# Overview of the SISAP 2023 Indexing Challenge

Eric S. Tellez[1,2], Martin Aumüller[3], and Edgar Chavez[2]

[1] eric.tellez@ieee.org INFOTEC, México; IxM CONACyT, México.
[2] elchavez@cicese.mx CICESE, Ensenada, Baja California, México.
[3] maau@itu.dk ITU Copenhagen, Denmark.

**Abstract.** This manuscript presents the premiere **SISAP 2023 Indexing Challenge**, which seeks replicable and competitive solutions in the realm of approximate similarity search algorithms. Our aim is recall, all while optimizing build time, search time, and memory consumption. Using a subset of the deep features of a neural network model provided by the LAION-5B dataset, the challenge posed three tasks, each with its unique focus:

- **Task A**: Conduct classical approximate nearest neighbor search, ensuring an average recall of at least 0.9 for 10-NN queries.
- **Task B**: Find a succinct binary embedding of the original data that ensures high recall on the original data.
- **Task C**: Index and search binary representations from Task B.

Notably, an innovative and competitive binary mapping method emerged from the challenge. It also spotlighted graph methods as the preferred indexing technique for binary and real-valued high-dimensional vectors. However, these methods have little room for improvement. Enhancing memory efficiency, refining navigational strategies, and tackling the secondary memory challenge are pivotal next steps.

**Keywords:** Approximate nearest neighbor search · Indexing and searching pipelines · Experimental comparison of search methods.

## 1 Introduction

Similarity search algorithms are pivotal for efficiently retrieving similar items from vast datasets, underpinning tasks like information retrieval, multimedia indexing, and pattern recognition. As machine learning, dense retrieval, and computer vision become increasingly prevalent, similarity search methods must meet the quality and computational demands of both applications and the systems they run on.

The *curse of dimensionality* [3] dictates that all metric search algorithms falter when confronted with high-dimensional datasets. This matter necessitates adopting approximate or probabilistic methods to balance speed against quality. Furthermore, there's an opportunity to trade-off between construction time and memory usage, leading to a variety of indexing solutions, each with its merits and drawbacks.

The SISAP Indexing Challenge[4] seeks to identify efficient similarity search algorithms that strike a balance between accuracy and practical constraints like build time, search time, and memory consumption. To facilitate this, we devised a test bed utilizing the LAION deep features English subset, segmented into 10M, 30M, and 100M benchmarks. Additionally, there are two query sets: public and private, each comprising 10k vectors. The public queries were made available during the call for papers, while the private ones were revealed post-submission and evaluation.

## 2 The dataset

The LAION dataset, as detailed in [15], is an expansive public image collection comprising both images and textual descriptors. It has proven instrumental in training large visual and language deep-neural models, as cited in [4,13]. Every image within the collection is paired with a URL handle, simplifying the demonstration process. Moreover, the LAION consortium has made vector embeddings available using the Contrastive Language-Image Pre-Training (CLIP), specifically harnessing the OpenCLIP architecture [4]. These deep features manifest as 768-dimensional vectors, represented using 16-bit floating point numbers. The CLIP architecture was initially introduced in [12].

We employed three subsets from the English segment of the LAION collection (commonly referred to as LAION2B) as benchmarks. These subsets consist of 10, 30, and 100 million vectors, with vectors labeled as *Not Safe for Work* (NSFW) duly excluded. Further insights regarding the selection and packaging of these subsets can be found on the challenge's companion site.

## 3 Task Descriptions

The Indexing Challenge focuses on nearest neighbor queries, specifically on approximate $k$ nearest neighbor queries. We have established three tasks that emulate various application scenarios, each catering to different needs in terms of quality, speed, and memory.

A key aspect of this challenge is reproducibility. Submissions were accepted in the form of Github repositories with operational Github Action (GHA) workflows.[5] Teams crafted their solutions by meticulously setting and benchmarking hyperparameters for each task and clearly detailing their choices in their GHA entry point.

We designed four benchmarks from the LAION2B dataset, each with a distinct number of vectors: 300K for development and 10M, 30M, and 100M workloads designated for the challenge. Furthermore, we designed two sets of public

[4] Official site of the challenge `https://sisap-challenges.github.io/`.

