



SOFTWARE TOOL ARTICLE

Med-ImageTools: An open-source Python package for robust data processing pipelines and curating medical imaging data

[version 1; peer review: 2 approved with reservations]

Sejin Kim^{1,2}, Michal Kazmierski^{1,2}, Kevin Qu^{id}^{1,3}, Jacob Peoples^{id}⁴, Minoru Nakano¹, Vishwesh Ramanathan², Joseph Marsilla^{1,2}, Mattea Welch¹, Amber Simpson^{4,5}, Benjamin Haibe-Kains^{1,2,4,5}

¹Princess Margaret Cancer Centre, University Health Network, Canada, Toronto, ON, Canada

²Department of Medical Biophysics, University of Toronto, Toronto, ON, Canada

³Faculty of Engineering, University of Toronto, Toronto, ON, Canada

⁴Centre for Health Innovation, Queen's University and Kingston Health Science Centre, Kingston, ON, Canada

⁵Vector Institute, Toronto, ON, Canada

v1 First published: 01 Feb 2023, 12:118
<https://doi.org/10.12688/f1000research.127142.1>
 Second version: 07 May 2024, 12:118
<https://doi.org/10.12688/f1000research.127142.2>
 Latest published: 07 Feb 2025, 12:118
<https://doi.org/10.12688/f1000research.127142.3>

Abstract

Background: Machine learning and AI promise to revolutionize the way we leverage medical imaging data for improving care but require large datasets to train computational models that can be implemented in clinical practice. However, processing large and complex medical imaging datasets remains an open challenge.

Methods: To address this issue, we developed Med-ImageTools, a new Python open-source software package to automate data curation and processing while allowing researchers to share their data processing configurations more easily, lowering the barrier for other researchers to reproduce published works.

Use cases: We have demonstrated the efficiency of Med-ImageTools across three different datasets, resulting in significantly reduced processing times.

Conclusions: The AutoPipeline feature will improve the accessibility of raw clinical datasets on public archives, such as the Cancer Imaging Archive (TCIA), the largest public repository of cancer imaging, allowing machine learning researchers to process analysis-ready formats without requiring deep domain knowledge.

Keywords

medical imaging, deep learning, open source, data processing, dicom, nifti, nnunet

Open Peer Review

Approval Status

	1	2
version 3 (revision) 07 Feb 2025		
version 2 (revision) 07 May 2024	 view	 view
version 1 01 Feb 2023	 view	 view

1. **Rachel Sparks** , King's College London, London, UK

2. **Johann Faouzi** , ENSAI, Rennes, France

Any reports and responses or comments on the article can be found at the end of the article.



This article is included in the **Bioinformatics** gateway.



This article is included in the **Radiomics** collection.



This article is included in the **Python** collection.

Corresponding author: Benjamin Haibe-Kains (benjamin.haibe.kains@utoronto.ca)

Author roles: **Kim S:** Conceptualization, Data Curation, Formal Analysis, Investigation, Methodology, Project Administration, Software, Validation, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing; **Kazmierski M:** Conceptualization, Methodology, Software; **Qu K:** Data Curation, Methodology, Software, Validation; **Peoples J:** Conceptualization, Data Curation, Methodology, Validation, Writing – Review & Editing; **Nakano M:** Resources, Software; **Ramanathan V:** Methodology, Software, Validation; **Marsilla J:** Methodology, Validation; **Welch M:** Conceptualization, Methodology, Resources, Supervision; **Simpson A:** Resources, Supervision, Writing – Review & Editing; **Haibe-Kains B:** Conceptualization, Funding Acquisition, Methodology, Project Administration, Resources, Supervision, Writing – Review & Editing

Competing interests: No competing interests were disclosed.

Grant information: This study has been supported by the Canadian Institutes for Health Research (Project Scheme grant #426366) and the Canadian Cancer Society (Data Transformation grant #707609).

