and an expanded literature review.

The release of the Contrastive Language-Image Pre-Training (CLIP) model by OpenAI in 2021 has generated significant attention in the field of Natural Language Processing (NLP) and Computer Vision (CV). This model has demonstrated the ability to learn state-of-the-art image representations from a massive dataset of 400 million (image, text) pairs that were collected from the Internet. CLIP can predict the most relevant text snippet, given an image, using natural language instructions without explicitly optimizing for the task. This zero-shot ability is similar to that of GPT-2 and 3 [2]. The authors of CLIP have demonstrated that CLIP performs as well as the original ResNet50 on ImageNet in a zero-shot manner without using any of the original 1.28M labeled examples. This accomplishment overcomes significant challenges in computer vision.

In addition to these capabilities, CLIP allows researchers to perform image searches using natural language queries. This is precisely the application that `rclip` explores, demonstrating the practical viability of searching unlabeled image collections using natural language.

Since the publication of the original conference paper, the CLIP ecosystem has evolved considerably. The open-source `open_clip` library [3] has emerged as a reproducible alternative to OpenAI's original implementation, trained on publicly available datasets such as LAION-5B [4]. GPU acceleration for machine learning workloads on consumer hardware has also become more accessible, with Apple's Metal Performance Shaders (MPS) backend for PyTorch [5] enabling GPU-accelerated inference on Mac devices without requiring dedicated NVIDIA hardware. These developments have motivated a substantial evolution of `rclip`'s architecture and capabilities.

The exponential growth of data, particularly in the form of images, makes this research especially relevant. With the proliferation of digital devices and the Internet, more and more images are being produced and shared online every day. As a result, it is becoming increasingly challenging to find specific images manually. A robust and efficient image search solution can help users and businesses quickly find the images they need amidst this growing sea of data. Moreover, as the volume of photos being shared online continues to increase, it becomes increasingly critical to protect intellectual property and online security. An efficient image search solution can help identify and remove images that violate copyright laws or contain sensitive or inappropriate content. Therefore, with the growing amount of data and images being created, an efficient image search solution is becoming increasingly necessary and relevant.

This extended journal paper makes the following contributions beyond the original conference publication:

- A detailed description of `rclip`'s migration from OpenAI's `clip` library to the open-source `open_clip` library, including the rationale and practical implications.
- An analysis of GPU acceleration through Apple's MPS backend, with quantitative performance comparisons against CPU-only execution.
- Corrected benchmark results for the ImageNet-1k dataset, along with a thorough explanation of the bug that affected

the original measurements.
- An expanded implementation description covering batched indexing, text-only model loading, expanded image format support, and database schema evolution.
- An expanded literature review covering recent developments in the CLIP ecosystem, CLIP-based retrieval tools, and GPU acceleration for consumer hardware.

The remainder of this paper is organized as follows. Section II explores related work in image search, CLIP-based systems, and GPU acceleration. Section III describes the proposed method for image search using CLIP, including extensions for query arithmetic. Section IV presents the implementation details, including the architectural changes introduced since the original publication. Section V discusses indexing and search performance, including MPS acceleration results. Section VI presents corrected search quality measurements and discusses the benchmark bug found in the original paper. Section VII situates `rclip` among other CLIP-based retrieval tools. Section VIII discusses practical implications, limitations, and lessons learned. Finally, Section IX concludes the paper and discusses future work.

II. Related Work

There exist multiple methods and solutions for image search. This section reviews both traditional approaches and the more recent developments in the CLIP ecosystem that are most relevant to this work.

A. Traditional Image Search Methods

Traditional image search methods rely on bag-of-features, which are sets of features that describe the contents of an image. These features can be manually specified by annotating the image with descriptive labels. For instance, Fiedler et al. [6] proposed an image tagging software that helps users enter the labels efficiently. Another way to obtain labels is to use labels injected into the photos by camera software, as explored by Tesic [7], who examined camera metadata for consumer photos.

Mobile phone software can also inject useful metadata within photos. Kim et al. [8] explored how this metadata can be leveraged to effectively manage and search photos using mobile smartphones. This metadata can include information about the location, time, or type of camera used, among others. By using such metadata, it's possible to automatically generate labels for images.

One way to automatically generate labels is by using image recognition algorithms, such as Convolutional Neural Networks (CNNs). CNNs can be specifically trained to recognize and classify images based on their contents. For example, Krizhevsky et al. [9] trained a CNN to recognize images based on their visual features. This approach can be combined with other methods, such as using camera metadata or manual annotations, to generate more accurate and diverse labels for images. Lee et al. [10] proposed a scalable method for image annotation that combines manual and automatic approaches to improve the accuracy and scalability of the labeling process.

In image retrieval, search algorithms are applied to the features extracted from images to find relevant images based on a user's query. The choice of search algorithm depends on the type of labels associated with the images and the problem at hand. For instance, if the labels are GPS coordinates, a distance-based search algorithm can be used to find images related to the query location, as demonstrated by Zhang et al. [11].

On the other hand, if the labels are in text form, a text-based search algorithm can be employed to retrieve images based on textual queries. There are various techniques available for text-based search, ranging from substring matching to lemmatization-based search, as shown by Balakrishnan et al. [12], or even using word embeddings, as explored by Günther [13] and Kenter et al. [14].

In practice, combining multiple features and approaches, such as GPS coordinates, text labels, image capture date, camera model, focal distance, etc. can yield more accurate and relevant results. Ismail [15] investigated image annotation and retrieval based on multi-modal feature clustering and found that this approach can significantly improve the retrieval performance of the system.

Therefore, in summary, image retrieval is a complex process that involves the extraction of features from images, the selection of suitable search algorithms based on the type of labels and the integration of multiple features and approaches to improve retrieval accuracy.

The methods mentioned earlier do not facilitate searching for images using a text query without first identifying the features. Moreover, they do not permit searching for a concept that is not included in the image labels, even if the concept exists within the image. These are the problems that OpenAI's CLIP [2] enables us to solve.

B. CLIP and Vision-Language Models

CLIP [2] introduced a paradigm shift in image understanding by jointly training an image encoder and a text encoder on 400 million image-text pairs using a contrastive learning objective. The resulting model maps images and text into a shared embedding space, enabling zero-shot transfer to a wide variety of downstream tasks. The image encoder in CLIP can be either a ResNet or a Vision Transformer (ViT) [16]. The ViT architecture, which treats an image as a sequence of fixed-size patches and processes them with a standard Transformer encoder, has proven particularly effective for learning rich visual representations. CLIP's `ViT-B/32` variant, which uses 32×32 pixel patches, offers a practical balance between computational cost and representation quality, and is the variant used in `rclip`.

