SLHDA_MVP: A Scalar-Ledger Hybrid Data Architecture for Flexible, Efficient Multi-Scale Numerical Data Management

Abstract:
Scientific computing and HPC simulations often produce large, high-dimensional numerical data that must be stored, manipulated, and reconstructed across different scales (e.g. grids, sub-grids, flattened vectors). Traditional array/file formats — while powerful — face limitations when workflows require frequent reshaping, aliasing, semantic tagging, and flexible retrieval patterns. We propose **SLHDA_MVP**, a minimal yet versatile storage architecture that (a) stores all numerics contiguously as scalars, (b) maintains a lightweight JSON-ledger mapping metadata (shape, tags, offsets), and (c) enables zero-copy tensor aliasing, semantic queries, and batch retrievals. We demonstrate that SLHDA addresses key pain points in multi-scale simulations and data-heavy HPC workflows. We discuss performance characteristics, trade-offs, and potential extensions.

1. Introduction & Motivation:

Modern HPC applications — in computational physics, climate modeling, fluid dynamics, parameter sweeps — often produce heterogeneous data: scalar fields, multi-dimensional grids, nested tensor structures, derived vectors, sub-regions, etc. Moreover, workflows may require:

- dynamic reinterpretation of data (reshaping, aliasing),
- semantic tagging (e.g. “pressure field”, “temperature grid”, “simulation step 42”),
- flexible querying by semantic tags or by structural form,
- batch retrieval of many objects at once, and
- memory/layout efficiency for large-scale data (millions of scalars).

Existing scientific storage formats such as HDF5 (and by extension netCDF-4) are widely adopted because they allow hierarchical organization, multi-dimensional arrays, and metadata embedding. NASA Earthdata+2NEON Science+2

However, several limitations arise when using them for highly dynamic, alias-heavy, or multi-scale workflows:

- **Metadata fragmentation and overhead:** In complex datasets with many variables or nested groups, metadata becomes scattered across the file, leading to inefficiencies when frequently inspecting or modifying metadata. NSIDC GitHub Pages+1
- **Rigid structure and limited reinterpretation:** Once a dataset is defined (shape, type), reinterpreting the same data as a different tensor shape — without copying or rewriting — is non-trivial. Changing shape usually requires duplication or rewriting.
- **I/O performance issues under heavy aliasing / small-block operations:** HPC workflows that frequently read many small blocks, sub-regions or alias views may suffer from overhead due to chunking, metadata access, or inefficient layout. PMC+1
- **Concurrency and parallel access limitations:** Some libraries built on HDF5 have constraints: global locks, difficulty managing parallel I/O, and risks of file corruption under parallel access. Unidata+1
- **Lack of semantic tagging / dynamic querying:** While HDF5 supports attributes and groups, it does not natively provide a flexible, uniform mechanism for semantic tagging and querying across datasets with different shapes, aliases, or versions.

Because of these, researchers sometimes resort to ad-hoc solutions: separate sidecar databases, bespoke file-naming conventions, custom indexing layers — which complicate workflows, increase maintenance, and reduce portability or reproducibility.

Thus, there remains a gap: a **simple, flexible, minimalistic storage layer** tailored for **dynamic, multi-scale numerical data workflows** that supports zero-copy aliasing, semantic tagging & querying, and efficient storage without heavy overhead.

2. Related Work & Existing Approaches:

- **HDF5 / netCDF-4 / PnetCDF:** These formats remain the de facto standard for multi-dimensional array storage, with support for hierarchical datasets, variable types, compression, chunking, parallel I/O (MPI-IO), and metadata. NASA Earthdata+2HDF Group+2

- Various domain-specific optimized formats: for example, use of specialized storage layouts or custom I/O strategies to optimize for small-block reads/writes in HPC simulations. PMC+1
- Emerging cloud-optimized / chunked array formats (e.g. Zarr, cloud-optimized NetCDF/HDF5) aiming to improve parallel access and object-store compatibility. OUP Academic+1

Despite these, none of the standard general-purpose formats focus on the combination of **scalar-flat storage + metadata ledger + zero-copy aliasing + semantic tagging + batch queries**. SLHDA_MVP is designed to fill exactly that niche.

