Semantically-Aligned Data Carving for Robust Reproducibility

Raffay Atiq*
SRI
Menlo Park, CA, USA
raffay.atiq@sri.com

Ashish Gehani
SRI
Menlo Park, CA, USA
ashish.gehani@sri.com

Tanu Malik
School of Computing
University of Missouri
Columbia, MO, USA
tanu@missouri.edu

Fareed Zaffar
Lahore University of Management Sciences
Lahore, Pakistan
fareed.zaffar@lums.edu.pk

Abstract

Reproducibility is integral to scientific progress. Containerization facilitates scientific reproducibility but often results in excessive storage footprints when applied to data-intensive workflows, as entire datasets are frequently packaged despite only partial access at runtime. While data carving can mitigate this, distinct file formats typically require specialized reduction tools. This paper demonstrates that a unified data carving architecture is achievable for distinct formats when a principled mapping exists between their underlying data models. Focusing on the structural correspondence between HDF5 and netCDF4, we introduce a framework that leverages this mapping to interpose on library calls and selectively extract only the format-defined data objects accessed during execution. By abstracting the carving logic based on the shared data model, our system produces slimmed-down datasets containing only the necessary files and data objects, regardless of which high-level API the application utilizes. Evaluation on geophysical and climate workloads confirms that this unified approach substantially reduces container size while maintaining application correctness, incurring moderate I/O overhead during the initial carving phase and minimal overhead during re-execution.

CCS Concepts

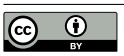
• **Information systems** → **Data management systems**; *Data exchange*; *Extraction, transformation and loading*; • **Networks** → **Cloud computing**; • **Software and its engineering** → *Software configuration management and version control systems.*

Keywords

Reproducibility, containerization, data carving, array-based formats, HDF/NetCDF.

ACM Reference Format:
Raffay Atiq, Ashish Gehani, Tanu Malik, and Fareed Zaffar. 2026. Semantically-Aligned Data Carving for Robust Reproducibility. In *Practice and Experience in Advanced Research Computing (PEARC '26), July 26–30, 2026, Minneapolis, MN, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3785462.3815793

*While visiting SRI.



PEARC '26, Minneapolis, MN, USA

ACM ISBN 979-8-4007-2377-3/26/07
https://doi.org/10.1145/3785462.3815793

1 Introduction

Reproducibility in scientific computing is paramount for validating results. However, modern data-intensive workflows often struggle with unwieldy data volumes. Data sourced from satellite observations or generated by large-scale modeling efforts can be prohibitively large and are frequently packaged in data formats like HDF5 [9] or netCDF4 [24]. These hierarchical, self-describing formats group semantically distinct data objects into one file, making data exploration and metadata organization more streamlined. Application code, however, may access only a fraction of these data objects. Although only a subset of these semantically separated objects may be needed for downstream tasks (such as analyzing specific geographic regions or time slices), the inherent structure of HDF5 and netCDF4 files results in including the entire data bundle. When scientific workflows containerize application code and data for portability and reproducibility, subsets of the files may remain unused, resulting in bloated container sizes and unnecessarily high distribution and deployment overheads.

Consider a large-scale geophysical dataset in netCDF4 or HDF5 format containing multiple semantically distinct data objects, such as ones for latitude, longitude, temperature, pressure, and relative humidity. Each of these data objects resides in its own section of the file's hierarchical structure. In one workflow, perhaps examining seasonal patterns, only temperature data, along with latitude and longitude, may be needed to compute specific metrics and generate visualizations. Though the application ignores other data objects like pressure or relative humidity, they remain bundled in the same file. When this workflow is packaged into a container for reproducibility, all data objects, even those never accessed, are included. As a result, transferring or storing the container incurs larger-than-necessary overhead, also known as *container bloat*. A more efficient approach would include just the latitude, longitude, and temperature data objects, thereby significantly reducing container size without sacrificing the accuracy or reusability of the application.

We present SemCarve, a framework that transparently embeds specialized behavior into application input and output (I/O) calls to reduce container bloat. This is done by monitoring file and data object accesses. SemCarve produces carved files containing only the data objects actually accessed. Since our focus is on HDF5 and

netCDF4 data formats, we consider specialization of I/O calls by interposing at the format-specific I/O library level. The carved data includes all format-specific data objects used during the original run. If an application re-execution later invokes eliminated objects, a fallback mechanism ensures correctness by dynamically retrieving the missing data.

In previous work [35], we demonstrated the efficacy of format-specific, object-level interception for HDF5. By intercepting I/O library calls to identify and extract entire data objects, we established a robust method for reducing dataset size while maintaining semantic validity. Since other prominent formats like netCDF4 share this underlying data model, we now investigate whether a unified carving framework can span multiple scientific data formats whenever a principled mapping exists between them. HDF5 and netCDF4 are a compelling pair to study because netCDF4 is layered atop HDF5, yielding shared storage and overlapping abstractions. We show in this paper that such a unified framework is feasible provided the design simultaneously exploits commonalities in the data models and accounts for each format's constraints.

Specifically, the paper makes the following contributions:

- Existing documentation largely treats netCDF4's reliance on HDF5 as an opaque implementation detail. We provide a systematic description of this mapping and use it to derive critical design principles for a unified carving strategy capable of reducing storage footprints in both formats (rather than just HDF5 in prior work [35]). (See Section 4.)
- We develop an approach that transparently identifies and extracts complete data objects that are read during application execution through interception at the I/O library level. This facilitates more robust data availability during container re-execution even if the application input parameters vary (since entire data objects are present, unlike prior work [16]). (See Section 5.)
- We evaluate our system on realistic workloads from multiple scientific domains. The diversity of the data and application access patterns highlight the carving framework's broad applicability. (See Section 6.)
- We make our code available under a permissive open source license [5, 6].

Our results indicate that SemCarve can reduce container sizes by up to 94% in realistic workloads, while preserving reproducibility and incurring moderate I/O overhead when creating carved files and minimal I/O overhead when reusing them.

2 Related Work

We focus on related work primarily relating to debloating.

Data-based Debloating. Existing tools address container bloat [36] via block-level deduplication (Slacker [11]) or whole-file lifting (LLIO [26]). SemCarve advances this by targeting structured array formats (HDF5 [9], netCDF4 [24]) to lift only utilized data subsets. Tikmany *et al.* [35] demonstrate access-guided debloating by performing system-level interception of I/O calls and object-level interception of HDF5 operations to carve HDF5 files, whereas SemCarve adds netCDF4 support and comprehensive metadata preservation. Additionally, unlike SCALPEL [1], which prunes data at the sub-dataset level, we emphasize complete dataset object carving to ensure correctness across varying execution parameters.

Code-based Debloating. Approaches to code bloat [12, 13, 37] and configuration debloating (OCCAM [14], Trimmer [25]) focus on reducing binary size, often building on C/C++ or LLVM bitcode. Unlike these code-centric methods, SemCarve explores data-based debloating to improve storage efficiency and reproducibility.