Following CLIP, several related vision-language models have been proposed. ALIGN [17] demonstrated that similar zero-shot transfer capabilities can be achieved with noisier but larger-scale training data (1.8 billion image-alt-text pairs), using a simpler dual-encoder architecture without the careful data curation employed by CLIP. BLIP [18] extended the paradigm by bootstrapping captions from a noisy web dataset

and introducing a multimodal mixture of encoder-decoder architectures that unifies vision-language understanding and generation. SigLIP [19] replaced CLIP's softmax-based contrastive loss with a sigmoid loss that operates on independent image-text pairs rather than requiring a full pairwise similarity matrix, enabling more efficient training at larger batch sizes.

C. OpenCLIP and Reproducible Models

A significant limitation of the original CLIP release was that OpenAI did not publish its training data or training code, making it impossible for the research community to reproduce or extend the training process. The `open_clip` library [3] addressed this gap by providing a fully open-source implementation of CLIP's training and inference pipeline. Cherti et al. conducted an extensive study of scaling laws for contrastive language-image learning, training models on publicly available datasets including LAION-400M and LAION-2B [4] (a 2-billion-sample subset of LAION-5B). Their work demonstrated that the scaling behavior of CLIP-like models is predictable and reproducible when both code and data are open. The `open_clip` library provides pre-trained model checkpoints with detailed documentation of training configurations, including the specific dataset, image resolution, and number of training samples seen. This transparency was a key motivation for `rclip`'s migration from OpenAI's `clip` library to `open_clip`, as discussed in Section IV.

D. Similarity Search at Scale

For large-scale image retrieval systems, the efficiency of the similarity search step becomes critical. Libraries such as FAISS [20] provide highly optimized implementations of approximate nearest neighbor (ANN) algorithms, including Inverted File Indices (IVF) and Hierarchical Navigable Small World (HNSW) graphs, that can search through billions of vectors in milliseconds. These approaches trade a small amount of retrieval accuracy for dramatic improvements in query latency.

However, ANN indices introduce additional complexity: they require a separate index-building step, consume additional memory, and their accuracy depends on tuning parameters such as the number of clusters or graph connectivity. For `rclip`'s target use case—personal photo collections and moderately sized image catalogs typically containing thousands to low millions of images—the brute-force dot product computation over cached vectors remains fast enough. As shown in Section V, querying 1.28 million images takes approximately 11 seconds on an Apple M1 Max, which is acceptable for an interactive command-line tool. Should future scaling requirements demand sub-second queries over tens of millions of images, integrating an ANN index would be a natural extension.

E. GPU Acceleration on Consumer Hardware

The introduction of the MPS backend for PyTorch [5] in 2022 marked an important step toward accessible GPU acceleration on consumer hardware. MPS enables PyTorch operations to execute on Apple Silicon GPUs (M1, M2, M3, M4, and M5 series) on Mac systems. The unified memory architecture of Apple Silicon is particularly well-suited for machine learning inference workloads, as it eliminates the need to copy data between CPU and GPU memory. This is relevant for `rclip` because many of its users run the tool on Mac laptops and desktops. The MPS backend enables `rclip` to leverage hardware acceleration, significantly improving the indexing throughput as demonstrated in Section V.

III. Method

With CLIP's text transformer, it is possible to convert a text query to a n-dimensional vector (where n differs depending on the CLIP model used). With CLIP's image transformer, it is possible to convert an image to a n-dimensional vector. Then, we can calculate the dot product (Equation (1)) of the normalized query vector and each of the normalized image vectors. After this, we sort the images by the decreasing dot product and take first k images; this gets us the k images that match the query the most.

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \tag{1}$$

where $\boldsymbol{a}$ is the L2-normalized query vector (text or image) and $\boldsymbol{b}$ is the L2-normalized image feature vector.

It is worth noting that because both the query and image vectors are L_2-normalized (i.e., $\|\boldsymbol{a}\| = \|\boldsymbol{b}\| = 1$), the dot product is equivalent to cosine similarity (Equation (2)). This normalization ensures that similarity scores are bounded between -1 and 1, enabling meaningful comparison across different queries and image collections.

$$\text{cosine\_similarity}(\boldsymbol{a}, \boldsymbol{b}) = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\|\boldsymbol{a}\| \|\boldsymbol{b}\|} = \boldsymbol{a} \cdot \boldsymbol{b} \tag{2}$$

where the last equality holds when $\|\boldsymbol{a}\| = \|\boldsymbol{b}\| = 1$.

While this approach works reasonably fast on a few images, it may not be scalable when dealing with a large number of images. This is particularly challenging if there is no access to GPUs to run the CLIP model. In order to address this scalability issue, the solution proposed in this paper suggests caching the image vectors. By caching the image vectors, repeated queries can be executed quickly, without having to wait for the image to be processed each time that a query is made.

Caching involves storing the computed image vectors on disk after the initial processing of images so that they can be accessed later. When a new query is received, the system retrieves the cached image vectors and computes only the query vector. This approach reduces the computational overhead associated with image querying and improves the response time for users. This is particularly useful when dealing with large image databases, where repeated queries are common. By leveraging caching, the system can handle repeated queries quickly and efficiently without having to recompute the image vectors each time.

The proposed solution also allows adding new images to the cache to avoid recomputing the whole cache when the image catalog is updated.

A. Query Arithmetic

Beyond simple text queries, the current version of `rclip` supports query arithmetic, enabling users to refine their search results through additive and subtractive query composition. Users can specify positive queries (which increase the similarity score for matching images) and negative queries (which decrease it). The combined feature vector is computed as:

$$\boldsymbol{f} = \sum_i \boldsymbol{q}_i^+ - \sum_j \boldsymbol{q}_j^- \tag{3}$$

where $\boldsymbol{q}_i^+$ denotes positive query feature vectors and $\boldsymbol{q}_j^-$ denotes negative query feature vectors. This is analogous to the vector arithmetic property observed in word embeddings, where semantic relationships can be expressed as vector operations.

Additionally, `rclip` supports query multipliers, allowing users to assign different weights to individual query components. A multiplied query takes the form `<multiplier>:<query>`, where the multiplier scales the corresponding feature vector before summation. This enables fine-grained control over the relative importance of different query aspects.

