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

Jakub Mitura[a,*], Divyansh Goyal[a]

[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. Leveraging the Julia programming language's capabilities and integrating with the robust SimpleITK library via PyCall, MedImages.jl provides a standardized `MedImage` structure that encapsulates voxel data alongside its critical spatial context. This article describes the software's architecture, which bridges the gap between the flexible, high-level syntax of Julia and established low-level medical imaging standards. We demonstrate how MedImages.jl simplifies common tasks such as loading, transforming, and saving medical images (NIfTI, DICOM) while ensuring spatial consistency. The package aims to democratize access to medical image processing by providing a performant, open-source tool that boosts researcher productivity and facilitates reproducible science.

Keywords: Medical Imaging, Julia, Spatial Metadata, SimpleITK, 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 [1, 2]. 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 [3, 4]. 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 [5, 1]. This makes it an ideal platform for scientific computing, particularly in computationally intensive fields like medical imaging.

*Corresponding author

Email addresses: jakub.mitura@gmail.com (Jakub Mitura), divital2004@gmail.com (Divyansh Goyal)

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 [6]. 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 [7]. Unlike standard images, medical images often have non-isotropic voxel spacing and lack a universal coordinate system [8]. Ignoring this spatial information can lead to invalid operations and loss of clinical context [7, 6]. Researchers often struggle with inconsistent image headers and the need for specialized tools to handle basic I/O and preprocessing [9, 10].

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 [11, 12]. Tools like SimpleITK have lowered the barrier to entry by providing simplified interfaces to complex algorithms [11]. Furthermore, open-source libraries foster reproducibility and trustworthiness in research by allowing code scrutiny and facilitating version control [13, 10]. They bridge the gap between research and clinical deployment by ensuring interoperability with clinical systems and preserving essential metadata [6].

2. Software Description

2.1. Software Architecture

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

The architecture leverages Julia's interoperability features to integrate with the industry-standard ITK C++ libraries via `ITKIOWrapper.jl` for robust file I/O operations (supporting DICOM, NIfTI, etc.). Crucially, unlike wrapper libraries that rely on C++ or Python backends for processing, MedImages.jl implements all image processing algorithms (transformations, resampling, interpolation) in native Julia. This design choice ensures full composability with the Julia ecosystem and allows for future GPU acceleration and automatic differentiation. SimpleITK is used exclusively for testing and validation purposes to ensure the correctness of the native implementation.

