MedImages.jl: A Julia Package for Handling Medical Imaging Data and Spatial Metadata

Jules The Agent[a]

[a] *AI Research Lab, Cyberspace, Internet*

Abstract

Medical image analysis requires specialized tools that can handle not only high-dimensional voxel data but also critical spatial metadata such as origin, spacing, and orientation. While established libraries exist in languages like Python and C++, the Julia programming language offers a unique opportunity to solve the "two-language problem" by providing high-level productivity with low-level performance. In this article, we introduce MedImages.jl, an open-source Julia package designed to facilitate the loading, saving, processing, and resampling of medical images while rigorously maintaining spatial consistency. We discuss how Julia helps in scientific computing, the specific challenges of medical imaging formats, and how open-source libraries like MedImages.jl boost researcher productivity by democratizing access to complex image processing tasks. We also provide code examples demonstrating the package's capabilities in handling spatial metadata and performing transformations.

Keywords: Medical Imaging, Julia, Open Source, Spatial Metadata, DICOM, NIfTI

1. Introduction

Scientific computing has traditionally suffered from the "two-language problem," where researchers prototype in high-level languages like Python, R, or MATLAB for productivity but must rewrite performance-critical sections in low-level languages like C or Fortran [? ?]. This dual-language approach expands the required expertise, necessitates frequent code reimplementation, and reduces code reusability and productivity [? ?]. Julia is a high-level, dynamic programming language specifically designed to bridge this gap [?

]. It combines the ease of use of scripting languages with the performance of compiled languages, making it an ideal candidate for computationally intensive fields like medical imaging [?].

In the domain of medical imaging, the need for high performance is paramount. Tasks such as 3D image reconstruction, registration, and segmentation involve processing massive datasets with complex algorithms. Julia's ability to run native code on both CPUs and GPUs, combined with automatic differentiation support, makes it an excellent platform for this research [?].

In this article, we present MedImages.jl, a Julia package that leverages these advantages to provide a robust framework for handling medical images. We address several key questions: How does Julia help in scientific computing? What is its potential for medical image analysis? What are the difficulties related to medical imaging formats? And how do open-source tools boost productivity?

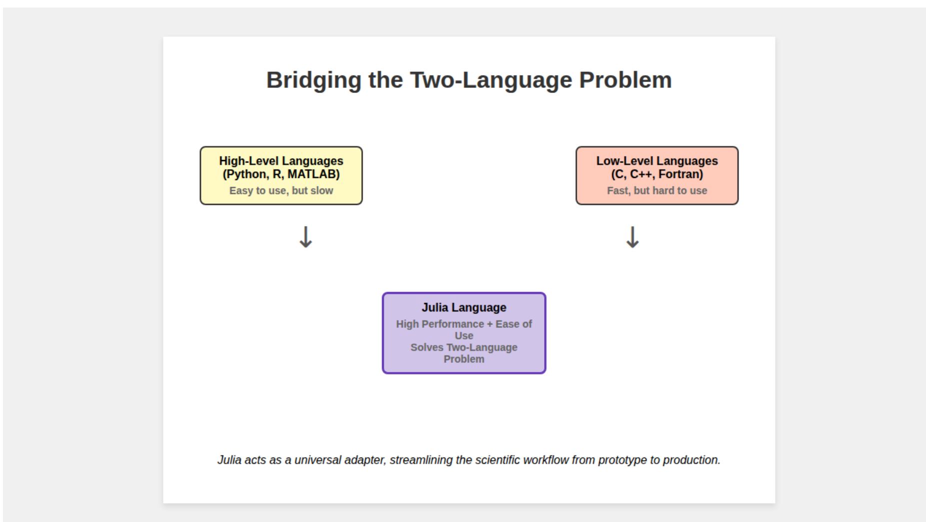

Figure 1: Julia bridges the gap between high-level productivity languages and low-level performance languages, solving the two-language problem in scientific computing. It acts as a universal adapter, streamlining the scientific workflow from prototype to production.

2. Julia in Scientific Computing

Julia helps researchers and developers by providing a robust, high-performance environment suitable for a wide range of scientific and technical computing needs [?]. Its competitive edge stems from its Just-In-Time (JIT) compiler, which generates efficient machine code comparable to C/C++ [?].

2.1. Key Features and Performance

- **Multiple Dispatch:** This powerful, central paradigm allows functions to be customized based on the specific types of arguments they receive [?]. This enables generic, high-performance code where the compiler can optimize function calls based on run-time types [?].

