arXiv:submit/7734516 [cs.DB] 20 Jun 2026

DaiSy: A Library for Scalable Data Series Similarity Search

Francesca Del Gaudio*
Université Paris Cité, LIPADE
F-75006, Paris, France
francescadelgaudio56@gmail.com

Manos Chatzakis*
Université Paris Cité, LIPADE
F-75006, Paris, France
manos.chatzaki@gmail.com

Gayathiri Ravendirane
Université de Bordeaux
Bordeaux, France
gayathiri.ravendirane@etu.u-bordeaux.fr

Botao Peng
Chinese Academy of Sciences
Beijing, China
pengbotao@ict.ac.cn

Themis Palpanas
Université Paris Cité, LIPADE
F-75006, Paris, France
themis@mi.parisdescartes.fr

ABSTRACT

Exact similarity search over large collections of data series is a fundamental operation in modern applications, yet existing solutions are often fragmented, specialized, or tailored to specific execution environments. In this paper, we present DaiSy, a unified library for exact data series similarity search that integrates multiple state-of-the-art (iSAX-based) algorithms within a single, coherent framework. DaiSy is the first library to support exact similarity search across diverse execution environments, including implementations for disk-based, in-memory, GPU-accelerated, and distributed scalable similarity search. Although designed for data series, DaiSy is also directly applicable to exact similarity search over vector data, enabling its use in a broader range of applications. The library supports interfaces in both C++ and Python, enabling users to easily integrate its functionality into a variety of tasks. DaiSy is open-sourced and available at: `https://github.com/MChatzakis/DaiSy`.

1. INTRODUCTION

Data series constitute one of the most popular data types, appearing across a wide range of application domains, including finance, astrophysics, neuroscience, engineering, seismology, and many others [53, 35]. Such domains typically generate large collections of data series that must be analyzed in order to extract meaningful knowledge [25, 52, 3, 46, 24, 41]. Within these data analytics tasks, similarity search serves as a fundamental operation that enables efficient and effective analysis. Moreover, many similarity search applications are highly sensitive to result accuracy, highlighting the need for similarity search methods that avoid approximation errors, i.e., exact similarity search [16]. In exact query answering, it is crucial to guarantee that the returned data series is indeed the most similar to the query among all data series in the collection.

Scalable Similarity Search. The increasing interest in scalable similarity search across different application domains has led to massive research on search time optimization [14, 12]. To enable scalable similarity search, numerous indexing methods have been proposed [36]. These methods construct index structures over data series collections and employ specialized search algorithms to answer queries. A large portion of prior work focuses on optimizing the core similarity search algorithms themselves, aiming to reduce query latency and improve throughput [13, 23, 28, 32, 18]. At the same time, several approaches extend similarity search to different execution environments based on the available hardware, such as disk-based systems [37], in-memory CPUs [38], GPU-accelerated systems [39], and distributed infrastructures [6, 51]. However, despite this rich body of work, existing methods are largely dispersed, with implementations scattered across different codebases and system designs, which makes practical adoption challenging.

DaiSy. We present DaiSy, the first scalable solution for exact data series similarity search that integrates multiple algorithms into a single, unified library. DaiSy enables efficient similarity search across a wide range of environments and system configurations by supporting scalable algorithms for disk-based, in-memory, GPU-accelerated, and distributed execution. This is achieved by incorporating representative state-of-the-art (iSAX-based) algorithms for each execution environment: ParIS+ [37] for disk-based processing, MESSI [38] for in-memory processing, SING [39] for GPU-accelerated process-

*These authors contributed equally to this work.

ing, and Odyssey [6] for distributed processing. ParIS+ efficiently supports disk-based similarity search when data do not fit in main memory, MESSI provides highly efficient parallel in-memory search, SING exploits GPU processing to accelerate key search operations targetted to CUDA-enabled GPUs , and Odyssey enables exact similarity search over massive datasets using distributed memory and optimized communication. Currently, GPU acceleration is available only on CUDA-enabled GPUs.