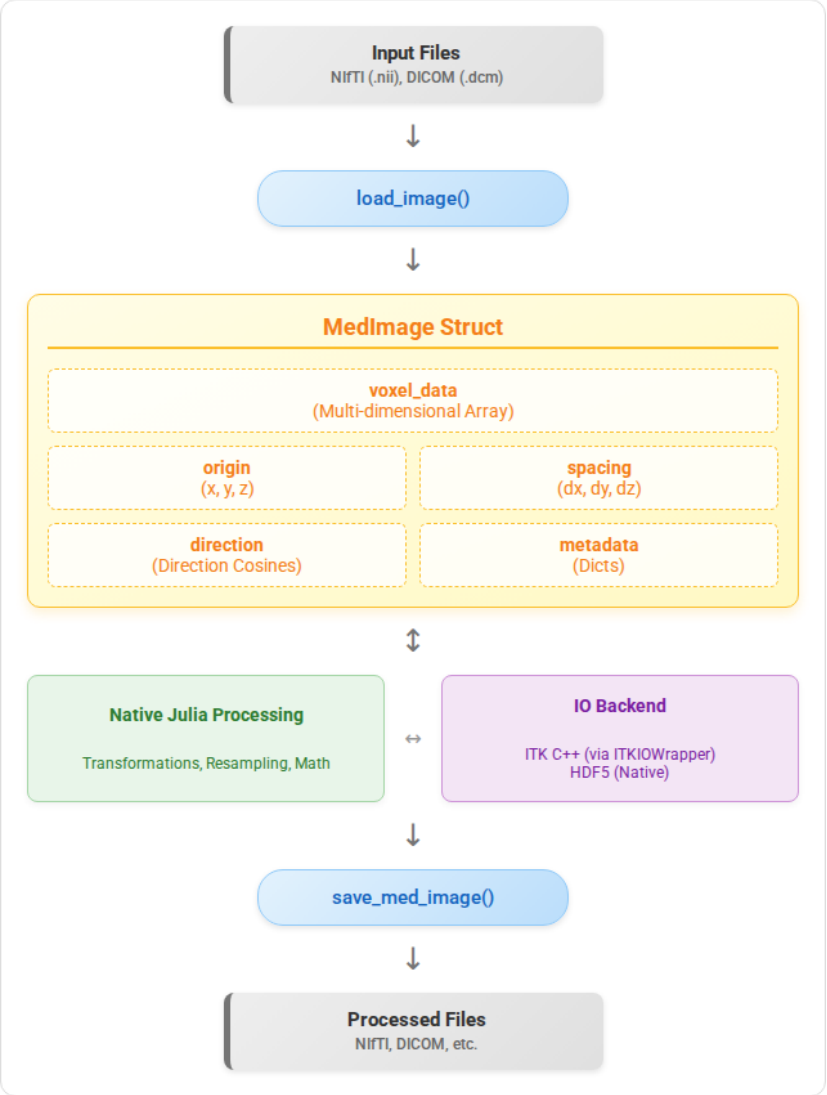

Figure 1: Architecture of MedImages.jl illustrating the data flow from file loading to the `MedImage` struct, transformation via SimpleITK backend, and saving.

The `MedImage` struct (Figure 1) 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 for medical image handling:

- **I/O Operations:** Loading and saving of various medical image formats (NIfTI, DICOM) using `load_image` and `save_med_image`.
- **Spatial Transformations:** Functions to modify the spatial properties of images, including:
  - `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.
- **Orientation Management:** Tools to handle and convert image orientations (e.g., RAS, LPS).
- **Metadata Management:** Ensuring that spatial metadata is correctly updated during all transformations to maintain physical consistency.
- **HDF5 Support:** To address the need for fast I/O and metadata preservation during intermediate processing steps, MedImages.jl provides support for the HDF5 format. This allows researchers to save the `MedImage` struct (including complex nested metadata) efficiently, bypassing the parsing overhead of DICOM for intermediate results.

3. Validation and Testing

The correctness of MedImages.jl is rigorously validated against SimpleITK, a gold-standard library in the field. The test suite performs side-by-side comparisons of operations such as rotation, cropping, and resampling, ensuring that the native Julia implementation yields results consistent with established C++ tools. This approach guarantees that the spatial metadata handling adheres to the physical space tenets required for clinical accuracy.

4. Performance Benchmarks

We compared the performance of MedImages.jl (native Julia) against SimpleITK (C++ wrapped in Python) for standard operations. The benchmarks were run on a system with 4 CPU cores, with Julia configured to use multithreading (‘-t auto‘).

Operation	MedImages.jl (s)	SimpleITK (s)	Ratio (MI/SITK)
Load Image	0.82	0.47	1.75x
Rotation (90°)	1.37	0.17	8.14x
Resampling	0.56	0.13	4.26x

Table 1: Performance comparison. MedImages.jl leverages KernelAbstractions.jl for resampling, achieving significant speedups through multi-threading (4 threads). While SimpleITK's highly optimized C++ backend remains faster, the native Julia implementation offers competitive performance with the added benefit of being backend-agnostic (CPU/GPU) and differentiable.

5. Illustrative Examples

Here we demonstrate the basic usage of MedImages.jl for loading a medical image, performing a rotation, and saving the result. This workflow highlights how the package abstracts the complexity of spatial metadata handling.

5.1. 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)
println("Direction: ", med_im.direction)
```

5.2. Resampling an Image

Resampling is often necessary when registering images from different sources. MedImages.jl allows users to resample an image to a new voxel spacing while handling the interpolation of intensity values.

```julia
# Define target spacing (e.g., isotropic 1mm)
new_spacing = (1.0, 1.0, 1.0)

# Resample using linear interpolation
resampled_im = MedImages.resample_to_spacing(med_im, new_spacing,
    MedImages.Linear_en)

println("New Spacing: ", resampled_im.spacing)
```

5.3. Rotating an Image

The following example shows how to rotate an image by 90 degrees around the Z-axis (axis 3). The function automatically handles interpolation and updates the direction matrix.