[5] Github Actions is a continuous integration platform that enables continuous testing of repositories within virtual machines.

and private queries. Teams were tasked with designing their solutions and determining the hyperparameters based on the public query set. The private set was subsequently used to re-test and rank all solutions on our system. We computed gold standards for $k$ nearest neighbor queries in public and private queries, which is the foundation for calculating the recall score in the final results. Public queries, along with their associated gold standards, were available from the commencement of the challenge, while private ones were unveiled post-validation. We expected teams to construct indexes that efficiently solve queries and excel under the specific conditions and metrics defined for each task. All tasks revolved around retrieving the approximate $k = 10$ nearest neighbors.[6]

During the challenge, Vladimir Míč (private communication) reported certain anomalies he detected in the public gold standard. He highlighted numerous distance value ties at $k$ and $k + 1$ neighbors and instances of neighbors at distance 0, i.e., duplicates. Upon confirmation, we ascertained that these discrepancies were likely due to the prevalence of near duplicates in the LAION database [20]. For the private query set, we implemented measures to curb these issues, utilizing IEEE 754 floating point arithmetic to compute distance functions in the gold standard and excluding query objects where the $k$ and $k + 1$ neighbors matched identically.[7] The subsequent segments of this section delve deeper into the intricacies of the tasks.

**Task A: Searching the Original Embeddings** Task A focuses on high-throughput solutions with little loss in quality. The aim is to design the fastest search algorithm that hits a recall of at least 0.9 (on average over all queries). Teams adopted the strategy to build a single index and used a large collection of search parameters for each subset size. A small catch is that only the best-performing probe in the *private query set* gets to stay. Repositories should be ready to run right out of the box with all the settings in place. Note that teams likely used the public query set to fine-tune their settings.

**Task B: Producing Binary Sketches** This task concerns the succinct representation of the original 768-dimensional real-valued vectors using fixed-length binary strings. These have a much smaller memory footprint and allow efficient distance calculations via SIMD instructions. The main goal of this task is to find embeddings that, using Hamming distance and a linear scan, produce a higher average recall than our baseline of 0.24. The baseline uses our current go-to method based on permutation binary sketches [18].

**Task C: Indexing and Searching on Binary Sketches** For Task C, the challenge seeks solutions that let us first index and then search using these binary sketches while using the Hamming distance as our measuring stick. Participants can use the embeddings they came up with in Task B or go with the baseline embeddings. The benchmark? The fastest solutions with a recall close to ours, meaning they should achieve or surpass a recall of 0.216, i.e., 90% of our baseline.

[6] Gold standards incorporate results up to $k = 1000$.

[7] This constraint was determined using the 100 million benchmark.

## 4 Solutions Overview

This section describes the set of solutions to the SISAP Indexing Challenge. The solutions use diverse programming languages: C++, Rust, Java, and some use Python as a wrapper language. One baseline uses C++ with Python wrappers, and the rest uses the Julia programming language. The teams used different techniques to tackle the challenge: graph-based indexes, hashing-based indexes, linear projections, reranking, combinatorial and numerical optimization, among others.

### 4.1 Baselines

We included three baselines to compare with previous work: BL-SearchGraph, BL-FAISS-HSNW, and Bruteforce. Note that the first two are far from trivial solutions. The rest of this section describes our baselines and explains its construction and searching hyperparameters.

**Bruteforce.** This is a straightforward solution. It is implemented as an exhaustive search using the `SimilaritySearch.jl` package. However, as with the rest of our baselines, this approach takes advantage of the multithreading capabilities of our running infrastructure. Unsurprisingly, a well-implemented brute force algorithm can improve more sophisticated algorithms when the intrinsic dimension of the data is high.