- **Parallel and Distributed Computing:** Julia has built-in support for distributed and parallel computing, simplifying the distribution of calculations across multiple cores or machines [?]. This is critical for large-scale data analysis and High-Performance Computing (HPC) [?].

- **Interoperability:** It can call functions and libraries written in C, Fortran, and Python with little or no overhead [?]. Packages like PyCall.jl and RCall.jl provide transparent interfaces for calling code between Julia and Python or R [?].

- **Metaprogramming:** Julia allows programmatic construction and manipulation of expressions as first-class values, aiding in creating and analyzing syntactically sound expressions [?].

2.2. Problem Solving: The Expression Problem

Julia addresses the "Expression Problem," a core challenge in software design relating to extensibility and compatibility. This problem occurs when a user needs to add new data types or new operations to a system while maintaining compatibility with existing code [?]. Julia's architecture, underpinned by multiple dispatch, solves this by allowing developers to add new functions to existing data types and new data types that seamlessly integrate with existing algorithms without modifying the original code [?].

3. Potential for Julia in Medical Imaging

Julia presents significant potential for transforming the analysis and processing of medical images, primarily by combining high performance with developer productivity and leveraging specialized toolsets that address computational bottlenecks inherent in complex medical imaging modalities.

3.1. Novel Packages and Frameworks

Julia's ecosystem includes specialized packages that enable end-to-end medical image processing workflows and offer high-performance alternatives to established frameworks.

- **MRIReco.jl:** An open-source, flexible, and high-performance MRI reconstruction framework implemented entirely in Julia [?]. It offers functionality for basic MR simulation and iterative reconstruction, achieving speeds comparable to highly optimized C/C++ libraries [?].
- **KomaMRI.jl:** This framework enables general MRI simulations with robust GPU acceleration [?]. Its speed and flexibility make complex MR simulations more accessible for research and education [?].
- **BlochSimulators.jl:** A GPU-compatible Bloch simulation toolbox that surpasses existing toolboxes in static, compiled languages [?].

3.2. Accelerating Quantitative Imaging

The computational speed of Julia makes advanced parameter estimation feasible for clinical use. 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 [?]. When fitting an entire human brain, Julia was approximately 90x faster than MATLAB's single-threaded operation [?]. This drastic reduction in computational cost is critical for making advanced quantitative MRI (qMRI) parameters accessible in clinical settings.

4. Challenges in Medical Imaging Formats

Medical imaging formats, particularly DICOM, present significant difficulties for researchers. The core challenges relate to the complexity of the standard and the handling of spatial metadata.

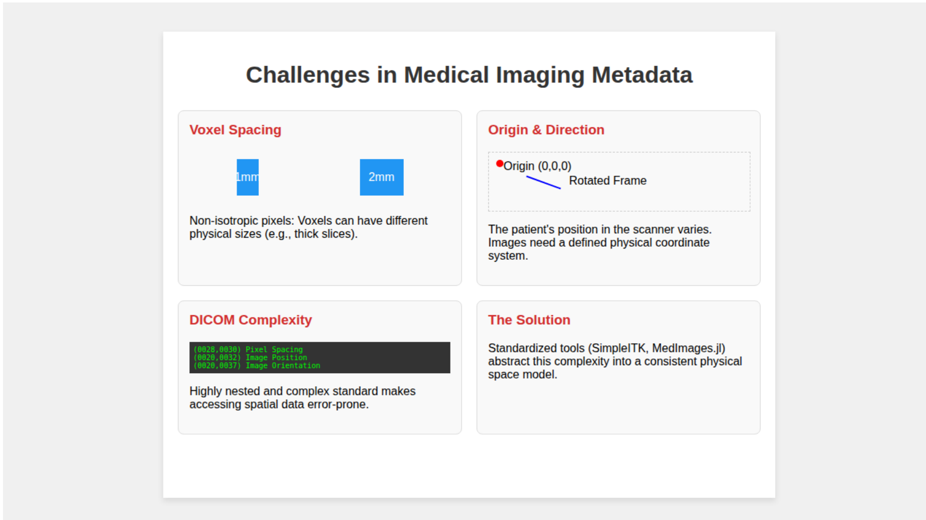

Figure 2: Challenges in medical imaging include handling non-isotropic voxel spacing, varying origins and orientations, and the complexity of the DICOM standard. Standardized tools are needed to abstract these complexities.