B. Multimodal Queries

A distinctive feature of `rclip`'s method is its support for multimodal queries. Because CLIP maps both images and text into the same embedding space, `rclip` can accept not only text queries but also image files and URLs as query inputs. When an image is provided as a query, `rclip` computes its CLIP embedding and uses it as the query vector, effectively performing image-to-image similarity search. This enables use cases such as finding visually similar photos in a collection, identifying near-duplicates, or locating different shots from the same scene.

Text and image queries can also be combined in a single search. For example, a user could search for images similar to a reference photo but with specific characteristics described in text (e.g., providing a landscape photo as input and adding the text query "sunset" to find similar landscapes specifically at sunset). The feature vectors from all query components—text, local image files, and URL-referenced images—are summed element-wise, with optional multipliers for weighting, to produce the final composite query vector.

The main advantage of the proposed method over the methods described in Section II-A is that the method does not require any metadata, labels, or annotations, which enables using it with any image catalog. Moreover, if used on labeled images, the proposed method allows searching for concepts not included in the image labels. The ability to combine text and image queries, apply query arithmetic with positive and negative components, and weight individual query parts with multipliers provides a flexible and expressive query language that enables users to utilize CLIP effectively in real-world scenarios involving image search.

IV. Implementation

The method described above is implemented in the Python utility called `rclip` [21]. `rclip` provides an easy-to-use command-line interface (CLI) that allows users to search images within any directory on a computer where `rclip` is installed. Search within nested directories is also supported. To use it, the user should open the terminal, navigate to the directory containing the files that they want to search through, type `rclip <search query>`, and hit "Enter."

Since the original conference publication, `rclip` has undergone a substantial architectural evolution. The following subsections detail the key changes and their rationale. Figure 1 shows the overall system pipeline.

A. Migration to OpenCLIP

The original version of `rclip` used OpenAI's `clip` library [2] directly to load the `ViT-B/32` model and compute feature vectors. While functional, the library received minimal updates after its initial release, and the training dataset (WebImageText) was never publicly released, making it impossible for the community to reproduce the training process or train new models.

The current version of `rclip` uses the `open_clip` library [3]. While both OpenAI's `clip` and `open_clip` provide open-source inference code, `open_clip` additionally includes the full training pipeline and provides checkpoints trained on public datasets such as LAION-5B [4], enabling fully reproducible model training. The `open_clip` library is also more actively maintained and offers a wider selection of model variants. `rclip` uses the `ViT-B-32-quickgelu` variant with the `openai` checkpoint. The `quickgelu` suffix indicates that this variant preserves the QuickGELU activation function from the original OpenAI CLIP implementation, ensuring that model behavior is identical to the original. This migration also positions `rclip` to adopt fully reproducible, publicly trained checkpoints in the future.

Figure 2 shows the model initialization code, which is where the key differences from the original implementation are concentrated. The original version used `clip.load()` to instantiate the model on CPU; the current version uses `open_clip.create_model_and_transforms()` with an explicit device parameter that enables MPS acceleration (described in Section IV-B). The feature computation methods themselves (`compute_image_features` and `compute_text_features`) follow the same pattern as the original implementation: preprocess, encode, normalize. The notable difference is in text feature computation (Figure 3), which uses a separate text-only model (`self._model_text`) instead of the full vision-language model, as described in Section IV-D.

B. GPU Acceleration with MPS

A significant performance improvement in the new version of `rclip` is the support for GPU-accelerated inference

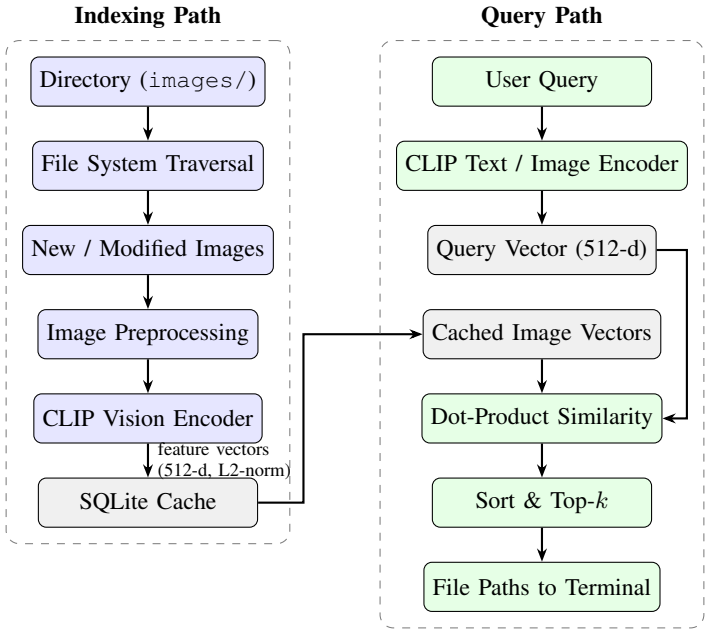

Figure 1. rclip system pipeline. The *indexing path* (left) traverses the directory, filters new or modified images, runs them through the CLIP vision encoder, and stores the resulting 512-dimensional L2-normalized feature vectors in a SQLite cache. The *query path* (right) encodes the user's text or image query with the CLIP encoder, retrieves cached image vectors from SQLite, computes dot-product (cosine) similarity, and returns the top-k matching file paths.

```
def _load_model(self):
  self._model, _, self._preprocess = (
    open_clip.create_model_and_transforms
        (
      self._model_name,
      pretrained=self._checkpoint_name,
      device=self._device,
    ))
```

Figure 2. Model initialization using `open_clip` with device selection.

```
def compute_text_features(self, text):
  with torch.no_grad():
    tokenized_text = self._tokenizer(text
        ).to(self._device)
    text_features = self._model_text.
        encode_text(tokenized_text)
    text_features /= text_features.norm(
        dim=-1, keepdim=True)
  return text_features.cpu().numpy()
```

Figure 3. Text vector computing method using a separate text-only model (`self._model_text`).

through Apple's MPS backend [5]. On macOS systems with Apple Silicon, `rclip` automatically detects MPS availability and uses it as the default compute device. Users can override this behavior by passing the `--device cpu` flag to force CPU-only execution.

The device selection logic is straightforward: `rclip` checks whether the system is running macOS and whether the MPS backend is available in PyTorch. When MPS is available, all model parameters and input tensors are moved to the MPS device, and the GPU handles the computationally intensive CLIP encoding operations. The unified memory architecture of Apple Silicon means that there is no explicit data transfer overhead between CPU and GPU memory, which is particularly beneficial for the image preprocessing pipeline where images are loaded and preprocessed on the CPU before being encoded on the GPU.