**BL-FAISS-HNSW.** This baseline uses the HNSW index from FAISS.[8] The Hierarchical Navigable Small World (HNSW) index, see [8], is a graph-based index using a hierarchical structure to navigate the graph efficiently. It is created iteratively, adding one new object at a time. The $i$th element is inserted by adding edges from the $i$th element to a set of $M$ approximate nearest neighbors using the graph containing the previous $i-1$ objects; the hierarchy is maintained throughout the construction. The search algorithm consists of navigating the graph greedily using two priority queues. The first is the result set of size *ef*, and the second is a candidate list to prioritize the navigation. The search is conducted in rounds: The not-yet-visited current closest point to the query is inspected at each round. The search finishes when it is impossible to improve the result set during the navigation. Due to its flexibility, the HNSW index is the *de-facto* standard in the industry; most vector databases also implement it. According to standard benchmarks [1], it is one of the faster metric indexes known. As a baseline, we set its parameters as follows. We set the $M = 32$ for all subsets and the *ef* parameter as 40 for construction. At the search stage, we probe with the following *ef* values: 32, 64, 128, 256, and 512.

[8] `https://faiss.ai/`.

**BL-SearchGraph.** This baseline uses the SearchGraph index from Julia's package `SimilaritySearch.jl`, see [17,16]. This index is a graph-based index similar to the HNSW, but instead of a hierarchy, it uses a small sample of disjoint neighbors to get fast navigation. The construction is also based on connecting the $i$th element with its neighbors, but it is simplified since there is no hierarchy. In contrast to HNSW, it uses variable-size neighborhoods using shrinking heuristics based on the Spatial Access Trees [10], with an upper bound defined as $M = O(\log i)$. It uses Beam Search (BS) as a search algorithm. The search stores candidates in a priority list of maximum size (beam size) and also limit what is considered to be inserted into the beam using a parameter $0 < \Delta < 2$; the result set is populated during the navigation, and the search finishes when the result set does not improve and the beam is empty. It supports single-pass automatic index optimization for a given quality score. It is a flexible alternative that supports automatic optimization and user-defined metrics, the latter due to Julia's just-in-time compiler. As a baseline, it was constructed with 0.9 as objective recall and a neighborhood size of $M = \log_{1.5} i$. During the search stage, we varied the optimized $\Delta$ parameter in the range $\Delta/1.05^2 \leq \Delta' < 2$ growing exponentially in a 1.05 factor.

### 4.2 Teams solutions

Six teams submitted a candidate for evaluation; one team (HIOB) targeted all three tasks, one team (SWANN) focused on indexing binary sketches (Task C), the remaining teams (UTokyo, CRANBERRY, LMI, HSP) focused on efficient retrieval in the standard setting (Task A). Teams used their implementations and modifications or tuning of well-known approximate nearest neighbor search libraries.

**UTokyo.** This approach proposes a pipeline of dimensionality reduction, database subsampling, and entry point optimization to solve Task A. The pipeline is designed for graph-based indexes and optimizes the computational requirements in terms according to specified accuracy, runtime, and memory requirements. It employs black-box optimization for parameter tuning. In particular, the authors optimized a Navigating Spreading-out Graph (NSG) [7] with a neighborhood of 32 as index for their submission. More details are given in [11].

**CRANBERRY.** This approach combines several search techniques in a three-stage pipeline: data partitioning, candidate filtering, and reranking. The input database is divided into a Voronoi partition. The search algorithm locates the nearest partitions to the query to retrieve a list of potentially similar vectors. Then, 512-bit sketches and 24-dimensional prefixes of vectors are used to reduce the candidate list; an early termination strategy accompanies the filtering. The list of candidates is reranked using the original distance, that is, the 768-dimensional CLIP vectors and the cosine distance. CRANBERRY is designed to solve Task A. More details are found in [9].

**SWANN.** This approach uses a collection of tries together with the bit-sampling locality-sensitive hashing scheme [2]. Their solution targets Track C (indexing binary vectors). During index building, each binary vector is hashed $K \cdot L$ times, and each block of length-$K$ bit strings is used to insert the vector into one out of $L$ tries. The query vectors are hashed and looked up in the trie during the search. If too few candidates are found on the leaf level, the search is dynamically expanded to cover larger parts of the trie. More details are given in [14].