4.1. Spatial Metadata: Spacing, Origin, and Direction

Unlike standard images (e.g., JPEGs), medical images represent physical objects in 3D space. They require precise definition of their relationship to the physical world:

- **Voxel Spacing (Non-Isotropic Pixels):** Medical images are often acquired as stacks of 2D slices with different resolutions or thickness (voxel spacing) in different dimensions [?]. The original MRI or CT scans often have non-isotropic pixel spacing [?].
- **Origin:** The physical coordinate of the first voxel $(0, 0, 0)$ relative to a scanner or patient frame.
- **Direction:** The orientation of the image axes in physical space, defined by a direction cosine matrix [?].

Ignoring this metadata leads to invalid operations. For example, adding two images is only valid if they occupy the exact same physical space [?]. A common issue is the "ill-defined coordinate system" when working purely with pixel indices, which fails to account for patient position or scan geometry [?].

4.2. *Format Complexity and DICOM*

The DICOM standard is ubiquitous but notoriously complex. It specifies Information Object Definitions (IODs) and services for communication, making adoption difficult for many researchers [?]. Even when using high-level libraries, accessing spatial metadata can be error-prone due to the highly nested structure of annotations [?]. Researchers often convert DICOM objects into simpler alternative formats (like NIfTI) for analysis, but this conversion risks losing important contextual information [? ?].

5. MedImages.jl: Software Description

MedImages.jl is developed to address these challenges within the Julia ecosystem. It provides a standardized structure for storing and manipulating medical image data and metadata, similar in philosophy to SimpleITK but leveraging Julia's native capabilities.

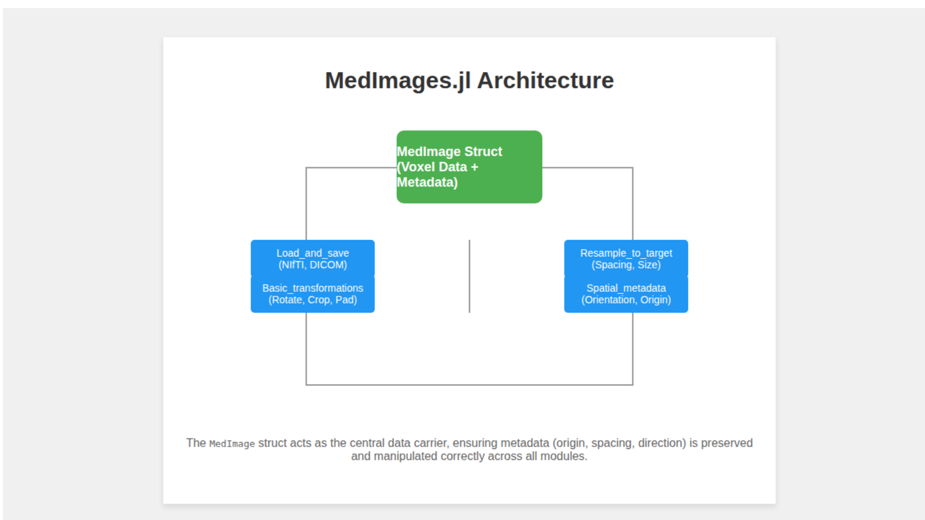

Figure 3: Architecture of MedImages.jl. The `MedImage` struct is central, supporting various modules for loading, resampling, transforming, and spatial metadata handling. The modular design allows for easy extension.

5.1. *Architecture*

The core of the package is the `MedImage` struct, which rigorously encapsulates both the image data and its spatial context:

- `voxel_data`: The multidimensional array of pixel values (e.g., 512x512x3).
- `origin`: Tuple of 3 Float64 values indicating the origin.
- `spacing`: Tuple of 3 Float64 values indicating voxel spacing.
- `direction`: 9-element tuple of Float64 values for orientation cosines.
- `image_type`, `image_subtype`: Enums for modality (MRI, CT, PET) and subtype (T1, T2, FLAIR, etc.).
- `patient_uid`, `study_uid`, `series_uid`: UUIDs for data management.

5.2. Key Functionalities and Usage

5.2.1. Loading and Saving

MedImages.jl supports reading and writing standard formats, ensuring metadata is preserved. The `Load_and_save` module handles the intricacies of file I/O.

```julia
using MedImages

# Load an image from a NIfTI file
img = load_med_image("brain_scan.nii")

# Access metadata
println("Spacing: ", img.spacing)
println("Origin: ", img.origin)
println("Direction: ", img.direction)
```

Listing 1: Loading a medical image

5.2.2. Resampling

The `Resample_to_target` module allows images to be resampled to a new geometry (spacing, size, orientation). This is a critical step for registering images from different sources or modalities, where pixel-to-pixel correspondence is not guaranteed [?].