3. Design and Implementation of SLHDA_MVP:

3.1 Storage Model: Flat Scalar Store + Metadata Ledger

- **Scalar Store (scalar.bin)**
  - All numerical data is stored as a contiguous sequence of 64-bit floats (float64).
  - There is no concept of "dataset shape" at storage time — only a linear block of scalars.
  - This ensures memory layout is compact and predictable, and avoids fragmentation or hierarchical file overhead.
- **Metadata Ledger (ledger.json)**
  - A JSON file mapping object-IDs (UUIDs) to metadata entries:
    - start: index (offset) of the first scalar in the store
    - length: number of scalars for this object
    - shape: intended tensor shape for reconstruction
    - tags: a list of semantic tags (strings) for flexible querying
  - This ledger acts as an "index / table of contents / catalog" separate from the data payload.

This separation simplifies the underlying store and moves all metadata to a human-readable, easily editable ledger. It avoids hierarchical file structures and keeps I/O minimal and predictable.

3.2 Dynamic Alias & Zero-Copy Reconstruction

Because the ledger maps offsets and shapes, the same block of scalars can be reinterpreted (aliased) into different tensor shapes — without copying or rewriting data. For example:

- a 4×4 grid of 16 scalars can be stored once, then aliased as:
  - a flat vector of length 16, or
  - a subgrid (e.g. 2×2×4), or
  - other shapes as needed.

This aliasing is zero-copy: only metadata changes (new UUID in ledger), data remains in place. This allows flexible, multi-scale views and reuse of the same data for different purposes (grid representation, vectorization, sub-block views, etc.).

3.3 Semantic Tagging & Querying

Because each object in the ledger carries tags, SLHDA enables semantic queries: e.g., fetch all data objects with tag "simulation", or "pressure", or "vector", independent of shape or storage offset. This supports workflows where data objects are logically grouped by semantic meaning (rather than just by filenames or dataset names).

Batch retrieval: once matching object-IDs are found by tag query, the interpreter can load and reconstruct all arrays in a batch, which is convenient for workflows that need to operate on many objects at once.

3.4 Practical Implementation

- Language: Python 3, using numpy for numeric handling.
- Package structure: clean, minimal, installable via setup.py.
- Demo & usage scripts / notebook provided for easy onboarding.
- No external dependencies besides numpy — this ensures portability.

4. Gap Analysis: What Problems We Overcome:

Challenge in existing systems	Why it's a problem	How SLHDA helps
Complex hierarchical formats with metadata scattered / fragmented (e.g. HDF5 with many variables/groups) NSIDC GitHub Pages+1	Metadata access becomes slow; modifying or reorganizing data is cumbersome	Single JSON ledger simplifies metadata, easy to edit, search, and maintain
Rigid dataset definitions — shape/type fixed; aliasing requires duplication or rewrite	Limits dynamic workflows, wastes storage/time when reinterpreting data	Zero-copy aliasing offers flexible reinterpretation without duplication
Overhead for small-block reads / many small writes typical in multi-scale or subgrid simulations PMC+1	I/O overhead and performance degradation in HPC / cluster environments	Flat scalar store ensures predictable layout; ledger avoids per-chunk metadata overhead
Lack of semantic querying / ad-hoc naming conventions or external catalogs	Hard to manage datasets semantically; error-prone manual bookkeeping	Tags + ledger enable flexible, robust semantic querying and automated cataloging
Parallel I/O / concurrency constraints in standard libraries (global locks, risk of corruption) Unidata+1	Limits scalability in large HPC / distributed settings	Simpler store format may integrate more easily with custom parallel I/O or distributed object stores (future work)

In short, SLHDA attempts to reduce complexity, maximize flexibility, and support multi-scale, dynamic workflows — exactly where classical formats start to struggle or require heavy boilerplate.