Program Specialization. A prior work on program specialization by Medicherla *et al.* [15] has been used to verify whether or not a program 'conforms' to the specified format and to specialize the code to the 'restricted' file format. This is program specialization being used for verification. As opposed to this, program specialization is being used for data reduction in SemCarve.

Lineage Methods. A precise lineage model informs about data flows of an application. There are several provenance models for capturing system call lineage [29]. System-event trace analysis processes are also used in other whole system provenance tracking methods [2, 10, 28]. In contrast, our library call interception approach enables identification of high-level semantic data objects, offering a more domain-focused means of monitoring data flow.

3 Background

Scientific data is predominantly stored as multi-dimensional arrays, with HDF5 and netCDF4 serving as popular formats. HDF5 offers a general-purpose hierarchy of groups and datasets with minimal semantics, essentially storing a climate dataset as a collection of arbitrary arrays. In contrast, netCDF4 enforces a semantic model that interprets arrays as either dimensions (the domain, e.g., latitude/longitude) or variables (the range, e.g., temperature), explicitly encoding the functional relationships between them.

Since netCDF4 employs HDF5 as its underlying storage layer, its datasets naturally form a strict subset of HDF5 structures. This subset relationship implies that a carving framework designed for HDF5 should theoretically extend to netCDF4. However, treating netCDF4 data strictly as HDF5 objects is insufficient—validity depends critically on preserving consistent netCDF4 metadata. The mapping described in Section 4 is required to recover dimension–variable relationships; without it, carving yields semantically invalid netCDF4 outputs.

3.1 HDF5 (Hierarchical Data Format, Version 5)

Figure 1 shows an example HDF5 file representing a simple climate dataset containing arrays for latitude, longitude, temperature, pressure, and relative humidity. At a high level, HDF5 organizes data objects within a hierarchical structure similar to a Unix file system, but encapsulated within a single, portable file. This structure allows all data and associated metadata to be stored together, enabling self-contained, cross-platform access.

In HDF5, data objects are organized as nodes in a directed graph [33], which users typically navigate as a hierarchy rooted at a distinguished root group, denoted by "/". In the example shown, all five arrays are stored as datasets directly under the root group, and each dataset is annotated with attributes such as physical units. Beyond this organization, HDF5 does not assign any intrinsic semantic meaning to the datasets or their relationships. The primary HDF5 data objects appearing in the example are:

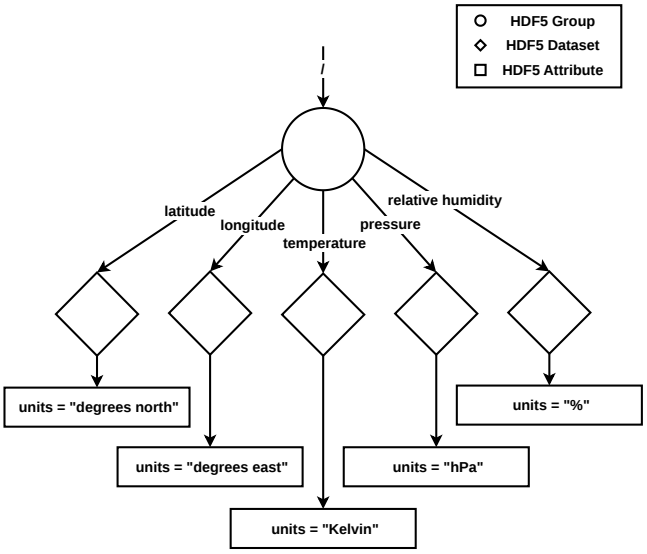


Figure 1: An example HDF5 file containing datasets for latitude, longitude, temperature, pressure, and relative humidity. Each dataset is organized under the root group (/) and annotated with metadata in the form of HDF5 attributes (e.g., units).

Groups. Groups act as containers that organize datasets and other groups, forming the hierarchical structure of the file. Every HDF5 file contains a root group ("/"), which serves as the entry point for navigating the file's contents. Additional groups may be introduced to structure data logically, though HDF5 imposes no constraints on how groups should be used.

Datasets. Datasets store the primary data as multidimensional arrays [32]. Each dataset is defined by a *datatype*, which specifies the representation of individual array elements (e.g., integer, floating point, string, or compound types), and a *dataspace*, which defines the array's rank and the extent of each dimension. In the example, each physical quantity—latitude, longitude, temperature, pressure, and relative humidity—is represented as a separate dataset, with shapes appropriate to the underlying grid or sampling structure.

Attributes. Attributes are named metadata objects attached to groups or datasets and are the primary mechanism by which HDF5 files are self-describing [31]. Conceptually, an attribute is a small dataset with its own datatype and dataspace. In the example, attributes are used to record contextual information such as units, but HDF5 allows arbitrary metadata to be attached without restriction.

Crucially, while HDF5 provides a rich and flexible mechanism for storing arrays and metadata, it does not impose domain-specific semantics on how datasets relate to one another. In the example, the five datasets coexist within the same file, but HDF5 does not encode which datasets represent independent variables (e.g., latitude and longitude), which represent dependent variables (e.g., temperature or pressure), or how these arrays should be interpreted together. Such relationships, if they exist, must be inferred or imposed by the application. This semantic neutrality gives HDF5 its generality and flexibility, but it also places the burden of interpretation entirely on the consuming software.

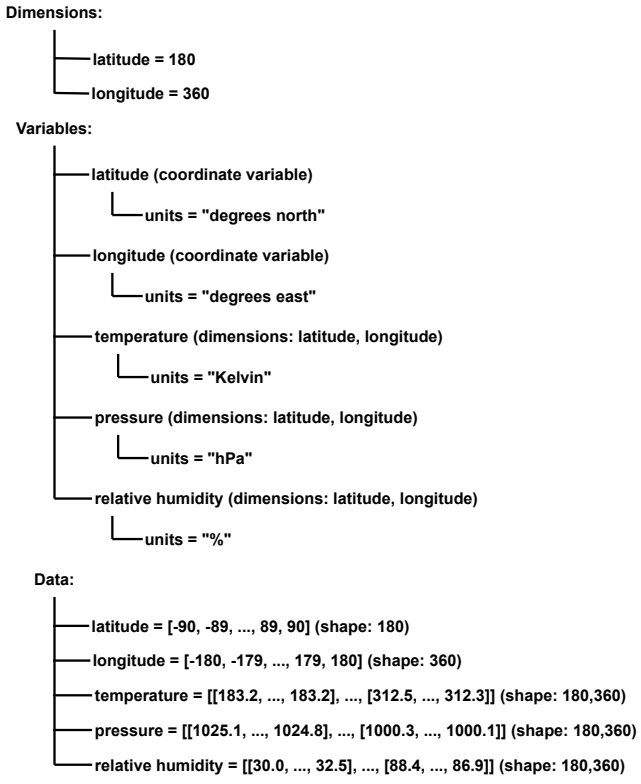


Figure 2: An example netCDF4 file with the same meteorological variables as in Figure 1. The file defines dimensions (latitude, longitude), coordinate variables, and multidimensional variables (temperature, pressure, relative humidity) with explicit domain-range mappings. Attributes (e.g., units) make the file fully self-describing.