The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Copyright: © 2023 Kim S *et al.* This is an open access article distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

How to cite this article: Kim S, Kazmierski M, Qu K *et al.* **Med-ImageTools: An open-source Python package for robust data processing pipelines and curating medical imaging data [version 1; peer review: 2 approved with reservations]** F1000Research 2023, 12:118 <https://doi.org/10.12688/f1000research.127142.1>

First published: 01 Feb 2023, 12:118 <https://doi.org/10.12688/f1000research.127142.1>

Introduction

Radiology is a powerful modality of data for clinical work — it gives clinicians the ability to see the inner workings of the human body, that cannot be seen from the outside.¹ They can inspect in 2D or 3D, the anatomy surrounding the disease, enabling key information to make life-altering clinical decisions. While regular images taken on cameras or phones are stored in a variety of accessible formats, medical images are encoded in the Digital Imaging and Communications in Medicine (DICOM) standard file format.²

The DICOM standard was developed in the 1980s due to the increasing need for an interoperable standard for 3D medical images across various manufacturers.² A key feature of DICOM is the plethora of metadata fields that store information beyond imaging data, such as patient information, clinical variables, and acquisition parameters. As modern medical practice evolved over the years, the DICOM standard has grown to accommodate more metadata fields and encompass new imaging modalities or therapy information.² This type of data format is unsuitable for imaging analysis as the relevant voxel array must be manually accessed through DICOM hierarchy.² Furthermore, 3D scans are acquired on a slice-by-slice basis; thus, researchers must stitch together data from multiple files to create one 3D image, adding delays caused by disk reads and consolidation processes.²

Specialties that rigorously use imaging data are heavily reliant on DICOM, one of which is radiation oncology. Images are used along every step of the clinical workflow: from deriving a precise diagnosis, to designing personalized radiation therapy plans, and delivering each radiation dose with the appropriate alignment and orientation with brief scans. While the defined standard serves as a good guideline, each manufacturer has slightly different implementations. This is especially the case for DICOM-RT (radiotherapy), a subset of modalities for communicating radiotherapy data.² The DICOM-RT standard includes additional modalities such as RTStruct for contour data, and RTDose for radiotherapy dose maps and dose-volume histograms (DVH).² While the broad adoption of the DICOM standard to accommodate for various use cases has allowed it to become the defacto standard for encoding, storing, and transferring of medical images, its comprehensive nature has made it difficult for researchers to navigate for the purposes of imaging projects.²

Current workflows

The Cancer Imaging Archive (TCIA) (RRID:SCR_008927) is one of the largest public repositories of DICOM images available, with over 140 datasets consisting of more than 60,000 patients.³ The datasets undergo a quality assurance process to ensure the recorded clinical variables are coherent and the DICOM files are not missing any important metadata fields.³ These stringent processes and infrastructure have allowed TCIA to become one of the most comprehensive repositories for biomedical imaging datasets, inviting researchers from different fields to explore new ideas and methods on high quality datasets.³

While the underlying data and its annotations are of clinical quality, processing the dataset for subsequent analysis requires a non-trivial amount of effort: manually reorganizing directories and matching radiation therapy structures (RTStruct), referred to as DICOM-RT contours, and radiation therapy plans (RTDose/RTPlan) to its corresponding images.⁴ This is partly due to the inherently complex nature of clinical datasets, as data is collected on the basis of need and iterative improvements, not structured scientific inquiries. It is also sometimes due to the lack of familiarity from machine learning (ML) and artificial intelligence (AI) researchers in handling the DICOM files for their analytical pipelines. Typical AI imaging datasets have pairwise associations of one image to a single ground-truth label.⁵ However, one patient in clinical datasets may have multiple RTStruct and RTDose files with one imaging acquisition, one RTStruct and RTDose with multiple images, or worst of all, multiple RTStruct and RTDose files with multiple images.⁶ In any of these cases, the directories are not always intuitively structured to help researchers understand which files correspond with another.

Once researchers have successfully curated the dataset into an organized structure for analysis, in order to process the raw dataset into analysis-ready format, they must choose from a variety of processing parameters, ranging from voxel spacing, RTStruct name parsing, and hounsfield unit (HU) window levels, based on the design of their analysis. While these implicit decisions for image processing are often arbitrary, they can greatly impact model training and performance, but are not transparently disclosed in publications.⁷ This leads to the difficulty of reproducibility of medical deep learning research, adding another deterrence to clinical adoption.

Furthermore, there are a limited number of software packages that researchers can use to quickly parse DICOM-RT files into analysis-ready arrays (Table 1). Chief among those, SlicerRT,⁸ an extension of 3Dslicer⁹ (RRID:SCR_005619), an open-source DICOM visualization tool, has been widely used by the medical imaging community. Despite its broad adoption, it is not easily interfaced with Python, the most popular language for machine learning and deep learning.¹⁰ **RT-Utils** is a lightweight, open-source Python package designed to handle RTStruct files with relative ease and simplicity,

Table 1. Comparison of existing medical imaging processing packages and their features. RTSTRUCTs: DICOM-RT Contours; RTDOSEs: DICOM-RT Dose; CT: Computed Tomography; MRI: Magnetic Resonance Imaging; Nifti: Neuroimaging Informatics Technology Initiative; Nrrd: Nearly raw raster data; DICOM: Digital Imaging and Communications in Medicine.