All four above algorithms, Paris+, MESSI, SING, and Odyssey, are guaranteed to always return the exact, correct answers. The selection of these systems was motivated by the fact that they represent the state-of-the-art approaches for exact similarity search [17, 48].

Finally, we note that although these algorithms were designed with data series in mind, they are equally applicable to general high-dimensional vectors (e.g., deep embeddings), where they exhibit state-of-the-art performance, as well [16, 17, 48]. This capability is particularly important today, since it constitutes a core component of modern Retrieval-Augmented Generation (RAG) systems for LLMs [15, 30, 4, 42, 31]. In addition, approximate vector search applications behind RAG systems usually require some form of training [7, 8] and/or tuning [33], which requires exact search to obtain groundtruth results. In such scenarios, DaiSy can significantly accelerate the groundtruth generation process, and represents the current state-of-the-art solution for this task.

DaiSy exposes a unified interface for all supported algorithms and is available in both C++ and Python to facilitate adoption by users. DaiSy is open sourced and available at: `https://github.com/MChatzakis/DaiSy`.

Contributions. We summarize our main contributions as follows:

- We present DaiSy, the first scalable library for exact data series, as well as vector, similarity search that integrates multiple state-of-the-art algorithms within a unified framework.
- We describe how DaiSy enables efficient data series similarity search across a wide range of configurations and systems, by supporting disk-based, in-memory, GPU-accelerated, and distributed execution environments.
- We design an extensible similarity search library architecture for DaiSy, accompanied by comprehensive benchmarking frameworks, and we provide both C++ and Python interfaces, facilitating easy adoption across diverse domains and applications.
- We release DaiSy as an open-source library and make it publicly available to the community through our GitHub repository.

2. BACKGROUND

Data Series. A *data series* $S = \{p_1, \ldots, p_n\}$ is a sequence of n points, where each $p_i = (u_i, t_i)$ represents a value u_i at position t_i.

iSAX Summary. The *iSAX summary* [44] discretizes a data series by dividing the x-axis into equal segments (represented by their mean) and the y-axis into regions based on normal distribution breakpoints. Each region is assigned a symbolic representation with a specific *cardinality*, enabling hierarchical *iSAX-based index trees* [36].

Similarity Search. Given a collection $\mathcal{C}$ and a query S, *similarity search* identifies the top-k series in $\mathcal{C}$ most similar to S according to a distance measure.

Distance Measures. The *L2 (Euclidean) Distance* is $L2S(T, S) = \sqrt{\sum_{i=1}^{n}(t_i - s_i)^2}$. The distance between iSAX summaries provides a *lower-bound* to the real distance. *Dynamic Time Warping (DTW)* allows non-linear alignments by computing the minimum cumulative cost over all valid warping paths [26].

2.1. Related Work

Data Series Indexes. Various indices, specific to data series, have been proposed in the literature [14, 12]. DSTree [47] is an index based on the APCA summarization [27]. The DSTree can adaptively perform split operations by increasing the detail of APCA as needed. The iSAX index is based on the SAX summarization, and its extension, iSAX [44]. Other iSAX-based indices have been proposed in the literature [5, 39, 38, 37, 51, 6, 28, 13, 23, 11, 32, 19, 49], summarized in [36] and evaluated in [16].

Similarity Search Libraries. FAISS [10] is a library that implements a wide range of algorithms for approximate vector similarity search. It is conceptually the most closely related system to DaiSy; however, it targets application scenarios in which the exactness constraint can be relaxed, e.g. Retrieval Augemented Generation (RAG) [31]. FAISS provides a limited functionality for exact similarity search through a naive bruteforce search implementation. In addition, **UCR Suite** [40] and **TSSEARCH** [21] are libraries for subsequence similarity search in C++ and Python respectively. They both tackle a similar variant of similarity search, but they do not provide various different methods to perform indexing or search. **AEON** [34] and **TSLEARN** [45] are machine learning oriented Python libraries for data series processing, with strong focus on analytics tasks. Although they provide some functionalities for computing distances between data series, they are not libraries for scalable data series similarity search and

they do not expose any interface for indexing or search.