The performance impact of MPS acceleration is quantified in Section V.

C. Batched Indexing

The original version of `rclip` processed images in fixed batches of 8 during indexing. The current version introduces a configurable batch size, controllable through the `--indexing-batch-size` command-line argument so that users can tune it based on their available memory and hardware capabilities.

Larger batch sizes improve throughput by amortizing the fixed overhead of model invocation across more images. However, they also increase peak memory consumption, as all images in a batch must be held in memory simultaneously. The configurable batch size allows users to find the optimal tradeoff for their specific hardware configuration.

D. Text-Only Model Loading

A notable optimization in the new version of `rclip` is the ability to load only the text transformer portion of the CLIP model when performing queries that do not require image indexing. The full CLIP model includes both a vision

transformer and a text transformer, but when a user passes the `--no-indexing` flag (indicating that no new images need to be indexed), only the text transformer is needed to compute query vectors.

The text-only model is extracted from the full model by creating a deep copy and removing the visual component. This stripped-down model is then serialized to disk using PyTorch's `torch.save` mechanism, along with a version file that tracks the `open_clip` library version used to create it. On subsequent runs, if the `open_clip` version has not changed, `rclip` loads the cached text-only model directly, bypassing the need to download and instantiate the full vision-language model. This optimization significantly reduces startup time for search-only queries, as the text transformer is substantially smaller than the full model.

Table I quantifies the impact of this optimization. The measurements were performed on the test image set included in the `rclip` repository. The text-only model reduces query time by approximately 50% and memory consumption by 3.78×. The RAM reduction comes primarily from not loading the vision transformer and is independent of the number of indexed images. The query time improvement may be less pronounced for larger datasets, where loading cached image vectors from disk becomes the dominant cost.

E. Expanded Image Format Support

The original version of `rclip` supported only JPEG and PNG image formats. The current version adds support for WebP, High Efficiency Image Container (HEIC), and RAW formats (ARW and CR2). HEIC support is enabled through the `pillow-heif` library [22], which registers a HEIF opener with Pillow [23]. RAW image support is provided as an experimental feature, enabled via the `--experimental-raw-support` flag. When RAW support is active, `rclip` intelligently skips RAW files that have a corresponding processed image (e.g., a JPEG alongside an ARW file), avoiding duplicate entries in the index.

F. Improved File System Traversal

The original implementation used Python's `os.walk` function to traverse directories, which requires a separate `os.path.getmtime` and `os.path.getsize` call for each file to obtain its metadata. The current version uses `os.scandir`, which returns `DirEntry` objects that cache file metadata from the initial directory listing, avoiding redundant system calls.

Additionally, the new implementation runs a separate thread to count the total number of image files in the directory tree, enabling a progress bar with accurate completion percentage during indexing. The directory traversal itself uses an iterative stack-based approach rather than the recursive `os.walk`, giving `rclip` more control over the traversal order and enabling the exclusion of directories (such as `.git` and `node_modules`) at the directory entry level rather than after recursion.

TABLE I. Impact of text-only model loading on search-only queries.

Metric	Full Model	Text-Only	Improvement
Query time	3.8s	1.9s	2× faster
RAM usage	1.7 GB	0.45 GB	3.78× less

```
self._con.execute("""
  CREATE TABLE IF NOT EXISTS images (
    id INTEGER PRIMARY KEY,
    deleted BOOLEAN,
    indexing BOOLEAN,
    filepath TEXT NOT NULL UNIQUE,
    modified_at DATETIME NOT NULL,
    size INTEGER NOT NULL,
    vector BLOB NOT NULL
  )
""")
```

Figure 4. Updated structure of the "images" table (schema version 2).

G. Database Schema Evolution

`rclip` features the image vector cache implemented using the `SQLite 3` relational database management system (RDBMS) [24]. The image vector is computed and added to the index only if the cache does not already contain an entry for a given image. The vectors are cached by image paths. This allows the user to execute repeated queries over the same image catalog without waiting for the image vectors to be recomputed.

The database schema has evolved from version 1 to version 2 since the original publication. Figure 4 shows the current schema. The most notable addition is the `indexing` column, which is used to track the indexing state of images during the re-indexing process. Previously, the `deleted` flag was set on all images before re-indexing and then cleared as each image was confirmed to still exist. This approach was problematic because it briefly marked all images as deleted, which could cause issues if the process was interrupted. The new approach uses a separate `indexing` flag: images are marked as "indexing" at the start, the flag is cleared as each image is confirmed, and any images still flagged as "indexing" at the end are marked as deleted. This ensures that the index remains consistent even if the process is interrupted.

The database module also includes a version migration system that automatically applies schema changes when a user upgrades `rclip`. When the stored schema version is lower than the expected version, the migration logic adds the new `indexing` column via an `ALTER TABLE` statement and updates the version number.

When the user executes the `rclip` command, the tool first recursively processes the current directory and all its subdirectories, finds all of the images in them, and, if the image is not already present in the cache, computes their feature vectors and saves them to the SQLite cache database. This step is quick on repeated queries because the images are

already cached, but if the user wants to skip the process of checking that the cache is up-to-date, they can pass the `-n` argument to `rclip` to skip the indexing step completely and speed up the `rclip` command execution even more.

Then, `rclip` computes the query vector, fetches from the cache database image vectors for all of the images located in the current directory, computes the similarity score between the query vector and each of the image vectors, sorts the scores by the decreasing order, and gets the first k images with the highest similarity scores.

Finally, `rclip` prints the paths to the images that match the query to the terminal. Users can then open the images in their favorite image viewer or editor.

The `rclip` source code is published on GitHub under the MIT license [21].

V. Performance

This section presents performance benchmarks for the current version of `rclip`, including comparisons with the original version and the impact of MPS GPU acceleration.

A. Model Comparison on Low-End Hardware

As reported in the original conference paper, `rclip` was benchmarked using two different CLIP models, `ViT-B/32` (smaller CLIP model) and `ViT-L/14@336px` (larger CLIP model), on a NAS running `Intel(R) Celeron(R) CPU J3455 @ 1.50GHz`. Table II shows how `rclip` performs when indexing and searching through 269 photos when running on this CPU.

As Table II shows, the `ViT-L/14@336px` performance will not scale well, which makes `rclip` unusable in practical scenarios when running CLIP on low-level and mid-level consumer CPUs. This is why `rclip` uses `ViT-B/32`.

