Eidolon: An Architecture for the Reproducible Publication of Scientific Data Collections

Lucas Calmon[1], Marcus Junior[1], Lucas Santos[1], Fernando Alexandrino[1,2], Rafaelli Coutinho [1], Gustavo Guedes [1], Diego Carvalho[1], Eduardo Ogasawara[1]

[1]Centro Federal de Educação Tecnológica Celso Suckow da Fonseca – CEFET/RJ

[2]Instituto Federal de Educação, Ciência e Tecnologia de São Paulo – IFSP

{lucas.calmon, marcus.junior, lucas.santos.16}@aluno.cefet-rj.br,

fernando.alexandrino@ifsp.edu.br, rafaelli.coutinho@cefet-rj.br,

gustavo.guedes@cefet-rj.br, dcarvalho@ieee.org, eogasawara@ieee.org

Abstract. *Scientific data collections used in data analytics workflows must reconcile four aspects: (i) data curation, (ii) reusable documentation, (iii) direct integration with a computational environment, and (iv) lightweight distribution. Although repositories and portals support storage and discovery, they do not always provide a runtime usage contract. This work proposes* Eidolon*, a publication architecture based on curation, structural standardization, compact local miniatures, and on-demand remote expansion. The architecture is instantiated in the integration between* tspredbench *and* tspredit*. This case study indicates that* Eidolon *preserves local discovery, documentation, and reproducibility without embedding the full dataset in the package.*

1. Introduction

Advances in e-Science make the production and consumption of data collections routine, and these collections must remain reusable across experiments, pipelines, and computational environments. In benchmark settings, it is not enough for data to be available in a repository: they must also be discoverable, documented, and consumable in a way that is integrated with the environment in which experiments are executed [Wilkinson et al., 2016; Peng, 2011; Sandve et al., 2013].

A recurring tension remains between two extreme strategies. In the first, data are kept only in external repositories, portals, or dataset hubs. This approach favors broad storage and sharing, as seen in initiatives such as OpenML and Hugging Face Datasets [Vanschoren et al., 2014; Lhoest et al., 2021], and is often unavoidable when collections reach multi-gigabyte scale. However, external availability alone does not ensure reproducible consumption. In this model, the links among metadata, schema, access interfaces, package structure, and actual runtime behavior often remain loosely coupled and distributed across distinct layers, especially when multiple datasets from different sources are involved. As a result, datasets may be discoverable and well documented, while their effective use in experiments still depends on ad hoc integration steps that the software artifact itself does not fully capture. This fragmentation becomes more pronounced as code, metadata, and data collections evolve over time.

In the second, data are embedded directly in packages. This approach simplifies initial use and local experimentation, but it faces practical distribution constraints. The issue becomes particularly relevant when the collections consumed by scientific software reach multi-gigabyte scale, a scenario explicitly acknowledged in package- and library-oriented data ecosystems [Kane et al., 2025; Hugging Face, 2026]. Such collections may range from multi-gigabyte datasets to very large corpora such as The Pile, presented in Hugging Face Datasets as an 825 GB resource [Hugging Face, 2026]. In the R ecosystem, for example, CRAN policies explicitly require minimizing package size and discourage use of the repository as the primary means of distributing large volumes of data [CRAN Repository Maintainers, 2025]. In the Python ecosystem, PyPI also imposes storage limits per file and per project [PyPI Docs, 2026].

This work starts from the following problem: *how can large scientific data collections be published in a way that is simultaneously curated, consistent, reproducible, and usable in the benchmark execution environment, without turning the package that exposes these data into an excessively heavy artifact?* The focus is therefore on publishing collections for *package-native* consumption, that is, directly in the *runtime* where the benchmark is configured, executed, and reproduced.

This paper introduces *Eidolon*, a scientific software architecture that brings packages and data into closer interaction. Rather than treating data as something described only through catalogs or external links, *Eidolon* allows packages to carry compact dataset miniatures that preserve the same access logic as the complete artifact.