	3Dslicer + SlicerRT	RT-Utils	PlatiPy	Med- ImageTools
Native Python interface		●	●	●
Command-line interface				●
Handles RTSTRUCTs	●	●	●	●
Handles RTDOSEs	●		●	●
Handles images (CT/MRI)	●		●	●
Built-in image transformations	●		●	●
Exports to analysis-ready Nifti/Nrrd	●		●	●
Image registration	●		●	
Built-in bulk processing of entire datasets				●
Automatic parsing of DICOM metadata	●			●

allowing users to easily export contours into segmentation masks in arrays. However, the functionalities of RT-Utils are limited to the RTStruct modality. **PlatiPy** is a recent processing library and analysis toolkit for medical imaging, mainly designed for the context of radiation therapy. It features a comprehensive set of image manipulation functions such as registration and atlas-based segmentation methods, allowing researchers the flexibility to process imaging data into any format they need. However, PlatiPy does not solve the inherent complexity of clinical datasets, and researchers must spend hours reorganizing the data into a structured set of samples and labels. The current landscape of open-source medical imaging tools highlights the need for a native Python package that can parse large DICOM/DICOM-RT datasets to an analysis-ready format for ML/AI development in a consistent, reproducible workflow.

To address the limitations of the current software packages used to process medical images, we developed Med-ImageTools,¹¹ a new Python package designed to help researchers transform complex medical datasets into analysis-ready format with few lines of code. It is also focused on helping researchers develop transparent and reproducible medical image processing pipelines by addressing most of the boilerplate code required for image transformations and processing parallelization. While Med-ImageTools has many modular functions for image, contour, and dose input/output (IO) built on popular frameworks such as SimpleITK, TorchIO and PyDicom, such functionalities are redundant and available in other open-source packages as well. Our main contribution is in the development of AutoPipeline and will mainly discuss its functionalities and implementation.

Methods

AutoPipeline

AutoPipeline is the main feature of Med-ImageTools that allows users to easily process raw DICOM clinical datasets into analysis-ready Nrrd or Nifti files, which are commonly used file formats for 3D volumetric data. It is interfaced using the command line, so the user only needs to submit a single command into the terminal to execute the three core steps in the AutoPipeline process (**Figure 1**):

1. **Crawl:** The crawler opens every DICOM file in the dataset, indexing important metadata, such as unique identifiers, modality information, and references to other modalities. This produces a database of every unique image and DICOM-RT modality.
2. **Connect:** In this step, each patient's indices of unique files of different modalities are connected to form one coherent sample. There are various heuristics the user may choose to connect samples. The default option is through DICOM metadata, as datasets derived from clinical practice are expected to have corresponding metadata that references unique identifiers of the parent image or RTPlan. Alternative heuristics allow users to deal with anomalies with corrupted or missing metadata.
3. **Process:** All identified samples are processed according to imaging and transformation parameters defined by the user.

