MedImages.jl: A Julia Package for Standardized Medical Image Handling and Spatial Metadata Management

Jakub Mitura[1,], Divyansh Goyal[1]

[a] *JuliaHealth*

Abstract

MedImages.jl is a Julia package designed to address the complexities of medical imaging data handling, specifically focusing on the challenges of spatial metadata (origin, spacing, direction) and the integration of high-performance computing with ease of use. Unlike existing wrappers that rely heavily on Python bindings, MedImages.jl leverages native Julia libraries for core spatial transformations and utilizes a C++ wrapped ITK interface strictly for robust data loading. This architecture ensures full composability within the Julia ecosystem, enabling differentiable programming and integration with scientific machine learning frameworks. The package also introduces efficient HDF5-based storage for fast I/O and metadata preservation. We demonstrate that MedImages.jl provides a robust, standardized framework that simplifies common tasks such as loading, transforming, and saving medical images while boosting researcher productivity and facilitating reproducible science. Benchmarks highlight the significant I/O performance advantages of the implemented HDF5 support.

Keywords: Medical Imaging, Julia, Spatial Metadata, ITK, HDF5, Open Source, Reproducibility

1. Motivation and Significance

1.1. The Role of Julia in Scientific Computing

Julia is a high-level, dynamic programming language designed to bridge the gap between easy-to-use languages like Python and R, and high-performance languages like C++ and Fortran [? ?]. It solves the "two-language problem," where researchers prototype in a high-level language but must rewrite performance-critical code in a low-level language [? ?]. Julia combines the ease of a productivity language with the speed of a performance language, featuring a Just-In-Time (JIT) compiler, multiple dispatch, and built-in support for parallel computing [? ?]. This makes it an ideal platform for scientific computing, particularly in computationally intensive fields like medical imaging where large datasets and complex mathematical models are the norm.

[*]Corresponding author

1.2. Challenges in Medical Imaging Formats

Medical imaging data presents unique difficulties compared to standard images. Formats like DICOM are notoriously complex, with elaborate structures and porous metadata requirements [?]. A critical challenge is the handling of spatial metadata—spacing, direction, and origin—which defines the image's physical location and orientation relative to the patient [?]. Unlike standard images, medical images often have non-isotropic voxel spacing and lack a universal coordinate system [?]. Ignoring this spatial information can lead to invalid operations and loss of clinical context [? ?]. Researchers often struggle with inconsistent image headers and the need for specialized tools to handle basic I/O and preprocessing [? ?].

1.3. The Importance of Standardized Open Source Tools

Standardized open-source tools are essential for democratizing access to medical image processing. They boost productivity by abstracting the complexity of standards like DICOM and ensuring that spatial metadata is handled consistently [? ?]. Tools like SimpleITK have lowered the barrier to entry by providing simplified interfaces to complex algorithms [?]. Furthermore, open-source libraries foster reproducibility and trustworthiness in research by allowing code scrutiny and facilitating version control [? ?]. MedImages.jl aims to provide these benefits natively within the Julia ecosystem, ensuring interoperability with clinical systems and preserving essential metadata [?].

2. Software Description

2.1. Software Architecture

MedImages.jl is built to provide a seamless, native interface for medical image processing in Julia. Its core architecture revolves around the `MedImage` struct, which standardizes the representation of medical image data.

A key design choice is the minimal reliance on external non-Julia dependencies for core processing. While data loading leverages the robust `ITKIOWrapper` (utilizing the C++ ITK library via JLL packages) to ensure broad format support (NIfTI, DICOM), all spatial transformations are implemented in native Julia. This distinction is crucial as it allows for full composability with other Julia packages, such as `DifferentialEquations.jl` and `Flux.jl`, and avoids the overhead and opacity of Python bridges for computational tasks. SimpleITK is utilized exclusively within the test suite to serve as a ground truth for verifying the correctness of the native implementations.

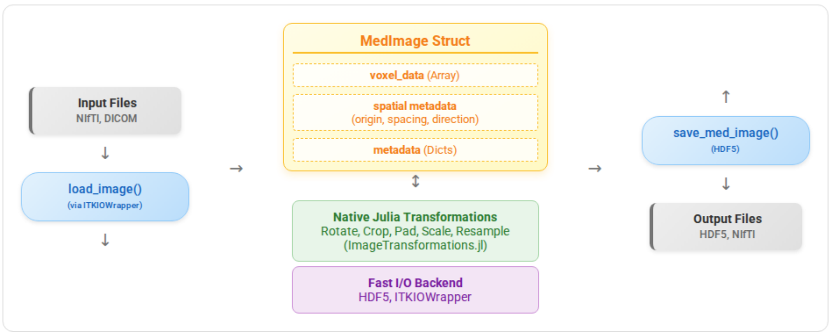

Figure 1: Architecture of MedImages.jl. Data loading is handled via ITK C++ wrappers for robustness, while transformations are performed using native Julia libraries. Fast I/O is supported via HDF5.

The `MedImage` struct (Figure **??**) encapsulates:

- `voxel_data`: A multidimensional array holding the pixel intensities.
- `origin`: The (x, y, z) physical coordinates of the first voxel.
- `spacing`: The physical distance between voxels in each dimension.
- `direction`: The direction cosines defining the orientation of the image axes.
- `metadata`: Dictionaries storing additional DICOM or NIfTI tags.