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

# Resample using linear interpolation
resampled_img = resample_to_spacing(img, new_spacing,
    Interpolator_enum.Linear_en)
```

Listing 2: Resampling an image to a new spacing

5.2.3. *Spatial Metadata Manipulation*

The `Spatial_metadata_change` module provides tools to change orientation and spacing explicitly, ensuring the physical reference frame is maintained. This adheres to the "Physical Space Tenet" [?], where the image is defined by its physical occupation rather than just its array indices.

```
# Change orientation to Right-Posterior-Inferior (RPI)
# This handles the permutation of axes and direction cosines
reoriented_img = change_orientation(img, Orientation_code.
    ORIENTATION_RPI)
```

Listing 3: Changing image orientation

5.2.4. *Basic Transformations*

Common image processing operations like rotation, cropping, and padding are handled by the `Basic_transformations` module. These operations are aware of the spatial metadata, updating the origin and direction as needed.

6. Democratizing Access through Open Source

Open-source libraries like MedImages.jl, SimpleITK, and MONAI play a crucial role in democratizing access to medical image processing by lowering barriers to entry.

6.1. *Boosting Productivity*

Open-source toolkits serve as high-level abstraction layers that hide the intricate, low-level details of core medical imaging standards and algorithms [?]. By abstracting the complexities of DICOM parsing and spatial math, these tools allow researchers to focus on algorithmic innovation rather than boilerplate code. For example, SimpleITK provides a simplified interface to ITK, making powerful algorithms accessible via Python or R [?]. MedImages.jl brings similar benefits to the Julia community, enabling rapid prototyping without sacrificing performance.

6.2. *Enhancing Reproducibility and Collaboration*

Standardized open-source tools ensure reproducibility, a foundational pillar of scientific integrity [?]. When researchers use a common framework for handling metadata, they avoid common pitfalls like coordinate system mismatch. Tools like highdicom ensure that ML model outputs are encoded in standardized DICOM formats, facilitating clinical integration [?]. Open-source projects also foster vibrant communities where code and knowledge are freely exchanged, leading to continuous evolution and enhancement of the software [?].

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 like ITK. By implementing core functionalities in native Julia, it allows for easier deployment and faster precompilation times. 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. Conclusion

MedImages.jl represents a significant step towards a robust medical imaging ecosystem in Julia. By solving the two-language problem, handling complex spatial metadata correctly, and adhering to open-source principles, it empowers researchers to build high-performance, reproducible, and clinically relevant imaging pipelines. As the Julia ecosystem grows, tools like MedImages.jl will be instrumental in unlocking the full potential of this modern language for medical science, enabling faster transition from research to clinical deployment.

Acknowledgements

We acknowledge the Julia community and the developers of ITK and SimpleITK for their pioneering work in defining standards for medical image analysis.

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] 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

[4] S. Ahn, S. G. Ross, E. Asma, J. Miao, X. Jin, L. Cheng, S. D. Wollenweber, R. M. Manjeshwar, Quantitative comparison of osem and penalized likelihood image reconstruction using relative difference penalties for clinical pet, Physics in Medicine and Biology 60 (7 2015). doi:10.1088/0031-9155/60/15/5733.
URL http://dx.doi.org/10.1088/0031-9155/60/15/5733

[5] 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

[6] N. J. Sisco, P. Wang, A. Stokes, R. Dortch, Rapid parameter estimation for selective inversion recovery myelin imaging using an open-source julia toolkit (9 2021). doi:10.1101/2021.09.27.461996.
URL http://dx.doi.org/10.1101/2021.09.27.461996

[7] 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

[8] E. Roesch, J. G. Greener, A. L. MacLean, H. Nassar, C. Rackauckas, T. E. Holy, M. P. H. Stumpf, Julia for biologists, Nature Methods 20 (4 2023). doi:10.1038/s41592-023-01832-z.
URL http://dx.doi.org/10.1038/s41592-023-01832-z

[9] 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

[10] 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

[11] R. Beare, B. Lowekamp, Z. Yaniv, Image segmentation, registration and characterization in <i>r</i> with simpleitk, Journal of Statistical Software 86 (2018). doi:10.18637/jss.v086.i08.
URL http://dx.doi.org/10.18637/jss.v086.i08

[12] 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

[13] 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

[14] 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