3.2 netCDF4 (network Common Data Form, Version 4)

Figure 2 shows a netCDF4 file representing the same climate dataset as in the HDF5 example, with arrays for latitude, longitude, temperature, pressure, and relative humidity. Unlike HDF5, which leaves relationships among arrays implicit, netCDF4 is designed to explicitly encode the semantic relationship between the *domain* and *range* of the data.

Conceptually, netCDF4 models scientific data as a function defined over a discrete domain: given a set of points D, the data assigns values from a range R such that $f : D \rightarrow R$ [17]. In netCDF4, this abstraction is realized directly in the file format. *Dimensions* define the domain—the index spaces over which data is defined—while *variables* represent the range values associated with those domains. The primary netCDF4 data objects illustrated in the example are:

Groups. Groups organize related data objects and metadata, providing a hierarchical namespace similar to that of HDF5 [20]. Every

netCDF4 file also defines an implicit root group ("/"), under which dimensions, variables, and attributes are declared.

Dimensions. A dimension is a named axis definition that specifies the length of an index space used by one or more variables [19]. Dimensions are shared objects: multiple variables may reference the same dimension to ensure consistent alignment along a common axis, such as time, latitude, or longitude. Dimensions may correspond to physical coordinates or to abstract indices.

Variables. A variable is a named, typed array whose shape is defined by an ordered list of dimensions [21]. By construction, variables reference previously defined dimensions, which determines both their rank and axis ordering. Variables that share dimensions are therefore implicitly aligned over the same domain. A one-dimensional variable with the same name as a dimension is a *coordinate variable*; it stores the coordinate values for that axis and provides the mapping from array indices to physical coordinates.

Attributes. Attributes are small, typed metadata objects attached to variables or groups and are used to make the file self-describing [18]. Variable attributes describe how array values should be interpreted (e.g., units or missing-value conventions), while group-level attributes capture metadata about collections of variables, such as provenance or dataset-level conventions.

In the example, latitude and longitude are first declared as dimensions, establishing the domain of the data. Coordinate variables then provide the actual spatial values associated with these dimensions. Variables such as temperature, pressure, and relative humidity are defined over the latitude–longitude domain by referencing the corresponding dimensions, forming multi-dimensional arrays whose semantics are unambiguous. Attributes further annotate these variables with units and contextual metadata. Together, these constructs enforce a clear and explicit domain–range relationship, making the structure and meaning of the data self-evident and machine-interpretable.

4 Mapping netCDF4 to HDF5

The netCDF4 data format is implemented on top of HDF5, using HDF5 as its underlying storage layer [22], as illustrated in Figure 3. This layered design enables a degree of interoperability between the two formats. In particular, every netCDF4 file is a valid HDF5 file, therefore any HDF5 application can syntactically read it. However, such an application cannot inherently interpret the semantic structure imposed by the netCDF4 data model—most notably the domain–range relationships defined by dimensions and variables-without additional knowledge. Assigning meaning to the stored data objects, such as identifying which objects represent domains and which represent ranges, is therefore the responsibility of the HDF5 application.

In contrast, a netCDF4 application can read an HDF5 file only if it satisfies a set of compatibility constraints. These constraints govern how netCDF4 dimensions and variables are represented in HDF5. Specifically, the file must contain the required metadata on HDF5 data objects—namely groups, datasets and their attributes—that identify dimension and variable entities in netCDF4 data model, and encode the associations between them in a form implicitly understood by the netCDF4 library.

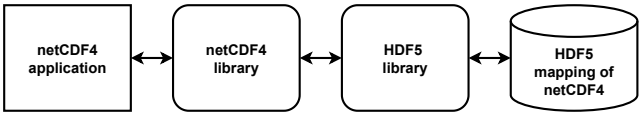


Figure 3: The netCDF4 data format is implemented on top of HDF5 by using the HDF5 library as its storage backend. A netCDF4 application accesses data through the netCDF4 library, which internally relies on the HDF5 library to perform low-level I/O operations.

Furthermore, this metadata must be internally consistent and unambiguous, correctly representing the bidirectional relationship between dimensions and variables. From the variable perspective, the metadata specifies the named dimensions that define each variables axes and their ordering (e.g., axis 0 corresponds to latitude and axis 1 corresponds to longitude). From the dimension perspective, the metadata defines the dimension entities, along with any associated coordinate descriptions, and records the variables that reference each dimension, including the axis position at which they are used. Consistency requires that these two perspectives agree: the variable's axis-to-dimension associations must resolve to the correct dimension definitions, and the implied ordering must match the variable's in-memory array layout. Any inconsistency, such as swapped axis order, missing/incorrect references, or conflicting definitions, breaks the intended dimension–variable relationship and leads to incorrect interpretation of coordinates or rendering the file nonconformant with netCDF4 conventions.

When these constraints are satisfied, a netCDF4 application can correctly parse the HDF5 file's structure and automatically infer the intended data semantics.

4.1 Representing netCDF4 semantics with HDF5

Although netCDF4 uses HDF5 as its storage layer, the netCDF4 data model is not inherent in HDF5's generic hierarchy of groups, datasets, and attributes. Instead, netCDF4 semantics are encoded by organizing HDF5 objects in prescribed ways and by attaching specific metadata that identifies netCDF4 constructs and their relationships. Most critically, this metadata specifies which datasets correspond to variables, which objects represent dimensions, and how the axes of each variable are defined and ordered. This section describes these conventions and the associated HDF5 structures expected by a netCDF4 library, enabling it to reconstruct the netCDF4 abstraction of the file, including its dimensions, variables, attributes, and groups, from the underlying HDF5 objects and metadata.

Figures 1 and 2 illustrate why these conventions are necessary. Both store the same scientific arrays and descriptive attributes, but the netCDF4 file in Figure 2 makes the dimension-variable relationships explicit, whereas the HDF5 file in Figure 1 contains only a generic collection of groups, datasets, and attributes. Consequently, the netCDF4 interpretation cannot be recovered from the HDF5 objects alone in a reliable, unambiguous manner: without netCDF4-specific metadata, there is no principled basis for determining which datasets represent dimensions (and coordinate variables), which datasets represent variables, or how each variable's axes map to and are ordered by dimensions. The conventions described in this section provide this missing information by representing dimensions

