Overview of the SISAP 2025 Indexing Challenge

Eric S. Tellez[1], Edgar Chavez[2], Martin Aumüller[3], and Vladimir Mic[4]

[1] eric.tellez@ieee.org INFOTEC; IxM SECIHTI, México.
[2] elchavez@cicese.edu.mx CICESE, México.
[3] maau@itu.dk ITU Copenhagen, Denmark.
[4] v.mic@cs.au.dk Aarhus University, Denmark.

Abstract. This paper summarizes the innovative solutions presented at the third edition of the SISAP Indexing Challenge held at SISAP 2025. The challenge featured two distinct tasks involving vector embeddings derived from a large corpus using neural encoders. It proposed the following two tasks under strict memory and computational constraints:

- **Task 1:** Approximate nearest neighbor search achieving an average recall of at least 0.7 for 30-NN, using out-of-distribution objects as queries.
- **Task 2:** k-NN ($k = 15$) graph construction for large datasets, requiring an average recall of at least 0.8.

Both tasks required solutions to operate within strict resource limits: 16 GB of RAM, 8 virtual CPUs, and a 12-hour wall-clock time for the end-to-end pipeline (including data loading, pre-processing, indexing, and searching). Each task imposes different minimum quality requirements and ranking specifications. Participants developed strategies such as data compression, optimized indexing, and efficient search algorithms to meet these constraints. This paper details the challenge design, explains the evaluation framework, and provides an overview of the submitted solutions.

1 Introduction

The SISAP 2025 Indexing Challenge[5] is an initiative that brings researchers and practitioners together to promote progress in similarity search. Historically, the SISAP Challenge has focused on evaluating k-nearest neighbor search algorithms and encoders under various conditions, using image embeddings from the LAION dataset [20,21]. The 2025 edition introduces two metric tasks: first, the indexing and retrieval of k-nearest neighbors (k-NN search), and second, the computation of the k-nearest neighbor graph (k-NN graph). Each task uses a distinct dataset, PUBMED23 and GOOAQ, resp., which consist of text passages encoded with Sentence-BERT models (see [19] for details on these models). In particular, the queries for Task 1 follow a different distribution than the dataset, presenting a realistic challenge for similarity search engines.

[5] Site: https://sisap-challenges.github.io/2025/index.html

Recent advances in deep learning have expanded the scope of similarity search beyond large-scale engines, integrating it into processing pipelines that run on standard hardware. Examples include workloads for Retrieval-Augmented Generation (RAG) [3] or k-NN graphs used in non-linear dimensionality reduction methods like UMAP [17].

The challenge's experimental environment was resource-constrained, designed for workloads that fit in memory or on disk. Thereby, it allows a trade-off between search speed and quality. This setup reflects common scenarios, such as (i) workloads processed on standard computing hardware, and (ii) applications demanding low-latency responses where disk access is prohibitive.

To ensure reproducibility, the challenge's methodology is consistent with previous editions. The process is structured as follows:

- The challenge is divided into a development stage and an evaluation stage.
- Separate datasets and queries are provided for each stage.
- Ground truth data is available for the development stage.
- Participants are given access to development data, while evaluation data is withheld until the final assessment.

The two tasks employ neural embeddings derived from Sentence-BERT models applied to the textual datasets. For Task 1, we provide both in-distribution (data and query vectors come from the same distribution) and out-of-distribution queries (data and query vectors come from different distributions), which enables robust development and comparative analysis. The final evaluation is performed using only the out-of-distribution queries. Out-of-distribution tasks are designed to challenge the limits of current techniques and foster the development of highly efficient and scalable solutions, see for example [12] for a more detailed discussion.