3. SYSTEM DESIGN

DaiSy's architecture follows a layered design centered on a C++ core that implements indexing, distance computation, and search functionality, while exposing the same conceptual model through a Python interface. Foundational primitives provide distance evaluation, data access, and reusable lower bounds, while an index layer encapsulates the iSAX index as an internal service [5]. On top of these, multiple execution models realize exact similarity search under different assumptions on data placement and available resources. The same abstractions are exposed in both C++ and Python, ensuring a uniform user-facing interface independent of the execution back ends enabled at build time.

3.1. Architecture

Design principles and abstractions. DaiSy is structured around a set of design principles that emphasize modularity, enforcing a strict separation of concerns along three orthogonal dimensions. Distance computation is encapsulated in a dedicated component responsible for exact distance evaluation and lower bounds, independent of data storage and execution strategy. Data access is handled through an explicit adapter-based abstraction [1], which decouples indexing and search logic from the physical layout and location of the data and enables support for heterogeneous data sources, including in-memory and on-disk representations. Execution strategy is treated as a separate concern, while search algorithms are implemented as interchangeable components exposing a common interface.

Core primitives layer. The core primitives layer provides the shared foundation on which all indexing structures and search algorithms in DaiSy are built. It deliberately implements no index or search strategy; instead, it factors out functionality common across algorithms and execution models to avoid duplication. Distance-related logic is encapsulated in a unified abstraction supporting exact evaluation and lower bounds, and is independent of data placement and execution strategy. Data access is exposed through a stable adapter-based interface that models datasets as streams of data series, keeping higher-level components agnostic to in-memory or on-disk storage. The layer also provides shared reduced representations and lower bounds enabling safe pruning while preserving correctness, forming a stable, metric- and storage-agnostic substrate for higher layers.

Index layer. The index layer encapsulates the iSAX index as an internal service that provides structured access to data series data, without prescribing a search strategy. Index construction is encapsulated by a single configuration object that defines representation parameters, buffer sizes, and storage mode. Importantly, the index layer does not implement search logic. By treating iSAX as a service rather than as an algorithm, DaiSy decouples index management from execution strategy and allows the same index implementation to be reused across multiple execution models.

Similarity Search layer. The similarity search layer in DaiSy is organized by execution model rather than by individual algorithm implementations. Each execution model defines how exact similarity search is carried out under specific assumptions about data placement and available resources, while reusing the same core primitives and index services. Taken together, these models position DaiSy as an execution framework that instantiates a single conceptual search pipeline through multiple back ends. DaiSy supports a variety of similarity search algorithms in this layer, covering use cases in disk-based, in-memory, GPU-accelerated and distributed environments. ParIS+ [37] is an algorithm designed for disk-based exact similarity search, and its tailored for situations where the dataset size exceeds the main memory capacity of the system. MESSI [38] is an exact similarity search algorithm that operates entirely in memory and relies on an iSAX index to guide candidate selection and refinement. It follows a multi-phase search pipeline that combines index-guided exploration with exact refinement while preserving exactness. The SING index [39] leverages GPUs to accelerate the search procedure. It is organized into two main stages: in-memory index construction and multi-phase query answering with GPU offloading. Odyssey [6] is a distributed framework for exact similarity search built on MPI, designed for multi-node environments where datasets can be accommodated in distributed memory. DaiSy also supports two exhaustive search implementations.
The first one, *Bruteforce*, simply performs all real distance computations over the raw data. The second one, *LbBruteforce*, applies lower-bounding, thus, avoiding real distance computations when the lower bound is larger than the BSF distance.