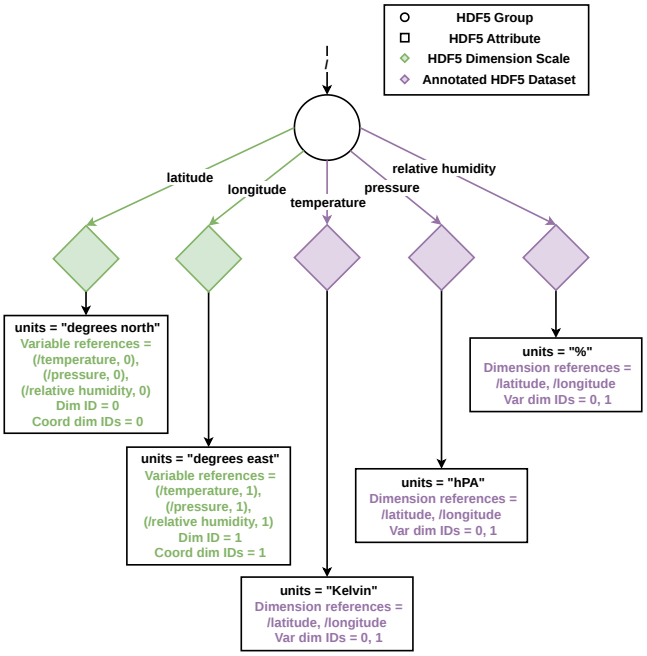


Figure 4: An HDF5 representation of the netCDF4 file shown in Figure 2. Dimension scales (in green) and annotated datasets (in purple) are associated with both standard metadata (e.g., units) and netCDF4-specific metadata, including variable references, dimension references, dimension IDs, coordinate dimension IDs, and variable dimension IDs. This mapping preserves netCDF4 semantics.

using *dimension scales*, assigning persistent identifiers to dimensions, encoding variable–dimension associations via HDF5 object references, and maintaining axis ordering by recording the ordered list of dimension identifiers. These conventions enable a netCDF4 library to reconstruct the netCDF4 structure from the underlying HDF5 representation.

In the following paragraphs, we describe each convention in turn and highlight the key metadata and object relationships. Throughout this section, Figure 4 serves as a running example of an HDF5 representation that encodes the netCDF4 structure of Figure 2 via netCDF4-specific metadata.

4.1.1 netCDF4 Groups → HDF5 Groups. Both netCDF4 and HDF5 employ the notion of groups to organize data in a hierarchical structure. This one-to-one correspondence means that a netCDF4 group aligns directly with an HDF5 group. However, netCDF4 leaves the root group implicit, while HDF5 makes it a tangible object in the file.

In Figure 4, this correspondence is reflected by the fact that all objects appear under the HDF5 root group, which serves as the container for the netCDF4 root group in Figure 2.

4.1.2 netCDF4 Dimensions → HDF5 Dimension Scales. NetCDF4 dimensions define the independent axes (domain) for variables. To represent this in HDF5, netCDF4 uses HDF5 *dimension scales* - specialized datasets annotated with metadata to track dimension usage and identity. This metadata includes:

- **Variable references.** This attribute links the dimension scale to the variables (datasets) that use it. It stores a list of HDF5 object references alongside integer indices, explicitly recording which axis of a variable is bound to this specific dimension scale.
- **Dimension ID.** To preserve identity and ordering, netCDF4 assigns a unique zero-based integer ID to each dimension. This identifier is stored as a scalar attribute on the dimension scale, ensuring the dimension's identity persists distinct from its name or path.
- **Coordinate-variable dimension IDs.** For dimensions that have associated coordinate variables, this attribute stores an ordered integer array of the corresponding netCDF4 dimension identifiers—containing the dimension's own ID for 1D coordinates or a sequence for multi-dimensional ones.
- **Other metadata.** Dimension scales are additionally marked and named using standard HDF5 dimension-scale conventions (e.g., attributes indicating that a dataset is a dimension scale, whether it represents only a netCDF4 dimension or also serves as a coordinate variable, and its associated name).

Strict consistency is required: while generic HDF5 allows arbitrary dataset shapes, a valid netCDF4 file requires that variables align precisely with their dimension scales. Figure 4 provides a concrete instance of this mapping for the two netCDF4 dimensions in Figure 2. The dimensions *latitude* and *longitude* are realized as HDF5 datasets marked as dimension scales (green), and in this example each also serves as the corresponding 1D coordinate variable by storing the coordinate values directly. Each scale is annotated with netCDF4-specific attributes that make dimension identity and usage explicit. In particular, the *latitude* scale stores Dimension ID = 0 and Coordinate-variable dimension IDs = 0 and records Variable references indicating that *temperature*, *pressure*, and *relative humidity* attach this dimension on axis 0. The *longitude* scale is analogous, with Dimension ID = 1 and Coordinate-variable dimension IDs = 1 and Variable references indicating attachment on axis 1 for the same variables.

4.1.3 netCDF4 Variables → Annotated HDF5 Datasets. netCDF4 variables, representing the range values indexed by dimensions, are realized as standard HDF5 datasets annotated with metadata to reconstruct their dimensional semantics. This metadata includes:

- **Dimension references.** To link data values back to their domain, this attribute stores HDF5 object references pointing to the dimension scales that define each axis. This explicit back-reference allows the library to re-establish the association between array axes and their corresponding netCDF4 dimensions.
- **Variable dimension IDs.** This attribute records an ordered integer array of netCDF4 dimension identifiers corresponding to the dataset's axes. This sequence preserves the intended logical order of dimensions, ensuring the multi-dimensional structure is interpreted correctly by the application.

Figure 4 shows the variables *temperature*, *pressure*, and *relative humidity* as annotated HDF5 datasets (purple). Each maintains Dimension references to the *latitude* and *longitude* scales alongside

Variable dimension IDs (0, 1), ensuring the axis order and domain relationships are fully recoverable.

4.1.4 netCDF4 Attributes → HDF5 Attributes. Both netCDF4 and HDF5 support the ability to attach attributes to data objects, enabling the annotation of datasets and groups with metadata. In the netCDF4-to-HDF5 mapping, variable-level attributes in netCDF4 become attributes on the corresponding HDF5 datasets and group-level attributes in netCDF4 become attributes on the corresponding HDF5 groups. HDF5 attributes used to encode netCDF4-specific semantics are hidden from netCDF4 applications but remain visible to HDF5 applications.

The descriptive attributes shown in Figure 4 correspond directly to the descriptive attributes attached to the coordinate variables and variables in Figure 2. The remaining attributes in Figure 4, such as those encoding variable references, dimension identifiers, coordinate-variable dimension identifiers, and variable dimension identifiers, provide the information required to reconstruct netCDF4 semantics from the underlying HDF5 objects. These attributes are interpreted as internal metadata by the netCDF4 library and are therefore not exposed through the netCDF4 interface, even though they remain visible to HDF5 applications.

5 Implementation

To implement our unified data carving strategy, we leverage library interposition to intercept and augment calls to critical HDF5 and netCDF4 library functions. This approach allows us to add carving-specific functionality without modifying application source code. We identify key library functions for interposition, develop wrapper functions that encapsulate carving logic, and implement metadata management routines to maintain semantic consistency. Additionally, our implementation includes fallback mechanisms to enable dynamic data loading, ensuring reproducibility and correctness even when applications access data beyond the initially carved subset.