Running `rclip` indexing with `ViT-B/32` on 72,769 photos on the same NAS powered by `Intel(R) Celeron(R) CPU J3455 @ 1.50GHz` took 23 hours. Performing a query over 72,769 photos takes 56 seconds.

B. MPS GPU Acceleration

Table III presents the updated performance measurements on an `Apple M1 Max`, comparing MPS-accelerated indexing against CPU-only indexing on the ImageNet-1k 1.28 million image train set.

The MPS backend provides a 2.15× speedup for indexing 1.28 million images. This improvement comes from offloading the CLIP vision transformer forward pass to the Apple Silicon GPU. The CPU-only indexing rate of 73.1 images per second still represents a significant improvement over the original version, which achieved approximately 42 images per second on the same CPU (indexing 1.28 million images in 8 hours 31 minutes, as reported in the original paper). This additional improvement is primarily attributable to the improved file system traversal mechanism (as described in Section IV-F) that uses `os.scandir` and optimizes the directory search loop.

C. Performance Improvements Summary

Table IV provides a comprehensive comparison of the performance characteristics between the original version of `rclip` (as described in the conference paper) and the current version, all measured on the ImageNet-1k 1.28 million image train set using the `Apple M1 Max`.

The combined effect of file system traversal optimizations and MPS GPU acceleration yields a 3.76× improvement in indexing throughput compared to the original version. Even without GPU acceleration, the CPU-only current version achieves a 1.75× improvement, demonstrating that the software-level optimizations (specifically the `os.scandir` traversal mechanism) contribute meaningfully independent of hardware acceleration.

D. Scalability

To give a better understanding of how `rclip` performance scales, Table V shows how `rclip` performs when indexing and searching through 50k images and 1.28m images on the `Apple M1 Max` CPU. As the Table V shows, the indexing time scales linearly with the number of images when the search time increases only slightly even when going from searching through 50 thousand images to searching through 1.28 million images.

It should be noted that real-life `rclip` application possibilities go far beyond results demonstrated in these benchmarks because `rclip` can be used to execute any queries and not just the ones present in the dataset labels.

VI. Search Quality

This section presents corrected and expanded search quality measurements. An important erratum regarding the original benchmark results is discussed first, followed by the corrected measurements.

A. Erratum: Original ImageNet-1k Benchmark

The original conference paper [1] reported a top-1 accuracy of 31.17% and a top-5 accuracy of 44.80% on the ImageNet-1k [25] 1.28 million images train set. These numbers were incorrect due to a bug in the benchmark code.

The bug was located in the original benchmark script [26]. In the original code, the number of classes was inadvertently

TABLE II. Indexing and search performance on `Intel(R) Celeron(R) CPU J3455 @ 1.50GHz` using different CLIP models.

Model	Indexing Time	Search Time
`ViT-B/32`	3m56.626s	0m18.064s
`ViT-L/14@336px`	125m0.507s	3m19.742s
Difference	x31.70	x11.06

TABLE III. Indexing performance on `Apple M1 Max` using `ViT-B-32-quickgelu`: MPS vs CPU.

Device	Indexing Time	Throughput	Speedup
MPS (GPU)	2h16m01s	157.0 img/s	2.15×
CPU only	4h52m12s	73.1 img/s	1.00×

TABLE IV. Summary of performance improvements: original vs. current `rclip` on ImageNet-1k 1.28m images (Apple M1 Max).

Metric	Original Version	Current (CPU)	Current (MPS)
CLIP library	OpenAI `clip`	`open_clip`	`open_clip`
Indexing time (1.28m images)	8h 31m	4h 52m	2h 16m
Indexing throughput	∼41.7 img/s	73.1 img/s	157.0 img/s
Speedup vs. original	1.00×	1.75×	3.76×
Text-only model loading	No	Yes	Yes
Batch size	Fixed (8)	Configurable	Configurable
File system traversal	`os.walk`	`os.scandir`	`os.scandir`

derived from `image_features.shape[0]`, which returns the feature vector dimension (512 for `ViT-B/32`), not the number of ImageNet classes (1000). The label-to-class mapping was then constructed using Python's `zip` function, which stops at the shorter iterable. Since the feature dimension (512) is smaller than the number of classes (1000), `zip` silently dropped 488 of the 1000 ImageNet classes from consideration. This meant that every top-1 and top-5 accuracy measurement was computed over only 512 classes instead of the full 1000, systematically underreporting the accuracy.

The CIFAR-100 benchmark results (55.15% top-1, 81.34% top-5) reported in the original paper were not affected by this bug, as the CIFAR-100 benchmark used a different code path that correctly handled all 100 classes.

B. Corrected Results

Table VI presents the corrected search quality measurements using the fixed benchmark code. The results are shown for `ViT-B-32-quickgelu`, the model used by the current version of `rclip`.

The corrected top-1 accuracy on ImageNet-1k is 58.78%, nearly twice the originally reported 31.17%.

C. Context for the Results

It is important to note that `rclip`'s search quality evaluation differs from the standard zero-shot classification approach used in the original CLIP paper. In the CLIP paper, zero-shot classification is performed by computing class description vectors using prompt templates such as "a photo of a {class}"/"a blurry photo of a {class}"/"a bad photo of a {class}" and selecting the class with the highest cosine similarity to the image vector. The original CLIP paper reported 63.2% zero-shot top-1 accuracy on ImageNet with `ViT-B/32`.

`rclip`'s benchmark follows a similar approach but uses a simpler prompt template ("photo of {class}") without the ensemble of prompt templates used in the original CLIP paper, which employed 80 different prompt templates and averaged their embeddings. This methodological difference partially explains the gap between `rclip`'s 58.78% and CLIP's reported 63.2%. Additionally, `rclip` benchmarks against the 1.28 million image train set rather than the 50,000 image validation set, and the train set contains images with varying quality and ambiguity.

TABLE V. Indexing and search performance of the original version of `rclip` on `Apple M1 Max` CPU using `ViT-B/32`.

Dataset	# of images	Indexing Time	Search Time
`ImageNet-1k` validation set	50k	19m24.750s	0m4.04s
`ImageNet-1k` train set	1.28m	8h31m26.680s	0m11.49s
Difference	x25.62	x26.35	x2.84

TABLE VI. Corrected `rclip` search quality.

Dataset	Top-1	Top-5
`ImageNet-1k` 1.28m	58.78%	84.21%
`CIFAR-100` 10k	55.15%	81.34%

D. Qualitative Examples

To get a better understanding of `rclip`'s performance, see Figure 6, Figure 7, and Figure 8 showing search results for a search performed on an unlabeled demo set of 142 images. Figure 5 gives a glimpse into the set and shows that there is a variety of different images present there.