In this context, the paper makes three main contributions. It introduces the *Eidolon* architecture for publishing scientific collections, positions it as the basis of a protocol centered on curation, standardization, lightweight packaging, and on-demand expansion, and evaluates its feasibility in a real-world time-series prediction scenario. This scenario is represented by the integration between the *tspredit* forecasting framework [Salles et al., 2023] and *tspredbench*, a curated collection of datasets for predictive evaluation [Ogasawara et al., 2025]. From this instantiation, we discuss benefits, limitations, and implications for other scientific software ecosystems.

Beyond this introduction, the paper is organized into five sections. Section 2 presents the literature review. Section 3 describes the Eidolon Architecture. Section 4 discusses its instantiation in *tspredbench*. Section 5 presents implications and limitations. Finally, Section 6 concludes the paper.

2. Literature Review

The literature relevant to this work spans scientific reproducibility, data curation, benchmark platforms, and artifact distribution in Data Analytics ecosystems.

In the field of reproducibility, Peng [2011] and Sandve et al. [2013] argue that the value of a computational artifact depends on its ability to be re-executed, inspected, and reused by third parties. The FAIR principles reinforce this view by arguing that scientific data should be findable, accessible, interoperable, and reusable [Wilkinson et al., 2016]. In the context of this paper, this means that a data collection should not merely be hosted, but organized in a format that allows experimental workflows to operationalize benchmarks and reproduce them across multiple datasets.

In the field of curation and documentation, proposals such as *Datasheets for Datasets* and *Data Cards* show that the utility of a dataset depends strongly on standardized metadata about provenance, structure, limitations, and recommended use [Gebru et al., 2021; Pushkarna et al., 2022]. Although these proposals were formulated mainly for machine learning, their logic is broader: documentation is part of the artifact, not an optional appendix. In scientific collections, this observation is especially important because different experiments require a quick understanding of granularity, observational units, variables, temporal coverage, and data origin.

Platforms such as OpenML and Hugging Face Datasets have made data easier to access through large catalogs, shared metadata, and programmatic interfaces [Vanschoren et al., 2014; Lhoest et al., 2021]. OpenML is typically used for benchmarking workflows, while Hugging Face Datasets is more commonly used for large-scale data distribution. Both are effective in practice, although they emphasize different aspects of data access. The discussion here moves in a different direction by focusing on how similar forms of access can be embedded into a *package-native* interface within the same environment where experiments are carried out.

Package distribution is one of the practical points where this problem appears most clearly. In the R ecosystem, packages are commonly used to combine code, documentation, and reproducible examples in a single artifact [Wickham and Bryan, 2023]. This works well up to a point. CRAN, for instance, expects packages to remain small and discourages the inclusion of large datasets [CRAN Repository Maintainers, 2025]. PyPI imposes similar restrictions by setting storage limits for files and projects [PyPI Docs, 2026]. Reproducible benchmarking, therefore, depends on keeping data close to the analysis workflow rather than shipping the entire dataset in the package.

When these contributions are considered together, a gap becomes clearer. We have well-established principles for reproducibility and data documentation [Peng, 2011; Sandve et al., 2013; Wilkinson et al., 2016; Gebru et al., 2021; Pushkarna et al., 2022], and platforms that support discovery and large-scale sharing [Vanschoren et al., 2014; Lhoest et al., 2021]. At the same time, package ecosystems impose limits on how much data can be distributed [Wickham and Bryan, 2023; CRAN Repository Maintainers, 2025; PyPI Docs, 2026]. What is still missing is a clear description of how to bridge these aspects. In particular, there is little guidance on an intermediate architecture that links local usage contracts, compact dataset miniatures, remote access to full data, and package-level integration. Table 1 summarizes related work and positions *Eidolon* against them.

3. Eidolon Architecture

The *Eidolon* architecture supports the publication of datasets integrated with packages. *Eidolon* combines curation, standardization, structured miniaturization, and on-demand expansion into a single workflow that preserves consistent data access. The portion distributed with the package follows the same usage logic as the complete artifact, while the latter remains stored remotely. In this sense, the architecture serves as a publication protocol that begins with curation and standardization, introduces compatible miniatures, and still allows access to the full data as needed.

In software engineering, the notion of a *contract* commonly describes a stable interface between components that need to interoperate without exposing internal details,

Table 1. Comparison of data collection publication strategies.