5.1 Library interposition

The HDF5 library parallels its data model by adopting an *object-oriented* approach in its function naming convention. Each function name begins with the prefix "H5", identifying the HDF5 library, followed by the object type it operates on, and ends with the specific function's purpose. For example, the function *H5Gopen* is used to open groups. The netCDF4 library employs a similar naming convention. Function names are prefixed with "nc", indicating the netCDF4 library, followed by a description of the function's purpose. For example, *nc_def_var* defines a new variable. Because applications access HDF5 and netCDF4 files through these libraries, it is possible to interpose at the function-call level. This interposition enables runtime monitoring and selective extraction of data, forming the basis for access-driven data carving.

Using *ltrace*, we analyzed the execution workflows of HDF5 and netCDF4 applications by tracing caller-callee relationships among functions within the HDF5 and netCDF4 libraries and applying heuristics aligned with the goal of producing a minimal yet complete carved file.

For instance, to identify files accessed by an application and to construct a minimal file structure, comprising groups and dataset objects but excluding dataset contents, we focused on functions that open files. Since we leverage HDF5 representation of netCDF4 files, constructing this structure is a prerequisite for carving both formats, as HDF5 organizes datasets hierarchically and identifies each dataset by a unique path. As a result, a dataset cannot be copied unless all its intermediate path components (i.e., parent groups) already exist in the target file. Copying dataset objects without their contents allows the carved file to retain empty placeholders that preserve the original HDF5 paths and indicate the presence of corresponding data in the source file. These placeholders additionally enable a fallback strategy in which data may be dynamically loaded from the original file during re-execution if needed. To identify the datasets accessed and copy them into the reconstructed hierarchy, we then tracked functions that read dataset contents. To enable a fallback strategy, where data objects may be dynamically loaded from alternative files, we examined functions that open data objects for subsequent reading, enabling redirection when needed. Finally, to ensure that attributes are carved only after all relevant objects, so as to avoid invalid references, we identified functions typically invoked at the end of application execution.

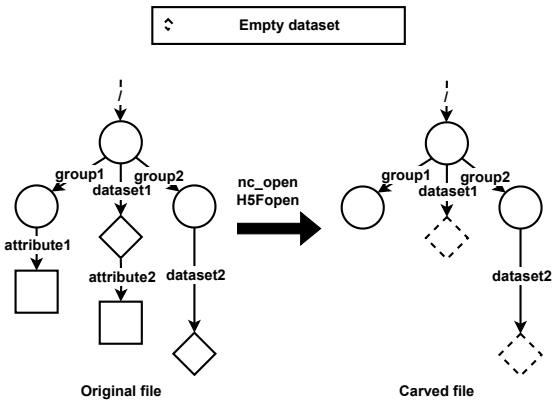
Through this methodical analysis, we identified five essential library calls that are sufficient for carving HDF5 and netCDF4 files, without redundancy or missing any necessary calls: *H5Fopen*, *nc_open*, *H5Dread*, *H5Ooopen*, and *H5_term_library*. For each of these calls, we wrap additional functionality around the original implementation using library interposition, as detailed below.

5.2 H5Fopen and nc_open to carve groups and empty datasets

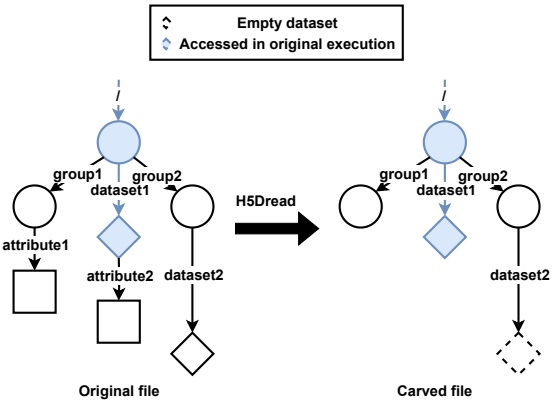
The *H5Fopen* function is used to open HDF5 and netCDF4 files. For netCDF4 applications, the *nc_open* function is called prior to *H5Fopen* to verify file semantics. For *H5Fopen*, the additionality functionality implemented includes performing a depth-first search on the directed graph structure of a file, treating both HDF5 and netCDF4 files as HDF5 files. Each HDF5 group is copied one by one into the carved file in the same location as the original file, preserving the original heirarchical structure. HDF5 datasets are also copied, but without their contents, as illustrated in Figure 5a. A small attribute is added to indicate that the copied dataset is empty and does not contain meaningful values. For netCDF4 applications, interposing on *nc_open* ensures that file paths are accurately relayed to subsequent *H5Fopen* calls during both the carving process and re-execution with the carved files.

5.3 H5Dread to carve datasets accessed

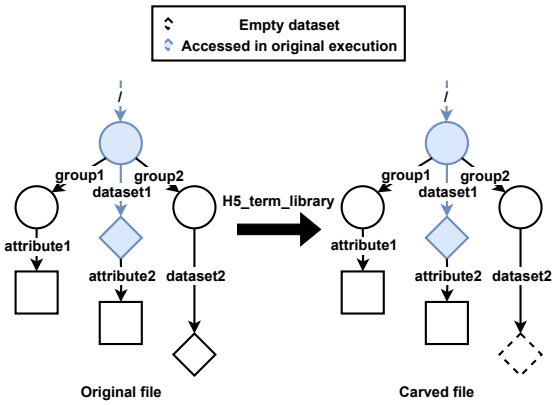
The *H5Dread* function interprets both HDF5 datasets and netCDF4 variables as HDF5 datasets and is used to read their contents. As each *H5Dread* call is made, the corresponding empty copy of the HDF5 dataset that is accessed is replaced by a copy of the HDF5 dataset complete with the data contents, as shown in Figure 5b. A dataset that is not read but is present in the original file will not be copied if an *H5Dread* call is not made on it. This results in a carved file containing only the subset of data that was accessed, at the granularity of HDF5 dataset objects.



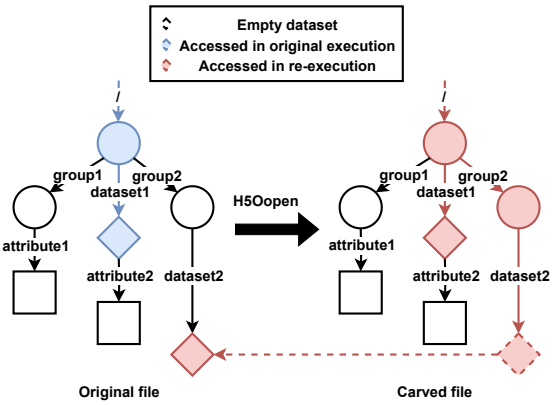
(a) Interposition on *H5Fopen* and *nc_open* to construct a minimal file structure.



(b) Interposition on *H5Dread* to selectively copy dataset objects based on read operations.



(c) Interposition on *H5_term_library* to perform the final metadata carving pass.



(d) Interposition on *H5Oopen* to enable fallback during re-execution to ensure correctness.

Figure 5: Overview of the library interposition points used to manage the carving lifecycle.