3.2. Distance Measures

DaiSy supports L2 Squared and DTW for exact similarity search, both exposed through a unified distance computer abstraction. Our library supports both optimized SIMD and scalar implementations, to ensure compatibility with different hardware con-

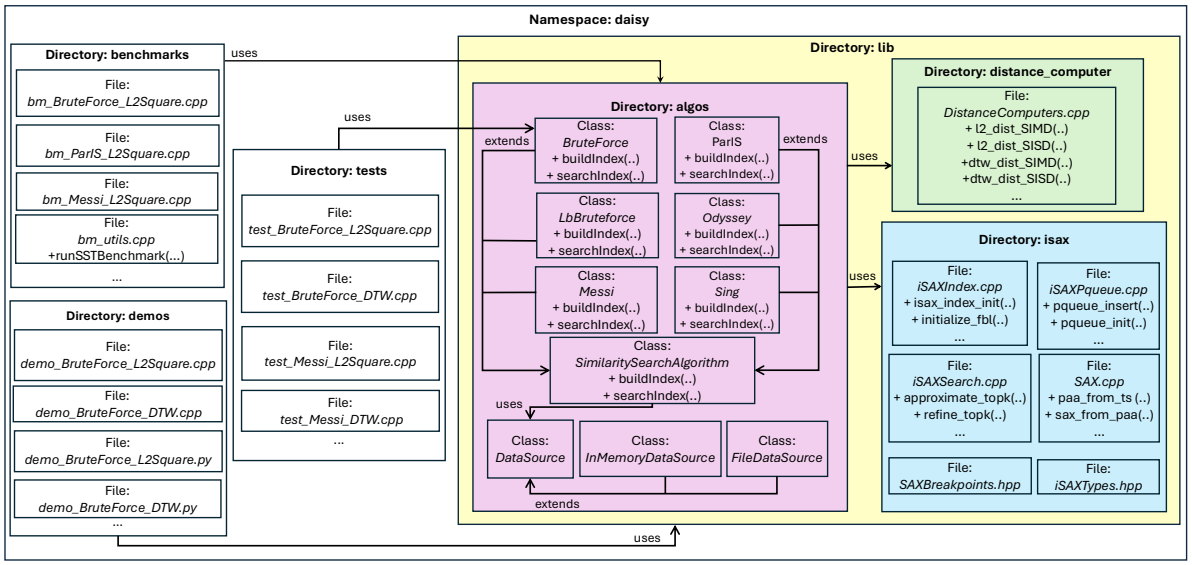


Figure 1: Component Diagram of DaiSy.

figurations. Distance computations support early termination via an optional upper bound, allowing evaluation to stop once the partial distance exceeds the current best-so-far value [40]. In addition, DaiSy provides exact lower-bound functions for both measures.

3.3. Z-normalization

DaiSy supports both z-normalized (i.e., each data series, or vector, has mean 0 and standard deviation 1) and non z-normalized data. Though, the algorithms are more suited to Z-normalized data (we use the default iSAX breakpoints [44], which have been optimized for z-normalized data).

4. USING DAISY

4.1. API and Tooling

DaiSy exposes a simple API to enable easy use in both C++ and Python.

- `buildIndex`. Initializes and builds the index structure for the selected algorithm.
- `searchIndex`. Performs the top-k index search for the given queries, returning the most similar data series from the index.

Furthermore, reproducibility and systematic evaluation are supported through a set of shared utilities, demos, benchmarks, and tests. Demos are provided in both C++ and Python and exercise the same pipelines exposed to end users. Benchmarks enable controlled performance evaluation across execution models and configurations, while tests validate correctness and integration. All tooling components consume the same APIs as the core library, ensuring consistency between usage, evaluation, and

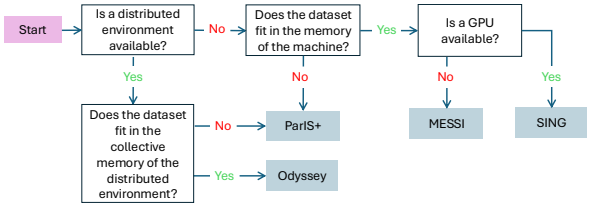


Figure 2: Algorithm selection decision tree of DaiSy.