Hardware specifications

The evaluation phase was conducted on a computer workstation equipped with 2x Intel(R) Xeon(R) CPU E5-2690 v4, operating at 2.60GHz with a total of 28 cores, supplemented by 512 GB of DDR4 RAM and a 1 TB SSD. The system runs a Linux kernel version 5.4 on Ubuntu 20.04.6 LTS. Each solution was executed within a Docker container, adhering to predefined resource constraints. The challenge was designed to operate within a strict 16GB physical RAM limit, with swap memory disabled to enforce this constraint. However, a postmortem analysis of the evaluation environment revealed that a minor memory overhead inherent to the containerization system itself could cause solutions operating very close to the 16GB limit to utilize a minimal amount of swap simply to function. This behavior was an artifact of the environment, not an intentional relaxation of the rules for any team.

Roadmap

Section 2 details the preprocessing and search methods we employed as baselines for both tasks. Section 3 provides an overview of Task 1, the solutions suggested

Table 1. List of participating teams and its members.

Team	Members	Task	Ref.
BrownCICESE	Foster, Magdaleno-Gatica, Kimia	1, 2	[9]
cm-lll	Lou, Ma, Luo, Ruan, Wu, Lu, Mao	1	[14]
Crusty Coders	Dearle, Connor, Claydon, McKeogh	1, 2	[4]
DCC-UChile	Bustos, Chen	2	[1]
hforest	Imamura	1, 2	no
JLapeyra	Lapeyra	1, 2	no
TeamDoubleFiltering	Higuchi, Imamura, Shinohara, Hiratta, Kuboyama	1	[11]

by the teams, and an analysis of the results. Section 4 presents the solutions and results for Task 2. Lastly, Section 5 concludes the challenge and explores potential future directions.

Participating Teams

Table 1 provides an overview of the teams that participated in the SISAP 2025 Indexing Challenge. It contains the team names, members, and a reference to each team's explanatory article. Four teams participated in both Task 1 and 2, two teams focused solely on Task 1, and one team addressed just Task 2.

2 Baselines

To enrich the context of the results achieved by competing teams, we add two baselines: *BL-SearchGraph* and *BL-Bruteforce*, based on the Julia's package `SimilaritySearch.jl` on the v0.12 series and PCA projections of different dimensionalities. The *BL-SearchGraph* uses the *SearchGraph* graph-based index using the beam search metaheuristic as its navigation algorithm; see [21,20,22]. It supports online auto-tuning of a couple of hyperparameters (Δ, β) to achieve a minimum recall r as the desired quality, and it also supports tuning to achieve a different recall r^* for search. The self-optimized parameter $\beta \geq 2$ controls the size of the beam size (i.e. backtracking memory) and $0 < \Delta < 2$ that specifies the exploration tendency of the navigation (i.e. what objects could be considered for backtracking); see [22] for more details.

In contrast to other graph indexes like HNSW [15], instead of a hierarchy of neighbors to start a search, it uses a sample of diverse objects to provide fast initialization of navigation and promotes the use of small neighborhoods of variable size selected filtered with a half-space partitioning (HSP or SAT [18]) over a ball of $\log_b(n)$ initial neighbors. *BL-Bruteforce* is a simple but parallel implementation of exhaustive scan.

The baselines use Principal Component Analysis (PCA) to reduce the memory footprint and speed up distance computations. In particular, after PCA projection, we applied scalar quantization to 8-bit integers. Baselines use the

Euclidean distance instead of the angle between vectors due to PCA projection; our implementation casts each byte component to a 32-bit floating point number before arithmetic computations.

3 Task 1: Resource-limited Indexing

Participants in Task 1 were required to develop memory-efficient solutions for an approximate nearest neighbor search. The task demands solutions that operate under strict 16 GB RAM, 8 virtual CPUs, and a wall-clock time of 12 hours for the complete pipeline, which includes loading data, preprocessing, indexing, and searching. The target recall was set to be at least 70%. The team's final score is determined by the highest search throughput exceeding the recall threshold measured across up to 16 different search hyperparameters.