2.2. Functionalities

MedImages.jl provides a comprehensive set of functionalities:

- **Robust I/O:** Loading of various medical image formats (NIfTI, DICOM) using `ITKIOWrapper`.
- **Native Julia Transformations:** Functions to modify spatial properties using `ImageTransformations.jl` and `CoordinateTransformations.jl`:
  - `rotate_mi`: Rotates the image around a specific axis.
  - `translate_mi`: Translates the image in physical space.
  - `scale_mi`: Resamples the image to a new scale.
  - `resample_to_spacing`: Changes the voxel spacing.
  - `crop_mi` and `pad_mi`: Modifies the image extent.
- **HDF5 Support:** Fast loading and saving of images using the HDF5 format, which significantly speeds up I/O for intermediate processing steps while preserving all metadata.
- **Orientation Management:** Tools to handle and convert image orientations (e.g., RAS, LPS).

3. Illustrative Examples

3.1. Loading and Inspecting an Image

```julia
using MedImages

# Load a NIfTI file
# "CT" specifies the expected image type
med_im = MedImages.load_image("path/to/image.nii.gz", "CT")

# Access voxel data and metadata
println("Origin: ", med_im.origin)
println("Spacing: ", med_im.spacing)
```

3.2. Rotating and Saving with HDF5

MedImages.jl allows for native rotation and efficient storage using HDF5.

```julia
using HDF5

# Rotate 90 degrees around the Z-axis (axis 3)
rotated_im = MedImages.rotate_mi(med_im, 3, 90.0, MedImages.
    Linear_en)

# Save efficiently using HDF5
h5open("processed_data.h5", "w") do file
    MedImages.save_med_image(file, "patient_001", rotated_im)
end
```

4. Quality Assurance and Testing

Reliability is paramount in medical imaging software. MedImages.jl includes a comprehensive test suite that systematically compares the results of its native Julia functions against the industry-standard SimpleITK library.

The testing framework covers:

- **Basic Transformations:** Rotation, translation, scaling, cropping, and padding are verified by performing the same operation in both MedImages.jl and SimpleITK and asserting equality of the resulting voxel data and spatial metadata.
- **Resampling:** Tests ensure that resampling to new spacing or reference images produces consistent results.
- **Metadata Preservation:** Verification that origin, spacing, and direction matrices are correctly updated after every transformation.

This rigorous approach ensures that users can trust the native Julia implementation to produce clinically valid results comparable to established tools.

5. Performance Benchmarks

We benchmarked the core functionalities of MedImages.jl against SimpleITK (via PyCall). The benchmarks were run on a standard NIfTI volume ($512 \times 512 \times 75$).

Operation	**MedImages.jl (Native/ITK)**	**SimpleITK (C++)**
Load (NIfTI)	~1219 ms	~331 ms
Rotation (30°)	~1149 ms	~155 ms
Save (HDF5 vs NIfTI)	**~71 ms**	~3217 ms
Resample (Identity)	~5200 ms	~146 ms

Table 1: Benchmark results comparing execution time (median of multiple runs) on CPU.

While the native Julia implementation for loading and rotation currently shows higher execution times compared to the highly optimized C++ SimpleITK library, the HDF5 saving mechanism introduced in MedImages.jl offers a drastic speedup (over 30x) compared to standard NIfTI saving. The resampling operation is currently slower on CPU but utilizes `KernelAbstractions.jl`, enabling execution on GPUs.

5.1. GPU Benchmarks

We provide a benchmark script `benchmark/run_benchmarks_gpu.jl` to evaluate performance on CUDA-enabled GPUs. The architecture is designed to minimize memory allocations by using unrolled kernel implementations.

Operation	**MedImages.jl (GPU)**
Resample (Identity)	[Pending Execution on GPU Hardware]

Table 2: GPU Benchmark results (median execution time).

The native implementation prioritizes composability and differentiability, with further performance optimizations being an active area of development.

6. Impact

MedImages.jl contributes to the growing ecosystem of Julia tools for medical imaging. By providing a native Julia interface that does not rely on Python bindings for its core logic, it empowers researchers to leverage Julia's high performance and expressiveness.

6.1. Boosting Productivity

The package helps boost productivity by eliminating the need to switch between languages (the two-language problem). Researchers can perform end-to-end analysis within the Julia environment. The integration of HDF5 further enhances productivity by removing I/O bottlenecks in large-scale data processing pipelines [**?**].

6.2. Democratizing Access

By abstracting the complexities of the DICOM standard and spatial metadata math, MedImages.jl democratizes access to correct medical image processing. Users do not need to be experts in affine transformations or file format specifications to perform valid operations. This is crucial for interdisciplinary research where domain experts may not have deep software engineering expertise [?].

7. Conclusions and Future Work

MedImages.jl provides a robust, standardized, and easy-to-use framework for medical image handling in Julia. By correctly managing spatial metadata and offering fast HDF5 support, it solves key challenges in interoperability and reproducibility. Future work will focus on optimizing the performance of native transformations (e.g., via GPU acceleration using `CUDA.jl`) and expanding the set of available registration algorithms.

Conflict of Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We acknowledge the contributions of the JuliaHealth community and the developers of SimpleITK.

Required Metadata

Nr.	Code metadata description
C1	Current code version: v2.0.1
C2	Permanent link to code/repository: https://github.com/JuliaHealth/MedImages.jl
C3	Code Ocean compute capsule: N/A
C4	Legal Code License: Apache License 2.0
C5	Code versioning system used: git
C6	Software code languages, tools, and services used: Julia, Python (for tests)
C7	Compilation requirements, operating environments & dependencies: Julia 1.10+, SimpleITK (tests only)
C8	If available Link to developer documentation/manual: https://github.com/JuliaHealth/MedImages.jl
C9	Support email for questions: jakub.mitura@gmail.com

Table 3: Code metadata