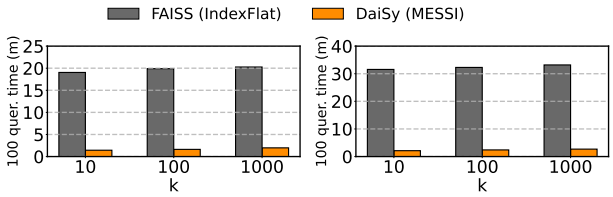
experimentation.

4.2. Hyperparameter Setting

As expected, the methods implemented in DaiSy involve a number of algorithm-specific hyperparameters. The library allows experienced users to explicitly configure all relevant hyperparameters for each supported algorithm. At the same time, DaiSy does not require the user to manually specify these parameters, as it initializes them to the values recommended in the original publications of the corresponding algorithms.

4.3. Algorithm Selection

As we demonstrated, DaiSy provides efficient similarity search algorithms tailored for different execution environments. Figure 2 summarizes the algorithm selection logic based on dataset size and available resources. When the dataset fits in main memory, DaiSy first considers whether a distributed environment is available and, if so, selects Odyssey to exploit distributed memory. In the absence of a distributed setting, the choice is driven by the available hardware accelerators: SING is used when a GPU is present, while MESSI is selected for CPU-based in-memory execution. When the dataset does



(a) Deep100M (b) Seismic100M

Figure 3: Query answering time for 100 queries when varying k (48 hyperthreads).

not fit in main memory, DaiSy resorts to ParIS+ for disk-based execution.

We would like to stress that, apart from data series, DaiSy is equally applicable to vector data, as well, where it exhibits state-of-the-art performance. For example, DaiSy is the solution of choice for applications that need exact answers in vector search, or for analysis tasks that involve vector approximate search and need to compute the groundtruth answers. As evidence, we present the results of the following experiment (rigorous experimental evaluations can be found in previous studies [16, 17, 6]). We conduct an in-memory evaluation against FAISS-IndexFlat, the most widely used library for such tasks, focusing on exact similarity search for two datasets: Deep100M [2] (100 million 96-dimensional image embeddings) and Seismic100M [22] (100 million 256-dimensional seismic data series). IndexFlat is the FAISS solution used to compute exact answers: it stores the full vectors and performs an exhaustive search. Both libraries are compiled with identical settings using `g++ 11.4.0`, configured for in-memory execution, and execution times are measured using the GoogleBenchmark [1] framework. Figure 3 reports the total query execution time for 100 queries on each dataset, when varying k between 10-1000, using 48 hyperthreads.

The results show that DaiSy-MESSI consistently outperforms FAISS-IndexFlat, calculating the exact answers for the entire workload 12× faster for the Deep100M vector collection. The same observation is true for the Seismic100M data series collection, where DaiSy-MESSI is 14× faster. Speed-up increases with the number of threads. Similar results hold for other vector datasets, as well.

4.4. Examples

We provide two examples of using DaiSy in C++ and Python to demonstrate the convenience our library provides for similarity search. Without loss of generality, in our examples we use the MESSI algorithm and the L2 Squared distance.

```cpp
#include <vector>
#include "daisy/daisy.hpp"
int main() {
    daisy::idx_t d = 256; // dimensionality
    daisy::idx_t n_db = 10000; // dataset size
    daisy::idx_t n_q = 100; // queries size

    // Load or generate input data series.
    float* db = /* load or generate database series */;
    float* q = /* load or generate query series */;

    // Build index using squared Euclidean distance.
    daisy::Messi messi(daisy::DistanceType::L2_SQUARED);
    messi.buildIndex(db, n_db, d);

    // Execute exact k-NN similarity search.
    const daisy::idx_t k = 5;
    std::vector<daisy::idx_t> I(static_cast<size_t>(n_q)*k);
    std::vector<float> D(static_cast<size_t>(n_q)*k);
    messi.searchIndex(q, n_q, k, I.data(), D.data());

    return 0; }
```

```python
from daisy import DistanceType, Messi

d = 256 # dimensionality
n_db = 100000 # dataset size
n_q = 1000 # queries size

# Load or generate input data.
db = ... # load or generate database series
q = ... # load or generate query series

# Build the index using squared Euclidean distance.
index = Messi(DistanceType.L2_SQUARED)
index.buildIndex(db)

# Execute exact k-NN similarity search.
k = 5
I, D = index.searchIndex(q, k)
```