The dataset for this task is derived from the PUBMED corpus of scientific papers [6]. We computed neural embeddings using the Sentence BERT model[7] for this task.[8] The vectors comprise 23.9 million 384-dimensional vectors, and use 35GB of hard disk space using 32-bit floating point numbers. This means that the original dataset cannot be loaded into memory within the resource constraints. The similarity between two objects is measured by their dot product. The dataset is composed of abstracts of scientific articles, and their titles are utilized as queries; this origin distinction of the type of text leads to an implicit distribution change that should be addressed by the indexes. Teams received 11,000 query objects for development, with solutions evaluated on a separate permutation of the dataset and a different 10,000 query set with the same distributions. Table 2 shows the distance distribution quantiles for the in-distribution and out-of-distribution query sets for the development dataset; it shows these statistics for 1NN and 30NN queries.

Table 2. Distance distribution quantiles of the PUBMED dataset.

query		quantiles of the kth nearest neighbor radius										
type	k	min.	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90	max.
in-dist	1	0.000	0.117	0.141	0.159	0.174	0.189	0.205	0.224	0.246	0.276	0.513
out-dist	1	0.000	0.137	0.161	0.180	0.196	0.213	0.229	0.248	0.271	0.307	0.566
in-dist	30	0.000	0.207	0.234	0.253	0.270	0.285	0.301	0.317	0.338	0.368	0.579
out-dist	30	0.131	0.247	0.275	0.296	0.314	0.332	0.350	0.369	0.392	0.424	0.646

[6] PUBMED corpus `https://huggingface.co/datasets/MedRAG/pubmed`.

[7] Sentence BERT model `https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2`

[8] The embeddings can be accessed in `https://huggingface.co/datasets/sadit/SISAP2025`

3.1 Solutions Overview for Task 1

This section outlines the range of solutions received in the SISAP 2025 Indexing Challenge. These solutions were developed in the programming languages C, C++, and Rust, with Python and the Linux's shell serving as high-level interfaces. Our baselines were implemented in the Julia language. As elaborated below, the solutions utilized various indexing structures and dimensionality reduction methodologies. Each submission was meticulously designed to take advantage of the dataset's specific attributes and execution environments.

Task 1 baselines. The *BL-SearchGraph* was built with $r = 0.97$, $r^* = 0.9$ and $b = 1.3$ on the PCA 160d 8-bit quantized dataset. We use $r^* = 0.9$, which is above the expected 0.7 minimum recall for Task 1, but is required to deal with the implicit loss due to the PCA projection. Note that r is relatively high since, as studied by [8], a high-quality graph index can significantly improve search performance at the cost of increasing indexing time. We generate 16 different hyperparameter setups for the search stage by geometrically varying the self-optimized parameter Δ. The starting point for this variation is $\Delta^{\text{opt}}/1.05^2$, with subsequent values increasing by a factor of 1.05. *BL-Bruteforce* uses the same PCA projection, but performing an exhaustive evaluation of the dataset.

3.2 Solutions of Task 1 from Teams

The *TeamDoubleFiltering* solution [11] approach focuses on an online approximate k-NN search using a two-stage filtering strategy. The first stage employs short binary sketches (16 – 20 bits) derived from selected pivots, with an *Annealing by Increasing Resampling* (AIR) process used to identify effective pivot sets. This stage filters candidates based on the asymmetric distance between the query vector and the indexed sketches, which provides higher recall compared to the traditional Hamming distance. The second filtering stage utilizes longer quantized projections, called QSMAP images, derived from dimension-reduced vectors, e.g., 192-bit QSMAPs from 1-bit quantization of 192-dimensional projections. These are stored in sketch-sorted order to improve memory locality and cache performance during filtering. For the final reranking, the system uses 8-bit quantized vectors instead of the original 32-bit floating vectors, further reducing the memory footprint while preserving accuracy. The entire system is designed to be memory-resident.