**HIOB.** This approach is based on creating binary sketches of a vector database explicitly designed for cosine similarity. The binarization is made through hyperplanes, i.e., encoding where the vector lies. The random sample consensus inspires the initialization of the set of encoding hyperplanes, RANSAC [5]. Then, the encoders are refined by maximizing bit independence and bit balance for binary sketches on the unit sphere. The iterative optimization process improves sketch quality through orthogonalization and is made in small batches, similar to stochastic gradient descent. In each iteration, a displacement vector is computed to update some hyperplanes. The bit assignments are recomputed after each batch. In the search stage, the bit-vectors under hamming distance are used to calculate a candidate list of size $n^\star$, this process is computed with a brute force procedure since the authors found no improvement on using HNSW or faiss [19]. The candidate set is reranked with the original database objects and cosine distance to get the $k$ nearest neighbors.

We used the specified hyperparameters for each task and subset. For Task A and C, we used 256, 192, and 128 bit-vectors for 10M, 30M, and 100M subsets, respectively. The $n^\star$ parameter is increased with the dataset for Task A, going from more than a thousand to 60,000. Task B uses 1024-bit vectors with $n^\star = k$. Task C is similar to task A, but $n^\star = k$. More details are given in [19].

**LMI.** The Learned Metric Index uses an architecture of interconnected learned models; the LMI demonstrates notable performance characteristics, often surpassing traditional methods in terms of efficiency and effectiveness. Central to the LMI is a tree structure that harnesses machine learning, particularly neural networks, to shrink the search space, facilitating a sequential search among significantly fewer objects than the original dataset. This is followed by a more time-intensive bucket-level sequential search within identified data partitions. The distances between objects are ascertained through a trained neural network, resulting in a probability distribution matrix that captures object-category relationships. The approach is adaptable, with the procedure iterating over matrix columns based on similarity, treating the exact count as a parameter. The approach is more useful with the help of a GPU or TPU, which was not considered for this challenge.

**HSP.** The HSP team performed several modifications and tuning to the HNSW index, specifically on the *hnswlib*.[9] The authors reduced the memory requirements by hacking how the database is loaded and maintained in memory. This

[9] Official site of the hnswlib project `https://github.com/nmslib/hnswlib`.

**Table 1.** Task A results for all LAION subsets. Entries are sorted by best rank in the 10M subset. Query time is measured in seconds for the entire query set. OOM label means for *out of memory* and NR for *not run*.

| Team | 10M | | | 30M | | | 100M | | |
|---|---|---|---|---|---|---|---|---|---|
| | Build time | Search time | Rank | Build time | Search time | Rank | Build time | Search time | Rank |
| HSP | 1h 21m | 0.34 | 1 | 4h 16m | 0.49 | 1 | 17h 15m | 0.51 | 1 |
| UTokyo | 38m | 0.49 | 2 | 2h 35m | 0.71 | 2 | OOM | - | - |
| BL-SearchGraph | 13m | 0.61 | 3 | 53m | 1.09 | 4 | 5h 55m | 1.67 | 2 |
| BL-FaissHNSW | 16m | 0.74 | 4 | 33m | 0.86 | 3 | 4h 48m | 21.40 | 3 |
| HIOB | 7m | 35.89 | 5 | 8m | 89.97 | 5 | 13m | 247.01 | 4 |
| CRANBERRY | 1h 57m | 107.05 | 6 | 5h 49m | 192.02 | 6 | 17h 29m | 589.76 | 5 |
| LMI | 7h 4m | 450.25 | 7 | NR | - | - | NR | - | - |
| Bruteforce | 0m | 2,415.75 | 8 | 0m | 9,010.50 | 7 | NR | - | - |

change allowed them to reduce the construction time by half. Another customization removes unnecessary functionality directed to vector databases and other search engines. The authors performed a broad ablation study and hyperparameter optimization to obtain a competitive setup for the challenge. Interestingly, one of the most critical parameters is the construction *ef*, which interchanges construction time by search quality. Note that construction was previously reduced, resulting in a net moderate increase in the building time. The parameters $M = 20$ and *ef*= 800 (construction) of the HSNW were determined to be the best choices for the provided workloads. The search state iterates on different values *ef* from 10 to 1000. The HSP team designed their solution for Task A. More details are given in the accompanying paper [6].