5.4 H5_term_library to carve attributes

The *H5_term_library* call is invoked during library shutdown, at the conclusion of an application's execution. Since attributes may contain references to other objects, we defer attribute carving to a final pass, after all groups and datasets have been copied and a complete mapping from source objects to their carved counterparts is available. By the time *H5_term_library* is invoked, this condition is met. The additional functionality implemented copies all associated attributes for the groups and datasets present in the carved file, including those not accessed, as shown in Figure 5c.

Within this pass, **Reference** attributes require specific intervention; a direct copy would retain invalid pointers to the original file, potentially causing segmentation faults, as shown in Figure 6a. Instead, the system dereferences the target object in the source and generates a new reference pointing to the corresponding object in the carved file, as shown in Figure 6b.

Composite datatypes, specifically Compound, Array, and Variable-Length, require recursive handling to preserve nesting structure and layout. For Compound attributes, the system traverses members (including nested composite types) and reconstructs an equivalent in-memory representation consistent with the datatype's declared member ordering and offsets for each element in the attribute. Array attributes are processed by identifying the base datatype and extents, then allocating and populating a representation that preserves the declared dimensionality. Variable-Length (VL) attributes are represented as sequences of (length, data-pointer) pairs; the system iterates over these pairs, interprets each payload according to the attribute's base datatype, and copies the resulting values into newly allocated storage. Throughout this process, any embedded references encountered within composite values are detected and rewritten using the same reference-copying logic as standalone reference attributes.

Finally, **Atomic** attributes (e.g., integers, floating-point values) represent the simplest carving case as they possess no internal substructure to interpret. Processing these attributes reduces to reading the raw value into a buffer and writing it verbatim to the carved file while preserving the datatype description.

This attribute pass completes carving at the metadata level. For HDF5, attributes are essential for self-description; omitting them yields a file with valid datasets but missing context. For netCDF4,

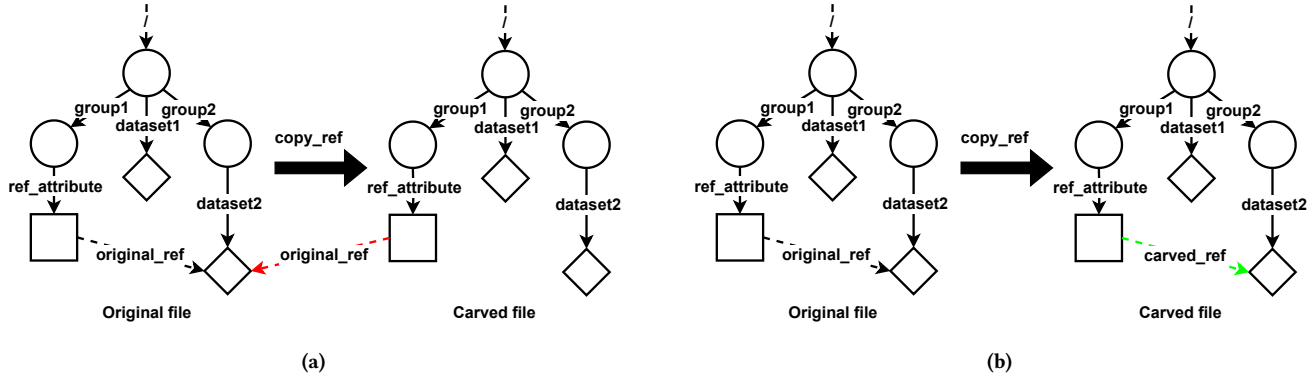


Figure 6: Comparison of reference attribute reconstruction. (a) A direct copy results in an invalid pointer to the original file. (b) Dereferencing and re-mapping ensures the reference points to the correct object in the carved file.

which encodes key elements of its data model in HDF5 attributes, this pass is critical for preserving structural integrity. Crucially, this pass handles the complex HDF5 datatypes used to enforce netCDF4 structural constraints. For netCDF4 dimensions stored as HDF5 dimension scales, it carves the variable references—implemented as a compound datatype containing nested HDF5 object references and axis indices—alongside dimension IDs, represented as scalar integers, and coordinate-variable dimension IDs, represented as integer arrays. For netCDF4 variables, the pass preserves dimension references, implemented as a variable-length datatype containing a sequence of HDF5 object references that link variables back to their defining dimension scales, alongside variable dimension IDs represented as integer arrays. Without correct reconstruction of these nested composite types and identifier metadata, the carved output reduces to disconnected HDF5 datasets that netCDF4 applications cannot interpret as a valid netCDF4 file.

5.5 *H5Oopen* for fallback access during re-execution

During re-execution, an application may request datasets that were not accessed in the original run and therefore were not included in the carved file. To preserve correctness, the system interposes on *H5Oopen*, which is commonly invoked to obtain an object identifier prior to issuing a read on that object. Interposition to this function is enabled only in re-execution mode.

The HDF5 Virtual File Driver (VFD) layer provides multiple backends for accessing content, such as the default *SEC2* driver for local files [34] and the *ROS3* driver for objects hosted on Amazon S3 [30]. The system leverages this abstraction to implement transparent fallback: upon each *H5Oopen*, it determines whether the target object (in particular, a dataset) exists in the carved file. If present, the call proceeds normally. If absent, the open is redirected to a designated fallback file, allowing the subsequent read to be satisfied from the original (or complete) dataset source, as shown in Figure 5d. This design maintains functional re-execution across a broader range of execution conditions.

Table 1: Scientific applications used for evaluation.

Application	Data Format	GitHub Repository
POMD-PF	HDF5	[23]
Feature Database	HDF5	[27]
Kriging OMI Level 2	HDF5	[3]
Kriging VIIRS	netCDF4	[4]
Tree Mortality	netCDF4	[8]
Kelp Forest Projection	netCDF4	[7]

6 Experiments

In this section, we detail the comprehensive experiments conducted to evaluate the efficacy of SemCarve in discerning data bloat within programs. The experimentation encompassed scientific workloads from NASA. All tests were executed on a Ubuntu 22.04 machine boasting 96 cores, 1.5TB of memory, and an AMD EPYC 7451 24-Core Processor, ensuring a standardized computing environment.

All programs involved in the experiments utilized HDF5 and netCDF4 data files. SemCarve is implemented in the C programming language to align with the HDF5 and netCDF4 libraries, which are also written in C. Applications developed in other languages rely on interfaces to these C libraries. To intercept library calls and execute the carving framework's functionality, the system utilizes the *LD_PRELOAD* mechanism.

6.1 Scientific Applications

To provide a robust assessment of SemCarve's capabilities, we subjected it to six scientific workloads sourced from various domains. Table 1 lists these applications, along with their data formats and references to their respective GitHub repositories. All six applications were implemented in the Python programming language.

6.2 Results

In our experiments, we measure the storage reduction because it is the main objective of the carving framework. In addition, we measure I/O characteristics of the application execution with the carving framework enabled to ensure that the carving framework does not incur excessive overhead costs.

Table 2: Comprehensive evaluation of the carving framework. We report the storage reduction achieved and the runtime overhead incurred during the initial carving phase ("Exec.") versus the reproducible re-execution phase ("Re-exec.").