The *cm-lll* team [14] introduces two key optimizations to the DiskANN [13] pipeline to tackle the task. First, PCA-based dimensionality reduction is applied to compress vectors from 384 to 192 dimensions. This significantly reduces the memory footprint, enabling in-memory indexing for large datasets that would otherwise exceed the memory limit during graph merging. This reduction is shown to retain over 90% of the original information. Second, a refined shard-assignment mechanism is introduced. Instead of the standard DiskANN 2x replication of points across two nearest shards, this optimized strategy assigns each

point to its primary shard and then to the secondary shard with only a probability of 0.5. This reduces the expected replication rate to 1.5x, decreasing indexing overhead, a smaller disk footprint, and faster build times while maintaining recall.

The *Brown-CICESE* [9] solution compresses each 32-bit floating-point value into an 8-bit integer through scalar quantization, achieving a 4x data compression. The team uses a graph-based index for an efficient k-NN search, the compression strategy is designed to preserve neighborhood relationships with high accuracy. The Half-Space Proximal (HSP) Test is employed as an edge selection strategy to create a sparse but geometrically diverse set of neighbors for the search index. The performance of this approach is influenced by the maximum degree of the graph and the number of bits used for data compression.

The *JLapeyra* team proposes a straightforward but effective pipeline built around Faiss [6]. It incorporates optional PCA-based dimensionality reduction to minimize memory usage and accelerate searches, particularly for high-dimensional datasets. The core indexing and search mechanism is based on Faiss IndexFlat IP, which performs an efficient brute-force search using inner product similarity. For Task 1, the dataset is processed in batches to overcome RAM limitations, with previously found candidates retained and merged across batches to ensure result consistency. The system also leverages multithreading to parallelize queries. After an initial candidate retrieval from Faiss, a re-ranking stage refines the results using inner product similarity, with top k results managed via a heap structure. The reported recall for Task 1 is 99.7%.

The methodology adopted by the *Crusty Coders* team [4] is linked to that detailed in 4, which leverages a special 2-bit quantization for vector databases and a related distance function b_2sp. The solution firstly uses their Task 2 solution to create a k-NN graph. This graph is enhanced with reverse neighbors and integrates a navigational algorithm analogous to that presented in [16]. The search algorithm is optimized for speed, employing their b_2sp function; subsequently, a reranking step is executed using the original vectors among the top 100 neighbors retrieved with b_2sp to meet the minimum quality requirements.

We did not receive a formal description from the *hforest* team; however, based on their source code,[9] and previous work of the authors and related work [10,2], the solution is based on binary sketches produced through Hilbert trees, a spatial method that uses the locality preservation properties of a space-filling Hilbert curve. The purpose is to map a sweep the high-dimensional dataset with the Hilbert curve, changing the range queries to interval queries in 1D. The team utilized several trees to form a forest, thereby overcoming the inherent limitations of the approach and enabling them to meet the minimum recall requirements. Note that the Hilbert curve approach produces candidates that must be verified in the original high-dimensional space.

Table 3. Performance listing for Task 1; the table shows the best performant setup per team among the 16 possible search setups that surpasses the recall lower bound. Memory usage statistics of the container's resident memory (rmem) are also shown.

team	rank	recall	build time (s)	query time (s)	throughput (q/s)	container time (s)	rmem (GB) median	rmem (GB) max.
BL-SearchGraph	1	0.7322	4,320	0.60	16,769	4,667	9.6	13.2
BrownCICESE	2	0.7884	9,563	1.44	6,928	9,646	14.2	14.3
TeamDoubleFiltering	3	0.7212	-	9.25	1,081	324	9.6	9.6
hforest	4	0.7053	2,243	15.70	637	2,594	12.2	16.0
cm-lll	5	0.8347	6,419	34.61	289	6,457	11.6	11.6
Crusty Coders	6	0.8048	2,980	178.00	56	3,161	14.5	14.5
JLapeyra	7	1.0000	-	870.32	11	873	7.9	15.3
BL-Bruteforce	8	0.8559	0	1,265.15	8	1,588	5.2	5.3

3.3 Results for Task 1

Table 3 shows the final ranking for Task 1, where solutions are ranked by query throughput for setups achieving the minimum 0.7 average recall. The table includes build time, total query time for 10,000 objects, and the total container time (the end-to-end pipeline). We also report the median and maximum resident memory (rmem) used by each solution's container.