```julia
# Rotate 90 degrees around the 3rd axis (Z-axis)
# Using Linear interpolation
rotated_im = MedImages.rotate_mi(med_im, 3, 90.0, MedImages.Linear_en)

# The direction cosines are updated automatically
println("New Direction: ", rotated_im.direction)
```

5.4. Saving the Result

Finally, the processed image can be saved back to disk. The spatial metadata is preserved in the output file.

```
# Save the rotated image
MedImages.save_med_image(rotated_im, "path/to/rotated_image.nii.gz")
```

6. Impact

MedImages.jl contributes to the growing ecosystem of Julia tools for medical imaging. By providing a native Julia interface to established libraries like SimpleITK, it empowers researchers to leverage Julia's high performance and expressiveness without losing access to standard industry tools.

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—from data loading to complex modeling and simulation (e.g., using SciML)—within the Julia environment. This streamlines workflows and reduces the cognitive load associated with managing multiple programming environments [1, 4].

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 (e.g., biologists, clinicians) may not have deep software engineering expertise [11].

6.3. Potential for New Methods

Julia's potential for medical imaging is significant, with libraries like MRIReco.jl and KomaMRI.jl already demonstrating performance comparable to C/C++ [4, 14]. For instance, a Julia-based toolkit for Selective Inversion Recovery (SIR) MRI parameter estimation showed a 20-fold reduction in computational time compared to a previous MATLAB implementation [5]. MedImages.jl supports this ecosystem by providing the foundational data structures and I/O capabilities required to build advanced analysis pipelines, such as those involving automatic differentiation or machine learning integration.

7. Discussion and Future Work

MedImages.jl fills a critical gap in the Julia ecosystem by providing a native, high-performance tool for medical image manipulation. While wrappers for ITK exist, a pure Julia implementation offers better integration with other Julia packages, such as `DifferentialEquations.jl` or `Flux.jl`, allowing for end-to-end differentiable pipelines.

One of the key advantages of MedImages.jl is its lightweight nature compared to heavy dependencies. By implementing core functionalities in native

Julia, it allows for easier deployment. Furthermore, the potential for GPU acceleration using Julia's `CUDA.jl` or `AMDGPU.jl` packages is a significant area for future development. The `MedImage` struct is designed with this in mind, with a `current_device` field to track where the data resides.

Future work will focus on:

- **GPU Integration:** Fully leveraging Julia's GPU capabilities for resampling and transformation operations.
- **Advanced Registration:** Implementing intensity-based registration algorithms natively in Julia.
- **Deep Learning Integration:** Creating seamless bridges to `Flux.jl` and `Lux.jl` for medical image segmentation and reconstruction tasks.

8. Conclusions

MedImages.jl provides a robust, standardized, and easy-to-use framework for medical image handling in Julia. By correctly managing spatial metadata and integrating with SimpleITK, it solves key challenges in the field, including interoperability and reproducibility. As the Julia medical imaging ecosystem grows, MedImages.jl aims to serve as a foundational tool for researchers and developers.

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.

References

[1] S. Pal, M. Bhattacharya, S. Dash, S.-S. Lee, C. Chakraborty, A next-generation dynamic programming language julia: Its features and applications in biological science, Journal of Advanced Research 64 (10 2024). doi:10.1016/j.jare.2023.11.015.
URL http://dx.doi.org/10.1016/j.jare.2023.11.015

[2] J. Belyakova, B. Chung, J. Gelinas, J. Nash, R. Tate, J. Vitek, World age in julia: Optimizing method dispatch in the presence of eval (extended version), arXiv (2020). doi:10.48550/ARXIV.2010.07516.