5. FUTURE WORK

DaiSy serves as the foundation for several planned future efforts aimed at further improving and extending the library. In particular, we plan to incorporate additional algorithms for exact similarity search, such as DumpyOS [49], including methods that do not rely on the iSAX index, such as Hercules [11] and SOFA [43], which is enabled by the modular architecture of DaiSy. We also plan to extend the library to support range queries [16], as well as subsequence similarity search [40, 20, 32, 50], streaming data series [29], and early termination [23, 13, 7]. We also intend to introduce automatic hyperparameter tuning capabilities for the supported algorithms, such as Bayesian optimization [9]. Our immediate priorities will focus on integrating non-iSAX-based systems and on supporting range-based queries.

6. CONCLUSIONS

We present DaiSy, a novel library for scalable exact similarity search on large collections of data series, as well as general high-dimensional vectors (e.g., deep embeddings). It supports efficient search algorithms

[1] `https://github.com/google/benchmark`

across a wide range of hardware configurations. We open source DaiSy and provide carefully designed C++ and Python interfaces.

ACKNOWLEDGMENTS

Work supported by EU Horizon projects ARMADA (101168951), TwinODIS (101160009), DataGEMS (101188416), and $Y\Pi AI\Theta A$ & NextGenerationEU project HARSH ($Y\Pi 3TA-0560901$) that is carried out within the framework of the National Recovery and Resilience Plan "Greece 2.0" with funding from the European Union – NextGenerationEU. Manos Chatzakis is supported with a PhD Scholarship from the Onassis Foundation.