The ranking is led by our BL-SearchGraph baseline. The top-performing participant was BrownCICESE, which secured the second rank with a query time of 1.44 seconds. TeamDoubleFiltering and cm-lll followed in the ranking. All participating teams developed solutions that surpassed the BL-Bruteforce baseline. In terms of overall efficiency, the TeamDoubleFiltering solution stands out; its total pipeline time of 324 seconds is nearly five times faster than the brute-force baseline, and its query phase is over 100 times faster, demonstrating the effectiveness of its two-stage filtering approach.

Figure 1 illustrates the speed-recall trade-offs for teams that submitted multiple hyperparameter configurations. Some solutions, like JLapeyra, performed a single, high-recall run, which is characteristic of their exhaustive search-based approach.

4 Task 2: *k*-NN Graph Construction

Task 2 requires memory-efficient solutions to approximate the k-NN graph, particularly for $k = 15$. The resource limits are identical to Task 1, see §3. Solutions must run with identical computing specifications to Task 1, yet with a minimum average recall of 0.8. Here we used the Google Question & Answers corpus GOOAQ[10], which consists of 3 million 384-dimensional vectors, where the similarity between two objects is measured by their dot product. The database

[9] SISAP 2025 fork `https://github.com/sisap-challenges/sisap25-hforest`.
[10] `https://huggingface.co/datasets/sentence-transformers/gooaq`

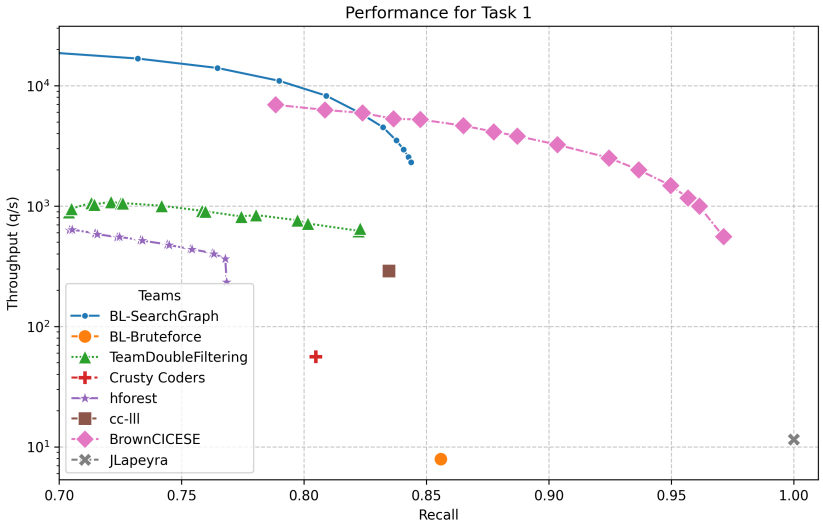

Fig. 1. Comparative performance for Task 1 with up to 16 search hyperparameters by team.

occupies 7.4GB using 32-bit floating point numbers. Along with quality control, solution ranking is determined by total computation time, which includes preprocessing, graph computation, and post-processing. We provided a development dataset that participants could test, while we used a different permutation of the same dataset for evaluation.

Task 2 baselines. We used three baselines:

- A *BL-SearchGraph* that first indexes the dataset and then searches all elements of the dataset using the same index. However, instead of using the typical internal structure to determine a good starting point to navigate, here we used the precomputed nearest neighbor already stored in the graph-index as the starting point, implemented in the function `allknn` of the `SimilaritySearch` package. We created the index with $r = 0.9$, $r^* = 0.9$, and $b = 1.3$;[11] on the dataset projected with PCA 160d using 8-bit quantization.
- Two instances of the bruteforce search algorithm *BL-Bruteforce*, using a PCA 160d and 32d, both with 8-bit quantization.