These qualitative results show that `rclip` returns visually and semantically coherent matches for both concrete object queries (Figure 6, "cat") and abstract compositional queries (Figure 7, "leading lines"), and that it also handles longer descriptive prompts effectively (Figure 8).

VII. Comparison with Other CLIP-Based Tools

Several other tools have been developed to leverage CLIP for image search. This section compares `rclip` with the most notable ones and discusses the design philosophy differences that distinguish these approaches.

`clip-retrieval` [27] is a comprehensive toolkit that provides components for computing CLIP embeddings (`clip inference`), building searchable indices (`clip index`), hosting them as a web service (`clip back`), and querying them through a web frontend (`clip front`). It is designed primarily for large-scale, server-based deployments and integrates with FAISS for approximate nearest neighbor search. In contrast, `rclip` is designed as a lightweight, single-user command-line tool that runs entirely locally and requires no server infrastructure. The `clip-retrieval` approach is better suited for organizations that need to serve multiple concurrent users over a shared image collection, while `rclip` targets individual users who want to quickly search

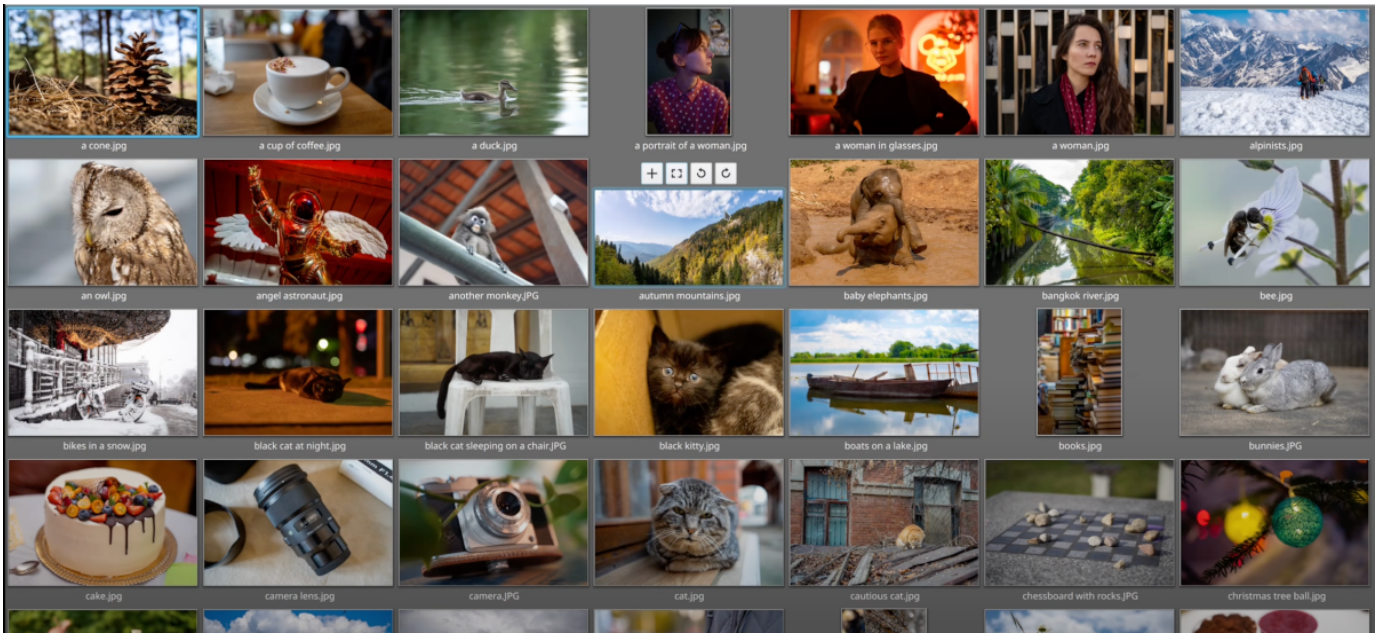

Figure 5. Demo set sample.

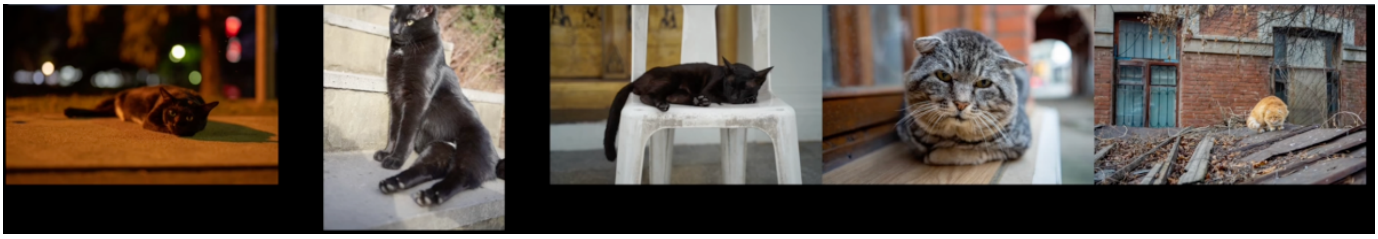

Figure 6. Search result for query “cat”.

Figure 7. Search result for query “leading lines”.

Figure 8. Top result for query “a kitten peeking from behind a corner”.

their personal photo libraries without setting up additional infrastructure.

CLIPPyX [28] is an AI-assisted desktop search tool that supports content-based, text, and visual similarity searches across local and external drives, storing embeddings in a vector database. While it provides a graphical user interface, `rclip` focuses on command-line simplicity and integration with existing terminal workflows. The GUI-based approach of CLIPPyX may be more accessible to non-technical users, but `rclip`’s command-line interface enables scripting, piping results to other tools, and integration into automated workflows—capabilities that are important for power users and developers.

CLIPSE (CLIP-based Image Search Engine) [29] is a minimalistic, research-oriented search engine built with Python. It

is designed for simplicity and easy extension, primarily aimed at researchers rather than end users. Unlike `rclip`, CLIPSE separates the indexing process (via a `build_index.py` script) from the search interface, supporting both CLI queries and a web interface. `rclip`, on the other hand, integrates these workflows into a single, zero-configuration executable aimed at everyday use without requiring users to manage separate indexing scripts or virtual environments.

Table VII summarizes the key differences between these tools.