5. Experimental Demonstration & Performance:

We conducted a simple performance experiment using SLHDA_MVP:

- Stored a large array of **1 million float64 scalars**
- Measured time to store and retrieve the entire array (single read/write)
- Demonstrated aliasing: e.g. storing a 4×4 grid, then aliasing it to multiple shapes (vector, subgrid) without additional write operations
- Demonstrated semantic queries: retrieving all objects with a given tag (e.g. "simulation"), retrieving multiple arrays at once (batch retrieval)

Results (on modern hardware): storing and retrieving 1 M scalars took on the order of **milliseconds to low seconds**, illustrating that flat-scalar storage and binary I/O remain efficient even at large scale.

These simple experiments serve as feasibility demonstrations; full benchmarking against HDF5 / netCDF / Zarr / chunked formats is future work.

6. Discussion & Trade-offs:

While SLHDA offers flexibility and simplicity, it comes with trade-offs:

- **No built-in compression / chunking / hierarchical grouping:** Unlike HDF5, SLHDA does not compress data or organize it in hierarchical groups — that can lead to larger file sizes and less organized structure for very complex datasets.
- **Loss of built-in cross-language or cross-tool support:** While HDF5 and its derivatives are widely supported (Python, MATLAB, GIS tools), SLHDA is custom and currently only Python + numpy.
- **No built-in concurrency / parallel I/O / distributed storage:** SLHDA in its MVP form is single-process; extending it to multi-process or distributed HPC setups will require additional work.
- **Reliance on external metadata file (ledger.json):** If ledger is lost or corrupted, reconstructing data may be difficult, unlike self-contained HDF5 files.

Nevertheless, for many applications — especially research workflows, prototyping, multi-scale simulations, dynamic data reinterpretation — SLHDA's design trade-offs may be acceptable or even advantageous.

7. Future Work:

Possible extensions and improvements:

1. **Compression and chunked storage layer** — add optional compression or chunking while retaining ledger-based metadata.
2. **Parallel I/O / distributed storage support** — integrate with object stores or distributed file systems; support concurrency, locking, or multi-process access.
3. **Language / tool bindings beyond Python** — e.g. C++, Julia, Fortran; interface with HPC simulation codes.
4. **Metadata versioning and robustness** — resilient ledger storage, transactional updates (in case of crashes), checksums for integrity.
5. **Hybrid workflow with standard formats** — allow export/import to/from HDF5/netCDF/Zarr for interoperability, while retaining SLHDA for internal workflow efficiency.

8. Conclusion:

SLHDA_MVP offers a simple yet powerful approach to numerical data storage, targeted at workflows that require **dynamic data reinterpretation, multi-scale aliasing, semantic tagging, and efficient batch retrieval**. By separating raw scalar storage from metadata (ledger), it enables flexible, zero-copy aliasing and practical semantic queries — addressing real pain points in multi-scale simulation and data-heavy HPC workflows. While not a universal replacement for established formats such as HDF5 or netCDF, SLHDA fills a niche where flexibility and simplicity outweigh the benefits of compression, complex metadata, or established tooling.

We believe this architecture may be particularly useful in research workflows, rapid prototyping, and custom simulation pipelines — and with further development, potentially scalable to larger HPC or distributed environments.

References

- HDF5 — The HDF5 Data Model, File Format and Library. NASA Earthdata documentation. NASA Earthdata+1
- Delaunay, X., Courtois, A., & Gouillon, F. (2019). Evaluation of lossless and lossy algorithms for the compression of scientific datasets in netCDF-4 or HDF5 files. *Geoscientific Model Development*. gmd.copernicus.org
- Godoy, W. F., Savici, A. T., Hahn, S. E., Peterson, P. F., et al. (2021). Efficient loading of reduced data ensembles produced at ORNL SNS/HFIR facilities. *arXiv preprint*. arXiv
- Torabzadehkashi, M., Bobarshad, H., Alves, V., Bagherzadeh, N., et al. (2019). Computational storage: an efficient and scalable platform for big data and HPC applications. *Journal of Big Data*. SpringerLink+1