[11] Note that $r = 0.9$ is faster than the $r = 0.97$ used for Task 1 but it will produce slower searches; also note that the PCA projection and quantization imply a recall loss.

4.1 Solution of Task 2 from Teams

The *DCC-UChile* [1] team's solution is based on an approximate algorithm called *Root Join*, enhanced with several preprocessing steps and modifications. The original Root Join algorithm [7] divides the dataset into disjoint groups and computes k-NN by considering elements within the same group and the next closest group. To improve performance for high-dimensional data, Principal Component Analysis (PCA) is used for dimensionality reduction. A key modification addresses group uniformity, as the original Root Join can create groups of non-uniform sizes. The improved version aims to ensure that all groups contain roughly the same number of elements, reducing the cost of finding the nearest neighbors. The solution also introduces a *substitution to the farthest point* during group creation, where if a new element is closer to a group's true center than its current furthest object, the furthest object is replaced. To enhance effectiveness, a small list (two in their implementation) of the next closest group centers is explicitly stored for each element. The target set for the k-NN computation is expanded to include the element's group plus these two next-closest groups, resulting in searches among a larger target size. Finally, the algorithm leverages parallelization by distributing k-NN searches within groups, which is simplified by their disjoint nature.

The *Crusty Coders* solution [4] introduces a technique called *ultra-quantization* focusing on data compression. This method compresses the data into two bits per original 32-bit floating point number, enabling a very compact data representation (each 384-dimensional embedding is stored in just 1024 bits). A fast bitwise metric is associated with this representation, providing a good approximation to the scalar product of the original data. The NN table is built using a modified version of the NN-Descent algorithm [5]. Key modifications include calculating a single reverse link table, removing all sampling (which was found to reduce output quality in their context), and a different local join algorithm that computes distances among all new links combined with all reverse links, and new links with old links. The implementation is in Rust, using parallelism with the Rayon package and employing lock-free table updates for the nearest-neighbor, similarities, and flags tables to achieve concurrent updates without race conditions.

As mentioned in §3, the solution of the *hforest* team was not documented in an article submitted along with the implementation. Based on a review of the source code and related manuscripts from the authors, the solution is based on the Hilbert Tree. It works by mapping multidimensional data points to a one-dimensional line, then building a B^+ tree on these sorted 1D values to enable efficient k-NN queries through interval searches and candidate verification. For improved accuracy and robustness, instead of using a single tree, a number of them are used, that is, a Hilbert forest comprising multiple trees built from slightly transformed data versions. This structure proved very efficient in Task 2. The method processes points in their 1D Hilbert order, considering only local neighbors, which allows for significant computational reuse, and low-memory rep-

Table 4. Performance listing for Task 2, ranked by total container time. An asterisk (*) denotes solutions that did not meet all constraints.

team	rank	recall	all-knn	container	rmem (GB)	
			time (s)	time (s)	median	max.
hforest	1	0.8049	99	105	7.1	7.4
BL-SearchGraph	2	0.8257	112	165	1.9	2.3
BrownCICESE	3	0.8198	446	450	6.2	6.2
Crusty Coders	4	0.8012	542	548	3.1	3.1
BL-Bruteforce (160d)	-	0.5210*	9,378	9,410	1.2	1.6
JLapeyra	-	0.9944	61,430	61,433*	5.4	14.8
DCC-UChile	-	0.5432*	113,203	113,213*	8.2	16.0

resentation without costly intermediate structures. Inherent parallelism across CPU cores enables high-performance k-NN graph construction.