## 5 Results and discussions

*Evaluation setup.* Following the GHA setup, we prepared docker Linux images. The evaluation was conducted on 2x Intel(R) Xeon(R) CPU E5-2690 V4 CPUs (28 cores, 56 hyperthreads) workstation with 512GiB of RAM. The original dataset resided on a 1TB SSD, but all solutions loaded data vectors and index data structures in memory. We encouraged participants to use multithreading or multiprocessing in the construction and searching stages to take advantage of the hardware—all participants except team SWANN employed multithreading. We enforced a time limit of 24 hours for building the index and running the query workload.

*Task A.* Table 1 shows the results of Task A. As mentioned in §3, all teams built a unique index. We recorded the build time and, for each set of hyperparameters, the search time accumulated over 10k queries. From these timings, we present the shortest search time that exceeded the recall requirement.

**Table 2.** Task B results. Recall values for 1024-dimensional bit-vectors.

| Team | Recall | | |
|---|---|---|---|
| | 10M | 30M | 100M |
| HIOB | 0.55 | 0.57 | 0.58 |
| Baseline | 0.24 | 0.24 | 0.25 |

**Table 3.** Task C results: Indexing binary vectors. The reference recall for bruteforce is around 0.24. Displayed are the top-performing parameters that surpass a recall of 0.216.

| | 10M | | | 30M | | | 100M | | |
|---|---|---|---|---|---|---|---|---|---|
| Team | Build time | Search time | Rank | Build time | Search time | Rank | Build time | Search time | Rank |
| BL-SearchGraph | 5m | 0.10 | 1 | 14m | 0.36 | 1 | 2h 6m | 1.09 | 1 |
| HIOB | 7m | 36.56 | 2 | 8m | 90.35 | 2 | - | - | - |
| Bruteforce | 0m | 74.51 | 3 | 0m | 246.95 | 3 | 0m | 816.93 | 2 |
| SWANN | 3m | 159.82 | 4 | 12m | 717.54 | 4 | 1h 3m | 3794.05 | 3 |

All solutions worked on the 10M subset, and this performance is used to sort the table; five worked on 30M, and only three on 100. The HSP team presents the top-performing solution in all subsets. It achieved search times below a second, which put it in the tens of thousands of queries per second. On the downside, it has one of the most costly constructions. UTokyo performs the second best, having a better trade with construction time but having memory issues and being unable to run the 100M benchmark. Focusing on build time, team HIOB has an order of magnitude shorter build times but provides rather slow searches. Entries marked as NR were not run due to diverse causes, like very high computational resources or not given hyperparameters.

*Task B.* Results of the second task are presented in Table 2. Here, the recall is used as the main performance score. Only the HIOB team participated, surpassing significantly our baseline by a factor of more than two. Interestingly, the 1024-dimensional bit-vectors are faster to compute and contain more metric information than approaches like PCA for the same given memory.

*Task C.* Table 3 shows the performance in Task C. Here, two teams were able to submit. The HIOB team uses the same configuration as in Task A but without reranking the results on the binary embedding using the original vectors. Since HIOB uses just 128-bit vectors for 100M, see §4, it achieves a lower recall behind the accepted threshold (AT). Using more bits improves their result quality at the cost of the search time. The SWANN team marked all entries beyond the AT, but its solution did not take advantage of the multicore architecture and was run in a single thread. Thus, they can improve their performance significantly if they solve queries in parallel.

## 6 Conclusions

The SISAP 2023 Indexing Challenge's first edition was a successful event that brought together researchers worldwide to work on the problem of approximate similarity search. The challenge becomes a trigger for innovative methods; several indexing methods for vector spaces emerged, along with binary mappings, and indexes for the binary Hamming space, as well as insights into the strengths and weaknesses of different approaches.

One of the most notable findings of the challenge was the emergence of a new binary mapping method that is both competitive and efficient. The challenge highlighted the importance of graph-based indexing techniques for real-valued and binary high-dimensional vectors. While the methods developed in the SISAP 2023 Indexing Challenge represent a significant step forward, there is still room for improvement. Future work should enhance memory efficiency, refine navigational strategies, and tackle the secondary memory challenge.

The SISAP 2023 Indexing Challenge was a valuable opportunity to advance the state of the art in approximate similarity search. The challenge's findings will interest researchers and practitioners working in various fields, including information retrieval, machine learning, and computer vision.