`rclip`'s distinguishing characteristics are its focus on local, command-line usage; its support for incremental indexing that efficiently handles evolving image collections; its MPS GPU acceleration for Apple Silicon; its text-only model loading for fast repeated queries; and its support for query arithmetic and weighted queries. The design reflects a deliberate choice to prioritize simplicity and low friction: a user can install `rclip` via `pip`, navigate to any directory, and immediately search their images without any configuration, database setup, or server management.

VIII. Discussion

This section discusses the practical implications of the results presented in this paper, the limitations of the current approach, and lessons learned during the development and evolution of `rclip`.

A. Practical Considerations for Deployment

`rclip`'s architecture makes it particularly well-suited for several real-world deployment scenarios. For individual users managing personal photo collections, the tool provides an immediate and intuitive way to find specific images without requiring manual tagging or organization. The incremental indexing mechanism means that once a collection is initially indexed, subsequent searches incur minimal overhead even as new photos are added.

For creative professionals such as photographers, graphic designers, and video editors, the ability to search by abstract concepts ("leading lines," "warm tones," "minimalist composition") rather than concrete objects offers a workflow advantage that traditional keyword-based search cannot provide. The query arithmetic feature further enables retrieval of images that match complex aesthetic criteria, such as searching for images that are "similar to this reference photo but without people."

In organizational settings, `rclip` can serve as a lightweight alternative to commercial digital asset management (DAM) systems for teams that need semantic search capabilities without the overhead of deploying and maintaining a full-featured DAM platform. The command-line interface also facilitates automation: `rclip` can be integrated into shell scripts for batch processing, automated curation, or content moderation workflows.

Another practical application is in security and compliance contexts, where organizations need to identify images containing specific content across large file systems. Because `rclip` works with natural language queries and does not require pre-trained classifiers for each content category, it can be quickly deployed to search for new categories of content without retraining.

B. Limitations

Despite the improvements described in this paper, `rclip` has several limitations that users should be aware of.

Model limitations. The search quality is fundamentally bounded by the capabilities of the underlying CLIP model. `ViT-B/32` is one of the smaller CLIP models, and while it offers a good balance of speed and quality, larger models such as `ViT-L/14@336px` would provide better search results at the cost of significantly longer indexing times and higher memory consumption. As shown in Table II, the larger model is approximately 31× slower for indexing on CPU, making it impractical for large collections on consumer hardware. CLIP models also have known biases inherited from their training data, which can affect search results for certain query categories.

Scalability ceiling. While `rclip`'s brute-force similarity search is acceptable for collections up to a few million images, it becomes increasingly slow for much larger collections. The search time scales linearly with the number of indexed images because every image vector must be loaded and compared on each query. For collections exceeding 5–10 million images, integrating an approximate nearest neighbor index (as discussed in Section II-D) would be necessary.

Cache invalidation. `rclip` identifies images by their file path, modification time, and file size. This means that renaming or moving an image file causes it to be treated as a new image and re-indexed, even though its content has not changed. A content-based hashing mechanism could address this limitation but would add overhead to the indexing process.

Single-machine design. `rclip` is designed as a single-machine tool with a local SQLite database. It does not support distributed indexing, shared caches across machines, or concurrent access from multiple users. For multi-user or multi-machine deployments, a tool like `clip-retrieval` [27] with a client-server architecture would be more appropriate.

Image preprocessing. CLIP's image preprocessing pipeline resizes and crops input images to a fixed resolution (224×224 pixels for `ViT-B/32`). This means that fine-grained details in high-resolution images may be lost during preprocessing, potentially affecting search quality for queries that depend on small details within larger images.

IX. Conclusion and Future Work

This extended journal paper has presented the substantial evolution of `rclip`, a command-line image search tool based on CLIP, since its original conference publication. The key contributions of this extended work include:

- The migration from OpenAI's `clip` library to the open-source `open_clip` library, positioning `rclip` to adopt fully reproducible, publicly trained checkpoints in the future.

TABLE VII. Comparison of CLIP-based image search tools.

Feature	rclip	clip-retrieval	CLIPPyX	CLIPSE
Interface	CLI	Web + CLI	GUI	CLI + Web
Deployment	Local	Server	Local	Local
ANN Index	No	Yes (FAISS)	Yes	No
MPS / Apple Silicon Support	Yes	No	Yes (MLX)	No
Incremental Indexing	Yes	No	Yes	No
Query Arithmetic	Yes	No	No	No
Multimodal Queries	Yes	Yes	Yes	No
RAW Image Support	Yes	No	No	No
Text-Only Model Loading	Yes	No	No	No
Persistent Cache	Yes (SQLite)	Yes (various)	Yes	Yes (JSON)

- The introduction of GPU acceleration via Apple's MPS backend, achieving a 2.15× speedup for indexing 1.28 million images on Apple M1 Max.
- A correction to the ImageNet-1k benchmark results: the original paper reported 31.17% top-1 accuracy due to a benchmark bug that silently dropped 488 of 1000 classes; the corrected accuracy is 58.78% with `ViT-B-32-quickgelu`, which uses the original OpenAI CLIP weights.
- Detailed descriptions of implementation improvements including batched indexing, text-only model loading, expanded image format support, improved file system traversal, and database schema evolution.
- A comparison with other CLIP-based image search tools, situating `rclip`'s design choices within the broader ecosystem.

Future work includes:

- Enriching `rclip`'s search capabilities by combining CLIP embeddings with existing image metadata such as GPS coordinates, capture date, and camera model.
- Exploring the integration of approximate nearest neighbor search (e.g., FAISS) for sub-second query performance on collections exceeding tens of millions of images.
- Exploring the use of larger CLIP models (e.g., `ViT-L/14@336px`) on systems with dedicated GPUs, trading increased computational cost for higher search quality.
- Investigating the impact of image corruption, compression artifacts, and unusual aspect ratios on search quality.
- Adding CUDA GPU support for systems with NVIDIA GPUs, which would further reduce indexing times on Linux and Windows workstations.
- Evaluating the text-only model loading optimization on larger datasets to quantify how the relative speed improvement scales as cached vector loading becomes the dominant cost.

In summary, this paper presents a practical and scalable method of searching images using natural language queries based on the CLIP model, details its architectural evolution over multiple release cycles, provides corrected benchmark results, and demonstrates that GPU acceleration on consumer hardware can significantly improve the tool's usability for large image collections. Taken together, these results confirm that large personal image collections can be searched by natural language on consumer hardware through a command-line interface, with no cloud infrastructure or specialized servers required.