Brown-CICESE [9] team introduces an iterative approach to build the k-NN graph directly for Task 2, bypassing the need for a separate upfront index. This method initiates with a random graph and iteratively refines it by querying each dataset element over the current graph, similar to [5], yet using a beam search algorithm. The returned candidate neighbors are then used to update the graph links. Key strategies and hyperparameter optimizations for improved convergence include adding reverse links, which proved significantly impactful. A top-layer graph, built on a sample of the dataset, is used to mitigate local minima and provide structure for the beam search. The degree of the graph is also optimized. Additionally, increasing the beam width of the search improves graph quality and allows for more updates to reverse neighbors.

The *JLapeyra* team solution leverages the fact that the GOOAQ dataset fits within memory. Therefore, the entire dataset is processed at once, with the dataset itself serving as both the source and the target of queries. PCA transformation can be applied globally to the dataset. As for Task 1, the solution uses a linear scan implemented in the FAISS library; it includes an optional self-loop exclusion for constructing the k-NN graph.

4.2 Results for Task 2

Table 4 shows the performance for Task 2, ranked by the total container time for valid solutions. The hforest team achieved the first position, completing the task in 105 seconds. The BL-SearchGraph baseline was second, followed by Brown-CICESE and Crusty Coders.

Several solutions were disqualified for not meeting the minimum requirements. For instance, JLapeyra achieved high recall but exceeded the 12-hour time limit, while DCC-UChile did not meet the time or recall constraints. The BL-Bruteforce baseline with 32d PCA projection, while within the time limit, was disqualified for its low recall of 0.5210.

5 Conclusions

This manuscript presents the results of the SISAP 2025 Indexing Challenge, which featured tasks in memory-limited environments. The strict 16 GB RAM constraint for Task 1 (Approximate Nearest-Neighbor Search) and Task 2 (k-NN Graph Construction) required participants to create highly optimized and memory-efficient solutions.

Out-of-distribution queries in Task 1 reflected real-world search scenarios where indexing and search strategies must be resilient to distributional shifts. Teams employed data compression and dimensionality reduction methods like PCA, scalar quantization, and bit-sketches to manage memory limits.

In Task 1, our baseline, BL-SearchGraph, achieved the fastest search time. The BrownCICESE team was the top-ranked participant with a query time of 1.44 seconds. The TeamDoubleFiltering solution had the shortest total container time at 324 seconds, making it well-suited for applications requiring fast end-to-end processing. In Task 2, the hforest team led the ranking with the fastest total container time of 105 seconds, followed by BL-SearchGraph at 165 seconds and BrownCICESE at 450 seconds.

The insights gained from these challenges, including the complex relationship between speed and low-memory data representations, contribute to the advancement of algorithms and techniques vital for a wide range of real-world technologies, from large-scale databases and search engines to machine learning and data analysis platforms.

Acknowledgments. The work of V. Mic on this challenge was supported by a research grant (VIL50110) from VILLUM FONDEN.

Disclosure of Interests. All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

References

1. Bustos, B., Chen, J.: Sisap indexing challenge 2025 – solution for task 2 using root join. In: Similarity Search and Applications: 18th International Conference, SISAP 2025, October 1st-3rd, Proceedings. Springer-Verlag, Berlin, Heidelberg (2025)
2. Chen, H.L., Chang, Y.I.: All-nearest-neighbors finding based on the hilbert curve. Expert Systems with Applications **38**(6), 7462–7475 (2011)
3. Cuconasu, F., Trappolini, G., Siciliano, F., Filice, S., Campagnano, C., Maarek, Y., Tonellotto, N., Silvestri, F.: The power of noise: Redefining retrieval for rag systems. In: Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 719–729 (2024)
4. Dearle, A., Connor, R., Claydon, B., McKeogh, F.: Building a near-neighbour table using equi-voronoi polytopes. In: Similarity Search and Applications: 18th International Conference, SISAP 2025, October 1st-3rd, Proceedings. Springer-Verlag, Berlin, Heidelberg (2025)