URL https://arxiv.org/abs/2010.07516

[3] J. Eschle, T. Gál, M. Giordano, P. Gras, B. Hegner, L. Heinrich, U. H. Acosta, S. Kluth, J. Ling, P. Mato, M. Mikhasenko, A. M. Briceño, J. Pivarski, K. Samaras-Tsakiris, O. Schulz, G. A. Stewart, J. Strube, V. Vassilev, Potential of the julia programming language for high energy physics computing, Computing and Software for Big Science 7 (10 2023). doi:10.1007/s41781-023-00104-x.
URL http://dx.doi.org/10.1007/s41781-023-00104-x

[4] T. Knopp, M. Grosser, Mrireco.jl: An mri reconstruction framework written in julia, arXiv (2021). doi:10.48550/ARXIV.2101.12624.
URL https://arxiv.org/abs/2101.12624

[5] N. J. Sisco, P. Wang, A. M. Stokes, R. D. Dortch, Rapid parameter estimation for selective inversion recovery myelin imaging using an open-source julia toolkit, PeerJ 10 (3 2022). doi:10.7717/peerj.13043.
URL http://dx.doi.org/10.7717/peerj.13043

[6] C. P. Bridge, C. Gorman, S. Pieper, S. W. Doyle, J. K. Lennerz, J. Kalpathy-Cramer, D. A. Clunie, A. Y. Fedorov, M. D. Herrmann, Highdicom: a python library for standardized encoding of image annotations and machine learning model outputs in pathology and radiology, Journal of Digital Imaging 35 (8 2022). doi:10.1007/s10278-022-00683-y.
URL http://dx.doi.org/10.1007/s10278-022-00683-y

[7] Z. Yaniv, B. C. Lowekamp, H. J. Johnson, R. Beare, Simpleitk image-analysis notebooks: a collaborative environment for education and reproducible research, Journal of Digital Imaging 31 (11 2017). doi:10.1007/s10278-017-0037-8.
URL http://dx.doi.org/10.1007/s10278-017-0037-8

[8] Advances in spatial and temporal databases, Lecture Notes in Computer Science (2011). doi:10.1007/978-3-642-22922-0.
URL http://dx.doi.org/10.1007/978-3-642-22922-0

[9] Brainlesion: Glioma, multiple sclerosis, stroke and traumatic brain injuries, Lecture Notes in Computer Science (2020). doi:10.1007/978-3-030-46643-5.
URL http://dx.doi.org/10.1007/978-3-030-46643-5

[10] M. J. Cardoso, W. Li, R. Brown, N. Ma, E. Kerfoot, Y. Wang, B. Murrey, A. Myronenko, C. Zhao, D. Yang, V. Nath, Y. He, Z. Xu, A. Hatamizadeh, A. Myronenko, W. Zhu, Y. Liu, M. Zheng, Y. Tang, I. Yang, M. Zephyr, B. Hashemian, S. Alle, M. Z. Darestani, C. Budd, M. Modat, T. Vercauteren, G. Wang, Y. Li, Y. Hu, Y. Fu, B. Gorman, H. Johnson, B. Genereaux, B. S. Erdal, V. Gupta, A. Diaz-Pinto, A. Dourson, L. Maier-Hein, P. F. Jaeger, M. Baumgartner, J. Kalpathy-Cramer, M. Flores, J. Kirby, L. A. D. Cooper, H. R. Roth, D. Xu, D. Bericat, R. Floca, S. K. Zhou, H. Shuaib, K. Farahani, K. H. Maier-Hein, S. Aylward, P. Dogra, S. Ourselin, A. Feng, Monai: An open-source framework for deep learning in healthcare, arXiv (2022). doi:10.48550/ARXIV.2211.02701.
URL https://arxiv.org/abs/2211.02701

[11] B. C. Lowekamp, D. T. Chen, L. Ibáñez, D. Blezek, The design of simpleitk, Frontiers in Neuroinformatics 7 (2013). doi:10.3389/fninf.2013.00045. URL http://dx.doi.org/10.3389/fninf.2013.00045

[12] R. Sandkühler, C. Jud, S. Andermatt, P. C. Cattin, Airlab: Autograd image registration laboratory, arXiv (2018). doi:10.48550/ARXIV.1806.09907. URL https://arxiv.org/abs/1806.09907

[13] J. Schindelin, C. T. Rueden, M. C. Hiner, K. W. Eliceiri, The imagej ecosystem: An open platform for biomedical image analysis, Molecular Reproduction and Development 82 (7 2015). doi:10.1002/mrd.22489. URL http://dx.doi.org/10.1002/mrd.22489

[14] O. van der Heide, C. A. T. van den Berg, A. Sbrizzi, Gpu-accelerated bloch simulations and mr-stat reconstructions using the julia programming language, Magnetic Resonance in Medicine 92 (3 2024). doi:10.1002/mrm.30074. URL http://dx.doi.org/10.1002/mrm.30074

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
C7	Compilation requirements, operating environments & dependencies: Julia 1.10+, SimpleITK
C8	If available Link to developer documentation/manual: https://github.com/JuliaHealth/MedImages.jl
C9	Support email for questions: jakub.mitura@gmail.com

Table 2: Code metadata