Strategy	Advantage (▲) / Limitation (▼)	First access	Lightweight
Raw repository	▲ a simple way to keep original data available over time. ▼ leaves data consumption largely undefined and disconnected from execution environments.	Low	High
Package with complete data	▲ data is readily available at runtime, which simplifies reproducibility. ▼ quickly runs into repository constraints, especially regarding size and long-term maintenance.	High	Low
External portal/catalog	▲ facilitates large-scale discovery and reuse through shared metadata. ▼ using the data typically requires extra steps to bridge catalog entries and local code.	Medium	High
Eidolon architecture	▲ combines local access with compact dataset miniatures and deferred loading of full data. ▼ relies on an additional curation stage and on keeping different dataset forms in sync.	High	High

as discussed in both classical and more recent work [Bass et al., 2021; Perry and Wolf, 1992]. Here, the idea is applied in a more operational sense. The contract corresponds to a small set of properties that must remain aligned between the miniature and the complete artifact, such as dataset identity, structural schema, and key provenance and usage metadata. The mechanism that resolves access to the full data is also part of this definition. The two representations do not need to match in size. However, they are expected to behave consistently throughout the experimental workflow.

The protocol also introduces a set of checks to evaluate whether an instantiation follows these principles: whether the miniature and the full artifact are treated in practice as the same dataset; whether schema and essential metadata remain stable across both forms; whether remote expansion preserves the local consumption interface; and whether the collection remains close to the analysis environment even when the full data is not embedded in the distributed package.

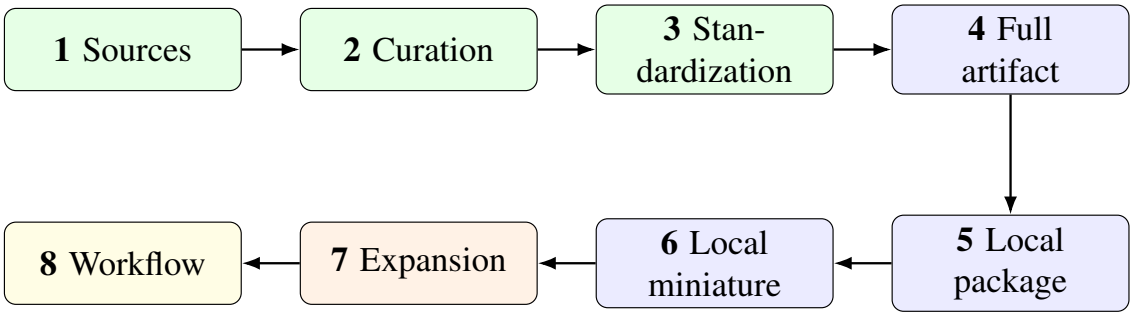

Figure 1. *Eidolon* operational flow: from source preparation to experimental use with on-demand expansion.

Figure 1 presents an operational view of the proposal, organized in eight steps across two rows. The first sequence (Step 1 – Step 4) covers sources, curation, standardization, and construction of the full artifact. The second sequence (Step 5 – Step 8) returns to the local environment, including the package, the local miniature, on-demand expansion, and the experimental workflow.

Step 3 (standardization) plays a key role in maintaining dataset consistency. Each dataset follows a shared contract that includes elements such as name, description, format, category, references, source, and a usage example. This choice follows the general direction of Gebru et al. [2021] and Pushkarna et al. [2022], which emphasize the importance of documenting data use. The metadata are intended both to support human interpretation and to guide how the data are consumed computationally.

The architecture's core behavior becomes clearer between Step 4 and Step 6. Instead of distributing the full dataset within the package, a smaller local miniature is provided. In the current instantiation, these miniatures are generated automatically from each full `.RData` object by retaining the first 30 rows of a `data.frame`, or the first element of a `list` and, when applicable, the first 30 rows of its first tabular component. An attribute stores the remote address of the complete artifact. The key point is that reducing data volume does not change how the dataset is used, so the local package and the full artifact remain logically aligned.

Step 7 is responsible for materializing the full artifact when necessary. In the case discussed here, this is done by a `loadfulldata()` function that reads the remote address stored in the miniature, downloads the corresponding `.RData` file, loads it in a temporary environment, and returns the referenced object. This step verifies whether the transition from the miniature to the full data occurs without an interface break in the experimental workflow represented in Step 8.