7. REFERENCES

[1] M. G. Al-Obeidallah, D. G. Al-Fraihat, A. M. Khasawneh, A. M. Saleh, and H. Addous. Empirical investigation of the impact of the adapter design pattern on software maintainability. In *ICIT*. IEEE, 2021.
[2] A. Babenko and V. Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. In *IEEE CVPR*, 2016.
[3] P. Boniol and T. Palpanas. Series2Graph: Graph-based Subsequence Anomaly Detection for Time Series. *PVLDB*, 2020.
[4] S. Bruch, F. M. Nardini, C. Rulli, and R. Venturini. Efficient inverted index-based approximate retrieval over high-dimensional learned sparse representations. *IEEE Data Eng. Bull.*, 48(3):43–62, 2024.
[5] A. Camerra, T. Palpanas, J. Shieh, and E. Keogh. isax 2.0: Indexing and mining one billion time series. In *IEEE ICDM*, 2010.
[6] M. Chatzakis, P. Fatourou, E. Kosmas, T. Palpanas, and B. Peng. Odyssey: A journey in the land of distributed data series similarity search. 2023.
[7] M. Chatzakis, Y. Papakonstantinou, and T. Palpanas. Darth: Declarative recall through early termination for approximate nearest neighbor search. *SIGMOD*, 2025.
[8] M. Chatzakis, Y. Papakonstantinou, and T. Palpanas. DARTH+: Approximate Nearest Neighbor Search with Declarative Recall and Quality Guarantees. 2026.
[9] S. Daulton, M. Balandat, and E. Bakshy. Differentiable expected hypervolume improvement for parallel multi-objective bayesian optimization. *NeurIPS*, 2020.
[10] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou. The faiss library. *IEEE Transactions on Big Data*, 2025.
[11] K. Echihabi, P. Fatourou, K. Zoumpatianos, T. Palpanas, and H. Benbrahim. Hercules against data series similarity search. *arXiv preprint arXiv:2212.13297*, 2022.
[12] K. Echihabi and T. Palpanas. Scalable analytics on large sequence collections. In *MDM*. IEEE, 2022.
[13] K. Echihabi, T. Tsandilas, A. Gogolou, A. Bezerianos, and T. Palpanas. Pros: data series progressive k-nn similarity search and classification with probabilistic quality guarantees. *VLDB J.*
[14] K. Echihabi, K. Zoumpatianos, and T. Palpanas. New trends in high-d vector similarity search: ai-driven, progressive, and distributed. *PVLDB*.
[15] K. Echihabi, K. Zoumpatianos, and T. Palpanas. Scalable machine learning on high-dimensional vectors: From data series to deep network embeddings. In *WIMS*, 2020.
[16] K. Echihabi, K. Zoumpatianos, T. Palpanas, and H. Benbrahim. The Lernaean Hydra of Data Series Similarity Search: An Experimental Evaluation of the State of the Art. *PVLDB*, 2018.
[17] K. Echihabi, K. Zoumpatianos, T. Palpanas, and H. Benbrahim. Return of the lernaean hydra: Experimental evaluation of data series approximate similarity search. *PVLDB*, 2019.
[18] P. Fatourou, E. Kosmas, T. Palpanas, and G. Paterakis. Fresh: a lock-free data series index. In *SRDS*. IEEE, 2023.
[19] P. Fatourou, E. Kosmas, T. Palpanas, and G. Paterakis. Fresh: A lock-free data series index. In *SRDS*. IEEE, 2023.
[20] K. Feng, P. Wang, J. Wu, and W. Wang. L-match: A lightweight and effective subsequence matching approach. *IEEE Access*, 8:71572–71583, 2020.
[21] D. Folgado, M. Barandas, M. Antunes, M. L. Nunes, H. Liu, Y. Hartmann, T. Schultz, and H. Gamboa. Tssearch: Time series subsequence search library. *SoftwareX*, 2022.
[22] I. R. I. for Seismology with Artificial Intelligence. Seismic Data Access. `http://ds.iris.edu/data/access/`, 2018.
[23] A. Gogolou, T. Tsandilas, T. Palpanas, and A. Bezerianos. Progressive similarity search on time series data. In *BigVis*, 2019.
[24] P. Huijse, P. A. Estevez, P. Protopapas, J. C. Principe, and P. Zegers. Computational intelligence challenges and applications on large-scale astronomical time series databases. *CIM*, 2014.
[25] K. Kashino, G. Smith, and H. Murase. Time-series active search for quick retrieval of audio and video. In *ICASSP*, 1999.
[26] E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and information systems*, 7(3):358–386, 2005.
[27] E. J. Keogh, K. Chakrabarti, S. Mehrotra, and M. J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In S. Mehrotra and T. K. Sellis, editors, *SIGMOD*, 2001.