References

[1] Y. Mikhalevich, "Using text queries to look up unlabeled images: A command-line search tool based on CLIP," in *Proceedings of CONTENT 2023, The Fifteenth International Conference on Creative Content Technologies*, IARIA, 2023, pp. 7–11, ISBN: 978-1-68558-048-3.

[2] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, et al., "Learning transferable visual models from natural language supervision," arXiv, 2021. Accessed: Mar. 5, 2026. [Online]. Available: https://arxiv.org/abs/2103.00020

[3] M. Cherti, R. Beaumont, R. Wightman, M. Wortsman, G. Ilharco, C. Gordon, et al., "Reproducible scaling laws for contrastive language-image learning," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2023, pp. 2818–2829. DOI: 10.1109/cvpr52729.2023.00276

[4] C. Schuhmann, R. Beaumont, R. Vencu, C. Gordon, R. Wightman, M. Cherti, et al., "Laion-5b: An open large-scale dataset for training next generation image-text models," 2022. arXiv: 2210.08402 [cs.CV]. Accessed: Mar. 5, 2026. [Online]. Available: https://arxiv.org/abs/2210.08402

[5] Apple Inc., *Accelerated PyTorch training on Mac*, 2026. Accessed: Mar. 5, 2026. [Online]. Available: https://developer.apple.com/metal/pytorch/

[6] N. Fiedler, M. Bestmann, and N. Hendrich, "Imagetagger: An open source online platform for collaborative image labeling," in *RoboCup 2018: Robot World Cup XXII 22*, Springer, 2019, pp. 162–169.

[7] J. Tesic, "Metadata practices for consumer photos," *IEEE MultiMedia*, vol. 12, no. 3, pp. 86–92, 2005.

[8] J. Kim, S. Lee, J.-S. Won, and Y.-S. Moon, "Photo cube: An automatic management and search for photos using mobile smartphones," in *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, IEEE, 2011, pp. 1228–1234.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[10] B. N. Lee, W.-Y. Chen, and E. Y. Chang, "A scalable service for photo annotation, sharing, and search," in *Proceedings of the 14th ACM international conference on Multimedia*, 2006, pp. 699–702.

[11] J. Zhang, A. Hallquist, E. Liang, and A. Zakhor, "Location-based image retrieval for urban environments," in *2011 18th IEEE International Conference on Image Processing*, IEEE, 2011, pp. 3677–3680.

[12] V. Balakrishnan and E. Lloyd-Yemoh, "Stemming and lemmatization: A comparison of retrieval performances," *Lecture Notes on Software Engineering*, vol. 2, no. 3, pp. 262–267, 2014.

[13] M. Günther, "Freddy: Fast word embeddings in database systems," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 1817–1819.

[14] T. Kenter and M. De Rijke, "Short text similarity with word embeddings," in *Proceedings of the 24th ACM international on conference on information and knowledge management*, 2015, pp. 1411–1420.

[15] M. M. B. Ismail, *Image annotation and retrieval based on multi-modal feature clustering and similarity propagation*. University of Louisville, 2011.

[16] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021. Accessed: Mar. 5, 2026. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy

[17] C. Jia, Y. Yang, Y. Xia, Y.-T. Chen, Z. Parekh, H. Pham, et al., "Scaling up visual and vision-language representation learning with noisy text supervision," in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 18–24 Jul 2021, pp. 4904–4916. Accessed: Mar. 5, 2026. [Online]. Available: https://proceedings.mlr.press/v139/jia21b.html

[18] J. Li, D. Li, C. Xiong, and S. Hoi, "BLIP: Bootstrapping language-image pre-training for unified vision-language understanding and generation," in *Proceedings of the 39th International Conference on Machine Learning*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., ser. Proceedings of Machine Learning Research, vol. 162, PMLR, 17–23 Jul 2022, pp. 12 888–12 900. Accessed: Mar. 5, 2026. [Online]. Available: https://proceedings.mlr.press/v162/li22n.html

[19] X. Zhai, B. Mustafa, A. Kolesnikov, and L. Beyer, "Sigmoid loss for language image pre-training," 2023. arXiv: 2303.15343 [cs.CV]. Accessed: Mar. 5, 2026. [Online]. Available: https://arxiv.org/abs/2303.15343

[20] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, vol. 7, pp. 535–547, 2017. Accessed: Mar. 5, 2026. [Online]. Available: https://api.semanticscholar.org/CorpusID:926364

[21] Y. Mikhalevich, *rclip*, version 2.0, 2025. Accessed: Mar. 5, 2026. [Online]. Available: https://github.com/yurijmikhalevich/rclip

[22] A. Piskun and contributors, *Pillow-heif: Python library for reading and writing HEIF images*, https://github.com/bigcat88/pillow_heif, 2026. Accessed: Mar. 5, 2026. [Online]. Available: https://github.com/bigcat88/pillow_heif

[23] A. Murray, H. van Kemenade, wiredfool, J. A. Clark, A. Karpinsky, O. Baranovič, et al., *Python-pillow/pillow: 12.1.1*, version 12.1.1, Feb. 2026. DOI: 10.5281/zenodo.18604291 Accessed: Mar. 5, 2026. [Online]. Available: https://doi.org/10.5281/zenodo.18604291

[24] R. D. Hipp, *SQLite*, version 3.31.1, 2020. Accessed: Mar. 5, 2026. [Online]. Available: https://www.sqlite.org/index.html

[25] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2009, pp. 248–255.

[26] Y. Mikhalevich, *rclip benchmark script (ImageNet) – original (buggy) revision*, 2023. Accessed: Mar. 5, 2026. [Online]. Available: https://github.com/yurijmikhalevich/rclip/blob/f862d9a/benchmarks/benchmark-imagenet.py

[27] R. Beaumont, *Clip retrieval: Easily compute clip embeddings and build a clip retrieval system with them*, https://github.com/rom1504/clip-retrieval, 2022. Accessed: Mar. 5, 2026. [Online]. Available: https://github.com/rom1504/clip-retrieval

[28] U. Ahmed and contributors, *CLIPPyX: AI powered search tool offers content-based, text, and visual similarity system-wide search*, https://github.com/0ssamaak0/CLIPPyX, 2026. Accessed: Mar. 5, 2026. [Online]. Available: https://github.com/0ssamaak0/CLIPPyX

[29] S. Göring, *Clipse – a minimalistic clip-based image search engine for research*, 2025. arXiv: 2504.17643 [cs.CV]. Accessed: Mar. 5, 2026. [Online]. Available: https://arxiv.org/abs/2504.17643