4. Instantiation in the *tspredit-tspredbench* Integration

This section illustrates how the protocol is instantiated in the context of time-series forecasting benchmarks and shows how curation, standardization, miniaturization, and on-demand expansion are embedded in the collection-package workflow.

In the current version of the collection, we identify 17 datasets from multiple data sources in the *tspredit-tspredbench* ecosystem [Salles et al., 2023; Ogasawara et al., 2025]. These objects cover classical competitions, macroeconomic series, and public-domain datasets, with standardized descriptions, source information, categories, references, and usage examples. Table 2 summarizes them.

The integration between *tspredit* and *tspredbench* provides a concrete illustration of the protocol. The collection brings together 17 curated objects, all documented under a common contract and published as lightweight miniatures within the package. The ETL repository makes the construction of the full `.RData` artifacts explicit, and a separate script generates the miniatures automatically. The `loadfulldata()` function reconstructs the full artifact from the miniaturized object, enabling discovery and initial inspection while preserving the link between the miniature and the complete data. Steps 1 to 6 of *Eidolon* are available at https://github.com/cefet-rj-dal/tspredbench/tree/main/etl. Finally, examples showing how to load the complete datasets using *tspredbench* (Step

Table 2. Summary of the *tspredit-tspredbench* integration.

Group	Content (►) / Role (♦)	Objects
Benchmarks	► *EUNITE*, *NN3*, *NN5*, *CATS*, *SantaFe*. ♦ methodological comparison and controlled experimentation.	8
Macroeconomics	► *ipeadata.d*, *ipeadata.m*. ♦ real-world series with applied use and daily and monthly frequencies.	2
Multi-domain data	► *bioenergy*, *climate*, *emissions*, *fertilizers*, *gdp*, *pesticides*. ♦ thematic diversity across environmental, agricultural, and economic domains.	6
Financial market	► *stocks*. ♦ series with strong practical demand in forecasting experiments.	1
Total	♦ collection organized under a common metadata and access contract.	**17**

7 of *Eidolon*) are available at https://github.com/cefet-rj-dal/tspredit/tree/main/examples/datasets.

This architecture has practical implications. Installation and initial exploration become less demanding because users can load objects, inspect examples, and understand the collection without immediately accessing the full data volume. At the same time, the experimental flow remains continuous because, when full data is required, expansion occurs without changing the programming model or introducing dependencies on external APIs. In the case study, the protocol checks are satisfied at the level of interface preservation, metadata stability, and proximity between the collection and the analysis environment.

The case study also highlights the importance of standardization in *Eidolon*. Without consistent naming, formats, descriptions, and examples, the value of miniaturization would be limited. Local inspection would still be possible, but there would be no clear guarantee that the objects correspond conceptually to the remote artifact.

5. Discussion

The case study shows that the integration between local miniature and on-demand expansion preserves the dataset consumption interface throughout the experimental workflow and keeps the collection close to the benchmark tool without embedding its full volume in the distributed package.

Table 1 helps frame the contribution of the architecture. Instead of presenting data publication as a choice between keeping everything in the package or keeping everything outside it, the discussion can proceed more gradually. Raw repositories and external portals emphasize storage, discovery, and community reach but often weaken the immediate link between the catalog, the usage contract, and the execution environment. Packages that include complete data offer strong local access but face clear limits in distribution and maintenance. The *Eidolon* architecture is positioned between these alternatives, preserving local discovery and a uniform contract while allowing the full data to remain remote and be accessed on demand.

The architecture is particularly suited to cases in which the collection is too large

to be fully embedded in the package but still needs to remain close to the analysis environment. This advantage is evident in the analyzed case study: the *tspredbench* repository occupies about 152.5 MiB, whereas the `data/` folder of the *tspredit* package occupies about 64.6 KiB, corresponding to an approximate reduction of 2416x. Its advantages become less clear when the collection is small enough to be distributed at low cost, when there is no reliable infrastructure to host the full artifact remotely, or when compatible miniatures cannot be constructed without affecting practical use.

Although the instantiation presented here focuses on time series, the architecture does not depend on time-specific semantics, but rather on the structural requirements of the publication protocol: a stable schema, the possibility of compatible miniaturization, and a reliable mechanism for resolving the full artifact. For this reason, it can be applied to other scientific domains in which the package or library functions as the natural analysis environment and the data collection is large enough to make full embedding undesirable.