	Storage Footprint			Runtime Performance		
Application	**Original**	**Carved**	**Red.**	**Original**	**Exec.**	**Re-exec.**
POMD-PF	862 MB	475 MB	45%	64 s	64 s	61 s
Feature Database	18 GB	4.4 GB	76%	2731 min	3017 min	2856 min
Kriging OMI Level 2	579 MB	34 MB	94%	5 s	212 s	8 s
Kriging VIIRS	75 GB	10 GB	87%	2710 min	2781 min	2633 min
Tree Mortality	10.3 GB	6.7 GB	32%	449 s	506 s	491 s
Kelp Forest Projection	7.8 GB	4.3 GB	45%	437 min	460 min	406 min

Table 2 shows storage size of the original files, the storage size of the carved files and the storage reduction percentage achieved with the carving framework. Figure 7a shows the original size and carved size of the files in log scale. The POMD-PF application shows a reduction from 862 MB to 475 MB, achieving a 45% size reduction. The Feature Database application, originally 18 GB, was carved down to 4.4 GB, resulting in a significant 76% reduction. The Kriging OMI Level 2 application experienced the highest reduction percentage, shrinking from 579 MB to just 34 MB, achieving a 94% reduction. Similarly, the Kriging VIIRS application was reduced by 87%, from 75 GB to 10 GB. The Tree Mortality application underwent a more modest 32% reduction, from 10.3 GB to 6.7 GB, while the Kelp Forest Projection application saw a 45% reduction, decreasing from 7.8 GB to 4.3 GB.

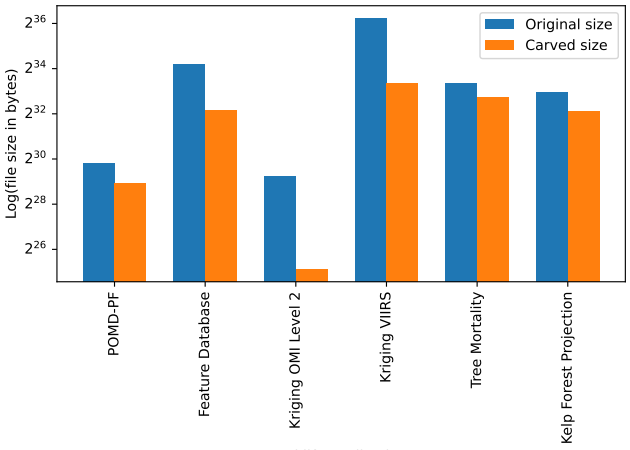
Table 2 also presents the runtime for each application and Figure 7b presents the runtime in log scale, under three configurations:

- **Original:** The application runs without the carving framework.
- **Execution Mode:** The carving framework is active, intercepting library calls and creating carved files.
- **Re-execution Mode:** The application runs against the carved files.

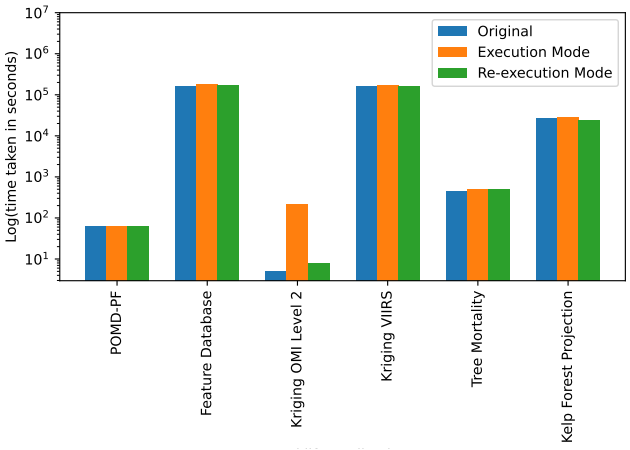
In the execution phase, the carving framework incurs additional I/O overhead because it creates files and populates the necessary datasets as the application accesses them. In the re-execution phase, overhead is primarily due to the framework's bookkeeping and decision logic for resolving each access against the carved files. The extent of this overhead depends on the total number and size of files, and whether files are reopened multiple times throughout the workflow. In certain cases, re-execution may exhibit lower runtimes compared to the original execution, owing to reduced I/O overhead when accessing a minimized subset of data.

Execution vs. Original For all applications, the runtime for execution mode is higher than the original execution, reflecting the overhead of creating file hierarchies and metadata, and copying data objects as they are accessed. A notable jump is seen for Kriging OMI Level 2 (from 5s to 212s), because the application's baseline runtime is very low and it processes a large number of files which amplifies the runtime for execution mode.

Re-execution vs. Execution Re-execution mode is faster than execution mode because the needed data objects have already been carved and stored locally.



(a) File size comparison (log scale)



(b) Execution time comparison (log scale)

Figure 7: Evaluation of the carving framework. (a) Comparison of original and carved file sizes in bytes. (b) Comparison of original, execution, and re-execution times.

Re-execution vs. Original While some re-execution times are slightly higher or lower than the original, the differences are generally modest. Re-execution can be marginally slower due to the additional bookkeeping required to validate and resolve accesses against the carved files. Conversely, re-execution can be marginally

faster when the carved files reduce I/O by excluding unused data and enabling more efficient reads over the accessed subsets.

These results demonstrate that SemCarve significantly reduces storage overhead by isolating the specific subsets of data accessed by scientific workflows without incurring excessive overhead costs. To ensure reproducibility, each application was re-executed using the carved files and the output was validated to match that of the original execution, confirming that the extracted subsets retained the necessary data for execution. Although an application can theoretically access all of its available data, our findings indicate that real-world applications often utilize only a subset of the available data. Consequently, storing complete datasets in their entirety introduces unnecessary storage and distribution overhead, which can be mitigated by leveraging data carving techniques to extract and store only the relevant data.

7 Conclusion

In this paper, we demonstrated that a unified data carving architecture is achievable for distinct file formats when a principled mapping exists between their underlying data models. Focusing on the structural correspondence between HDF5 and netCDF4, we introduced a framework that leverages this shared abstraction to transparently extract only the data objects accessed during execution. This ensures that only the data subsets actually needed by the application are retained, at the granularity of format-specific data objects. This selective data inclusion reduces container footprints and improves deployment times while preserving the application's reproducibility. Our experimental evaluations demonstrate substantial storage savings with moderate I/O overhead when creating carved files and minimal I/O overhead when reusing them. By aligning carving logic with the shared data model, our work offers a more generalized solution to container bloat, accelerating scientific workflows and facilitating efficient reproducibility.

Acknowledgments

This material is based upon work supported by the National Aeronautics and Space Administration (NASA) under Grant AIST-21-0095. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NASA.

References

[1] Raffay Atiq, Ashish Gehani, Tanu Malik, and Fareed Zaffar. 2025. SCALPEL: Structured Content Access Logging and Pruning for Efficient Layouts. *21st IEEE International Conference on e-Science* (2025). https://www.csl.sri.com/users/gehani/papers/eScience-2025.SCALPEL.pdf

[2] Nikilesh Balakrishnan, Thomas Bytheway, Ripduman Sohan, and Andy Hopper. 2013. OPUS: A Lightweight System for Observational Provenance in User Space. In *5th USENIX Workshop on the Theory and Practice of Provenance (TaPP 13)*.