[28] H. Kondylakis, N. Dayan, K. Zoumpatianos, and T. Palpanas. Coconut palm: Static and streaming data series exploration now in your palm. In *Proceedings of the 2019 International Conference on Management of Data*, 2019.
[29] H. Kondylakis, N. Dayan, K. Zoumpatianos, and T. Palpanas. Coconut: sortable summarizations for scalable indexes over static and streaming data series. *VLDB J.*, 28(6):847–869, 2019.
[30] R. Krishnaswamy, M. D. Manohar, and H. V. Simhadri. The diskann library: Graph-based indices for fast, fresh and filtered vector search. *IEEE Data Eng. Bull.*, 2024.
[31] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *NeurIPS*, 2020.
[32] M. Linardi and T. Palpanas. Scalable, variable-length similarity search in data series: The ulisse approach. *PVLDB*, 2018.
[33] D. Lu, H. Caminal, M. Chatzakis, Y. Papakonstantinou, Y. Chronis, V. Jain, and F. Özcan. An in-depth study of filter-agnostic vector search on a postgresql database system:[experiments & analysis]. *SIGMOD*, 2026.
[34] M. Middlehurst, A. Ismail-Fawaz, A. Guillaume, C. Holder, D. Guijo-Rubio, G. Bulatova, L. Tsaprounis, L. Mentel, M. Walter, P. Schäfer, and A. Bagnall. aeon: a python toolkit for learning from time series. *JMLR*, 2024.
[35] T. Palpanas. The parallel and distributed future of data series mining. In *HPCS*, 2017.
[36] T. Palpanas. Evolution of a data series index: The isax family of data series indexes: isax, isax2. 0, isax2+, ads, ads+, ads-full, paris, paris+, messi, dpisax, ulisse, coconut-trie/tree, coconut-lsm. In *ISIP*, 2020.
[37] B. Peng, P. Fatourou, and T. Palpanas. Paris: The next destination for fast data series indexing and query answering. In *Big Data*. IEEE, 2018.
[38] B. Peng, P. Fatourou, and T. Palpanas. Messi: In-memory data series indexing. In *IEEE ICDE*, 2020.
[39] B. Peng, P. Fatourou, and T. Palpanas. Sing: Sequence indexing using gpus. In *ICDE*. IEEE, 2021.
[40] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *SIGKDD*, 2012.
[41] U. Raza, A. Camerra, A. L. Murphy, T. Palpanas, and G. P. Picco. Practical data prediction for real-world wireless sensor networks. *TKDE*, 2015.
[42] V. Sanca, M. Chatzakis, and A. Ailamaki. Optimizing context-enhanced relational joins. *2024 IEEE ICDE*, 2024.
[43] P. Schäfer, J. Brand, U. Leser, B. Peng, and T. Palpanas. Fast and exact similarity search in less than a blink of an eye. In *IEEE ICDE*, 2025.
[44] J. Shieh and E. J. Keogh. *i*sax: indexing and mining terabyte sized time series. In *SIGKDD*. ACM, 2008.
[45] R. Tavenard, J. Faouzi, G. Vandewiele, F. Divo, G. Androz, C. Holtz, M. Payne, R. Yurchak, M. Rußwurm, K. Kolar, et al. Tslearn, a machine learning toolkit for time series data. *JMLR*, 2020.
[46] Q. Wang, S. Whitmarsh, V. Navarro, and T. Palpanas. iEDeaL: A Deep Learning Framework for Detecting Highly Imbalanced Interictal Epileptiform Discharges. *PVLDB*, 16(2), 2023.
[47] Y. Wang, P. Wang, J. Pei, W. Wang, and S. Huang. A data-adaptive and dynamic segmentation index for whole matching on time series. *PVLDB*, 6(10), 2013.
[48] Z. Wang, P. Wang, T. Palpanas, and W. Wang. Graph- and tree-based indexes for high-dimensional vector similarity search: Analyses, comparisons, and future directions. *IEEE Data Eng. Bull.*, 47(3):3–21, 2023.
[49] Z. Wang, Q. Wang, P. Wang, T. Palpanas, and W. Wang. Dumpyos: A data-adaptive multi-ary index for scalable data series similarity search. *VLDB J.*, 2024.
[50] H. Xiong, H. Zhang, Z. Wang, Z. He, P. Wang, and X. S. Wang. CIVET: exploring compact index for variable-length subsequence matching on time series. *Proc. VLDB Endow.*, 17(9):2123–2135, 2024.
[51] D. E. Yagoubi, R. Akbarinia, F. Masseglia, and T. Palpanas. Dpisax: Massively distributed partitioned isax. In *IEEE ICDM*, 2017.
[52] L. Ye and E. Keogh. Time series shapelets: a new primitive for data mining. In *SIGKDD*. ACM, 2009.
[53] K. Zoumpatianos and T. Palpanas. Data series management: Fulfilling the need for big sequence analytics. In *IEEE ICDE*, 2018.