5. Dong, W., Moses, C., Li, K.: Efficient k-nearest neighbor graph construction for generic similarity measures. In: Proceedings of the 20th international conference on World wide web. pp. 577–586 (2011)
6. Douze, M., Guzhva, A., Deng, C., Johnson, J., Szilvasy, G., Mazaré, P., Lomeli, M., Hosseini, L., Jégou, H.: The faiss library. CoRR **abs/2401.08281** (2024)
7. Ferrada, S., Bustos, B., Reyes, N.: An efficient algorithm for approximated self-similarity joins in metric spaces. Information Systems **91**, 101510 (2020)
8. 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)
9. Foster, C., Magdaleno-Gatica, S., Kimia, B.: Refinement-based graph construction for search in low-memory systems. In: Similarity Search and Applications: 18th International Conference, SISAP 2025, October 1st-3rd, Proceedings. Springer-Verlag, Berlin, Heidelberg (2025)
10. Higuchi, N., Imamura, Y., Kuboyama, T., Hirata, K., Shinohara, T.: Fast nearest neighbor search with narrow 16-bit sketch. In: ICPRAM. pp. 540–547 (2019)
11. Higuchi, N., Imamura, Y., Shinohara, T., Hirata, K., Kuboyama, T.: Double filtering using short and long quantized projections. In: Similarity Search and Applications: 18th International Conference, SISAP 2025, October 1st-3rd, Proceedings. Springer-Verlag, Berlin, Heidelberg (2025)
12. Jääsaari, E., Hyvönen, V., Ceccarello, M., Roos, T., Aumüller, M.: VIBE: vector index benchmark for embeddings. CoRR **abs/2505.17810** (2025)
13. Krishnaswamy, R., Manohar, M.D., Simhadri, H.V.: The diskann library: Graph-based indices for fast, fresh and filtered vector search. IEEE Data Eng. Bull. **48**(3), 20–42 (2024)
14. Lou, Y., Ma, L., Luo, K., Ruan, Y., Wu, H., Lu, M., Mao, R.: Efficient faiss-based k-nn indexing for sisap 2025 challenge. In: Similarity Search and Applications: 18th International Conference, SISAP 2025, October 1st-3rd, Proceedings. Springer-Verlag, Berlin, Heidelberg (2025)
15. 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)
16. Malkov, Y., Ponomarenko, A., Logvinov, A., Krylov, V.: Scalable distributed algorithm for approximate nearest neighbor search problem in high dimensional general metric spaces. In: Navarro, G., Pestov, V. (eds.) Similarity Search and Applications. pp. 132–147. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
17. McInnes, L., Healy, J., Melville, J.: Umap: Uniform manifold approximation and projection for dimension reduction. arXiv preprint arXiv:1802.03426 (2018)
18. Navarro, G.: Searching in metric spaces by spatial approximation. The VLDB Journal **11**(1), 28–46 (2002)
19. Reimers, N., Gurevych, I.: Sentence-bert: Sentence embeddings using siamese bert-networks. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics (11 2019)
20. Tellez, E.S., Aumüller, M., Chavez, E.: Overview of the sisap 2023 indexing challenge. In: Similarity Search and Applications: 16th International Conference, SISAP 2023, A Coruña, Spain, October 9–11, 2023, Proceedings. p. 255–264. Springer-Verlag, Berlin, Heidelberg (2023)
21. Tellez, E.S., Aumüller, M., Mic, V.: Overview of the sisap 2024 indexing challenge. In: Similarity Search and Applications: 17th International Conference, SISAP 2024, Providence, RI, USA, November 4–6, 2024, Proceedings. p. 255–265. Springer-Verlag, Berlin, Heidelberg (2024)

22. Tellez, E.S., Ruiz, G.: Similaritysearch.jl: Autotuned nearest neighbor indexes for julia. Journal of Open Source Software **7**(75), 4442 (2022)