[3] cniddodi. 2021. *Kriging OMI Level 2.* https://bit.ly/41m4hnG

[4] cniddodi. 2021. *Kriging VIIRS.* https://bit.ly/3OK05Xt

[5] data-carving. 2025. *Data Carving Dockerfiles.* https://github.com/data-carving/SemCarve-Dockerfiles

[6] data-carving. 2025. *HDF5/netCDF4 Data Carving Framework.* https://github.com/data-carving/SemCarve-HDF5-netCDF4-Data-Carving-Tool

[7] EcoPro-Systems. 2024. *Kelp Forest Projection.* https://bit.ly/4iq8Q6t

[8] EcoPro-Systems. 2025. *Tree Mortality.* https://bit.ly/3OGWdXl

[9] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. 2011. An overview of the HDF5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 workshop on array databases*. 36–47.

[10] Ashish Gehani and Dawood Tariq. 2012. SPADE: Support for Provenance Auditing in Distributed Environments. *13th ACM/IFIP/USENIX International Middleware Conference* (2012). http://www.csl.sri.com/users/gehani/papers/MW-2012.SPADE.pdf

[11] Tyler Harter, Brandon Salmon, Rose Liu, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. 2016. Slacker: Fast distribution with lazy docker containers. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*. 181–195.

[12] Y. JIANG, Qinkun Bao, Shuai Wang, Xiao Liu, and Dinghao Wu. 2018. RedDroid: Android Application Redundancy Customization Based on Static Analysis.. In *IEEE International Symposium on Software Reliability Engineering*.

[13] Y Jiang, D Wu, and P Liu. 2016. JRed: Program Customization and Bloatware Mitigation Based on Static Analysis.. In *Proceedings of the IEEE 40th Annual Computer Software and Applications Conference*.

[14] Gregory Malecha, Ashish Gehani, and Natarajan Shankar. 2015. Automated Software Winnowing. *30th ACM Symposium on Applied Computing (SAC)* (2015). http://www.csl.sri.com/users/gehani/papers/SAC-2015.Winnow.pdf

[15] Raveendra Kumar Medicherla, Raghavan Komondoor, and S. Narendran. 2015. Program specialization and verification using file format specifications. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 191–200. doi:10.1109/ICSM.2015.7332465

[16] Chaitra Niddodi, Ashish Gehani, Tanu Malik, Sibin Mohan, and Michael Rilee. 2023. IOSPReD: I/O Specialized Packaging of Reduced Datasets and Data-Intensive Applications for Efficient Reproducibility. *Access* 11 (2023). http://www.csl.sri.com/users/gehani/papers/Access-2023.IOSPReD.pdf

[17] NSF Unidata. 2012. *HDF5 Dimension Scales.* https://www.unidata.ucar.edu/blogs/developer/en/entry/dimensions_scales

[18] NSF Unidata. 2023. *netCDF4 Attributes.* https://docs.unidata.ucar.edu/netcdf-c/4.9.2/group__attributes.html

[19] NSF Unidata. 2023. *netCDF4 Dimensions.* https://docs.unidata.ucar.edu/netcdf-c/4.9.2/group__dimensions.html#details

[20] NSF Unidata. 2023. *netCDF4 Groups.* https://docs.unidata.ucar.edu/netcdf-c/4.9.2/group__groups.html#details

[21] NSF Unidata. 2023. *netCDF4 Variables.* https://docs.unidata.ucar.edu/netcdf-c/4.9.2/group__variables.html

[22] NSF Unidata. 2025. *netCDF File Format Specifications.* https://docs.unidata.ucar.edu/netcdf-c/4.9.3/file_format_specifications.html

[23] radiant-systems-lab. 2023. *POMD-PF.* bit.ly/3TrxwkO

[24] R Rew, E Hartnett, J Caron, et al. 2006. NetCDF-4: Software implementing an enhanced data model for the geosciences. In *22nd International Conference on Interactive Information Processing Systems for Meteorology, Oceanograph, and Hydrology*, Vol. 6.

[25] Hashim Sharif, Muhammad Abubakar, Ashish Gehani, and Fareed Zaffar. 2018. Trimmer: Application Specialization for Code Debloating. *33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)* (2018). http://www.csl.sri.com/users/gehani/papers/ASE-2018.Trimmer.pdf

[26] Christopher Smowton. 2014. *I/O optimisation and elimination via partial evaluation.* Ph. D. Dissertation. University of Cambridge.

[27] SpatioTemporal. 2022. *Feature Database.* https://bit.ly/41h1BI6

[28] Manolis Stamatogiannakis, Paul Groth, and Herbert Bos. 2014. Looking inside the black-box: capturing data provenance using dynamic instrumentation. In *IPAW*. Springer, 155–167.

[29] Manolis Stamatogiannakis, Hasanat Kazmi, Hashim Sharif, Remco Vermeulen, Ashish Gehani, Herbert Bos, and Paul Groth. 2016. Tradeoffs in Automatic Provenance Capture. *Provenance and Annotation of Data and Processes* 9672 (2016). http://www.csl.sri.com/users/gehani/papers/IPAW-2016.Instrumentation.pdf

[30] The HDF Group. 2022. *Cloud Storage Options for HDF5.* www.hdfgroup.org/2022/08/cloud-storage-options-for-hdf5/

[31] The HDF Group. 2024. *HDF5 Attributes.* https://docs.hdfgroup.org/releases/hdf5/v1_14/v1_14_5/documentation/doxygen/_h5_a__u_g.html

[32] The HDF Group. 2024. *HDF5 Datasets.* https://docs.hdfgroup.org/releases/hdf5/v1_14/v1_14_5/documentation/doxygen/_h5_d__u_g.html

[33] The HDF Group. 2024. *HDF5 Groups.* https://docs.hdfgroup.org/releases/hdf5/v1_14/v1_14_5/documentation/doxygen/_h5_g__u_g.html

[34] The HDF Group. 2026. *HDF5 Virtual File Layer.* https://support.hdfgroup.org/documentation/hdf5/latest/_v_f_l_t_n.html

[35] Rohan Tikmany, Aniket Modi, Raffay Atiq, Moaz Reyad, Ashish Gehani, and Tanu Malik. 2024. Access-Based Carving of Data for Efficient Reproducibility of Containers. *24th ACM/IEEE Symposium on Cluster, Cloud, and Internet Computing* (2024). http://www.csl.sri.com/users/gehani/papers/CCIC-2024.ABCD.pdf

[36] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at Google with Borg. In *Tenth European Conference on Computer Systems*.

[37] Qi Xin, Qirun Zhang, and Alessandro Orso. 2022. Studying and Understanding the Tradeoffs Between Generality and Reduction in Software Debloating.. In *The 37th IEEE/ACM International Conference on Automated Software Engineering (ASE) 2022*.