## References

1. Aumüller, M., Bernhardsson, E., Faithfull, A.J.: Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. Inf. Syst. **87** (2020)
2. Bawa, M., Condie, T., Ganesan, P.: LSH forest: self-tuning indexes for similarity search. In: WWW. pp. 651–660. ACM (2005)
3. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Comput. Surv. **33**(3), 273–321 (sep 2001). https://doi.org/10.1145/502807.502808, `https://doi.org/10.1145/502807.502808`
4. Cherti, M., Beaumont, R., Wightman, R., Wortsman, M., Ilharco, G., Gordon, C., Schuhmann, C., Schmidt, L., Jitsev, J.: Reproducible scaling laws for contrastive language-image learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 2818–2829 (2023)
5. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM **24**(6), 381–395 (1981)
6. Foster, C., Kimia, B.: Computational enhancements of hnsw targeted to very large datasets. In: Similarity Search and Applications: 16th International Conference, SISAP 2023, A Coruña Spain, October 9-11, Proceedings. Springer (2023)
7. Fu, C., Xiang, C., Wang, C., Cai, D.: Fast approximate nearest neighbor search with the navigating spreading-out graph. Proc. VLDB Endow. **12**(5), 461–474 (jan 2019). https://doi.org/10.14778/3303753.3303754, `https://doi.org/10.14778/3303753.3303754`
8. Malkov, Y.A., Yashunin, D.A.: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE transactions on pattern analysis and machine intelligence **42**(4), 824–836 (2018). https://doi.org/10.1109/tpami.2018.2889473

9. Mic, V., Sedmidubsky, J., Zezula, P.: CRANBERRY: Memory-Effective Search in 100M High-Dimensional CLIP Vectors. In: Similarity Search and Applications – 16th International Conference, SISAP 2023, Spain, Proceedings. pp. ??–?? (2023)
10. Navarro, G.: Searching in metric spaces by spatial approximation. The VLDB Journal **11**, 28–46 (2002)
11. Oguri, Y., Matsui, Y.: General and practical tuning method for off-the-shelf graph-based index: Sisap indexing challenge report by team utokyo. In: Similarity Search and Applications: 16th International Conference, SISAP 2023, A Coruña Spain, October 9-11, Proceedings. Springer (2023)
12. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al.: Learning transferable visual models from natural language supervision. In: International conference on machine learning. pp. 8748–8763. PMLR (2021)
13. Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-resolution image synthesis with latent diffusion models. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 10684–10695 (2022)
14. Schauser, T.H., Romild, C.J.W., Borup, J.A.: Enhancing approximate nearest neighbor search: Binary-indexed lsh-tries, trie rebuilding, and batch extraction. In: Similarity Search and Applications: 16th International Conference, SISAP 2023, A Coruña Spain, October 9-11, Proceedings. Springer (2023)
15. Schuhmann, C., Beaumont, R., Vencu, R., Gordon, C., Wightman, R., Cherti, M., Coombes, T., Katta, A., Mullis, C., Wortsman, M., et al.: Laion-5b: An open large-scale dataset for training next generation image-text models. Advances in Neural Information Processing Systems **35**, 25278–25294 (2022)
16. Tellez, E.S., Ruiz, G.: Similarity search on neighbor's graphs with automatic pareto optimal performance and minimum expected quality setups based on hyperparameter optimization. CoRR **abs/2201.07917** (2022), `https://arxiv.org/abs/2201.07917`
17. Tellez, E.S., Ruiz, G.: Similaritysearch.jl: Autotuned nearest neighbor indexes for julia. Journal of Open Source Software **7**(75), 4442 (2022)
18. Tellez, E.S., Chavez, E.: On locality sensitive hashing in metric spaces. In: Proceedings of the Third International Conference on SImilarity Search and APplications. pp. 67–74 (2010)
19. Thordsen, E., Schubert, E.: An alternating optimization scheme for binary sketches for cosine similarity search. In: Similarity Search and Applications: 16th International Conference, SISAP 2023, A Coruña Spain, October 9-11, Proceedings. Springer (2023)
20. Webster, R., Rabin, J., Simon, L., Jurie, F.: On the de-duplication of laion-2b. arXiv preprint arXiv:2303.12733 (2023)