This generality, however, does not eliminate the limitations. First, miniaturization introduces an additional maintenance cost because miniatures, metadata, and full artifacts must evolve in synchrony. Second, on-demand expansion depends on stable external infrastructure to host and provide the full dataset. Third, the current implementation resolves remote files by URL but does not yet enforce checksum validation or version pinning, which limits stronger guarantees of long-term reproducibility. Finally, the paper prioritizes architectural evidence through instantiation, leaving room for future studies on operational performance, version governance, and replication across other ecosystems, including Python.

6. Conclusion

The *Eidolon* architecture supports the reproducible publication of scientific data collections in Data Analytics ecosystems. Its viability depends on organizing the collection around a stable usage contract, supported by compatible local miniatures and explicit mechanisms for accessing the full data remotely.

Several directions remain open for further exploration, including explicit versioning and provenance mechanisms between miniatures and full artifacts, extension beyond the R ecosystem, and evaluation in different usage contexts.

Acknowledgments

The authors thank CAPES, CNPq, and FAPERJ for partially supporting this research.

Use of AI

Generative AI tools were used only for linguistic revision, with a focus on textual clarity. The authors reviewed all suggestions and remain fully responsible for the paper's content.

References

Bass, L., Clements, P., and Kazman, R. (2021). *Software Architecture in Practice*. Addison-Wesley Professional.

CRAN Repository Maintainers (2025). CRAN Repository Policy. Technical report, https://cran.r-project.org/web/packages/policies.html.

Gebru, T., Morgenstern, J., Vecchione, B., Vaughan, J. W., Wallach, H., Iii, H. D., and Crawford, K. (2021). Datasheets for datasets. *Communications of the ACM*, 64:86 – 92.

Hugging Face (2026). Big data? Datasets to the rescue!

Kane, M. J., Emerson, J. W., Haverty, P., and Determan, C. (2025). *bigmemory: Manage Massive Matrices with Shared Memory and Memory-Mapped Files*. CRAN.

Lhoest, Q., del Moral, A. V., Jernite, Y., Thakur, A., von Platen, P., Patil, S., Chaumond, J., Drame, M., Plu, J., Tunstall, L., Davison, J., Šaško, M., et al. (2021). Datasets: A Community Library for Natural Language Processing. In *EMNLP 2021 - 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175 – 184.

Ogasawara, E., Alexandrino, F., Gea, C., Santos, D., Salles, R., Birindiba, V., Pacheco, C., Bezerra, E., Pacitti, E., Porto, F., and Carvalho, D. (2025). A Benchmark Time Series Dataset Collection for Prediction Models.

Peng, R. D. (2011). Reproducible research in computational science. *Science*, 334:1226 – 1227.

Perry, D. E. and Wolf, A. L. (1992). Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, 17:40–52.

Pushkarna, M., Zaldivar, A., and Kjartansson, O. (2022). Data Cards: Purposeful and Transparent Dataset Documentation for Responsible AI. In *ACM International Conference Proceeding Series*, FAccT '22, pages 1776 – 1826, New York, NY, USA. Association for Computing Machinery.

PyPI Docs (2026). Storage Limits. Technical report, https://docs.pypi.org/project-management/storage-limits/.

Salles, R., Pacitti, E., Bezerra, E., Marques, C., Pacheco, C., Oliveira, C., Porto, F., and Ogasawara, E. (2023). TSPredIT: Integrated Tuning of Data Preprocessing and Time Series Prediction Models. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 14160 LNCS:41 – 55.

Sandve, G. K., Nekrutenko, A., Taylor, J., and Hovig, E. (2013). Ten Simple Rules for Reproducible Computational Research. *PLoS Computational Biology*, 9.

Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. (2014). OpenML: networked science in machine learning. *SIGKDD Explor. Newsl.*, 15:49–60.

Wickham, H. and Bryan, J. (2023). *R Packages: Organize, Test, Document, and Share Your Code*. "O'Reilly Media, Inc.".

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., Bouwman, J., Brookes, A. J., et al. (2016). Comment: The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3.