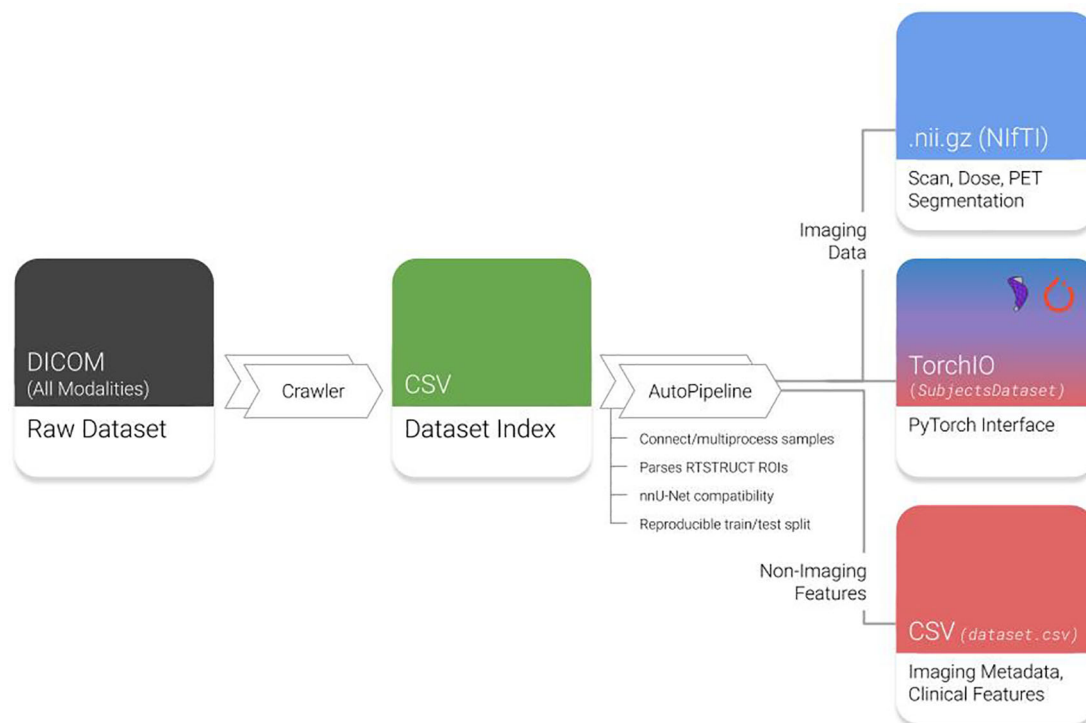


Figure 1. Overview of Med-ImageTools features. Raw datasets are indexed by the Crawler module and automatically processed by AutoPipeline. DICOM: Digital Imaging and Communications in Medicine; RTSTRUCTs: DICOM-RT Contours; PET: Positron emission tomography. NiftI: Neuroimaging Informatics Technology Initiative; CSV: Comma-separated values.

The AutoPipeline feature can be directly interfaced once Med-ImageTools is installed. To install Med-ImageTools, the recommended method is *via* the PyPI package repository in a virtual environment. Running the ‘*pip install med-image-tools*’ command will install the latest version of Med-ImageTools and its associated dependencies. Now, whenever the user is in the virtual environment where Med-ImageTools is installed, they can directly interact with the AutoPipeline feature in the command line. The simplest way to use it is through ‘*autopipeline input_directory output_directory*’. This will automatically process the dataset located in ‘*input_directory*’ and process them at ‘*output_directory*’ using the default parameters. An extended tutorial of AutoPipeline and all its associated parameters are available [here](#). At the present time, there are no minimum system requirements for Med-ImageTools as it will run regardless of number of processor cores or memory (RAM). However, if the researcher can leverage greater number of cores and RAM, it will allow the AutoPipeline to be parallelized and process the data faster.

As the crawl and process steps are computationally intensive, all steps in AutoPipeline are automatically parallelized to efficiently leverage all available computational resources. While the output of the AutoPipeline processing can result in terabytes of images, the crawl is limited to a few kilobytes of a metadata database, making it an ideal asset to share with other researchers as a detailed descriptor of a medical imaging dataset. We therefore propose to attach the crawled metadata spreadsheet to large TCIA datasets to allow Med-ImageTools users to process large datasets much faster and more efficiently. These databases are expected to save up to 1000 core-hours of crawling per dataset, accumulating over 2000 core-hours of computation saved per user. By standardizing a commonly repeated imaging processing pipeline into a single unified package, we hope to improve the reproducibility and transparency of future medical imaging research.

Use cases

We showcase the value of the AutoPipeline implemented in Med-ImageTools v1.0.0 for processing three medical imaging datasets, namely Pancreatic-CBCT-SEG from TCIA, liver metastasis private dataset and RADCUR pending public release on TCIA, in order of complexity and sizes. In each use case, we initially describe the process.

Using pre-crawled datasets on the Pancreatic Cone-beam Computed Tomography (CBCT)¹²

40 patients with abdominal CBCT scans and their associated contours of regions of interest and other organs, publicly available on TCIA

The Pancreatic-CT-CBCT-SEG dataset was processed twice using AutoPipeline: once from scratch, and once using the pre-crawled dataset, available on the tcia-crawls branch of Med-ImageTools. Processing from scratch, the dataset took 10.77 core-hours (10:46), whereas using a pre-crawled database allowed the processing to finish in 9.14 core-hours (9:08) (Figure 2a). The pre-crawled database reduced processing time by 1.63 core-hours, representing an 18% increase in total processing speed or 2 minutes 27 seconds per patient. The time saved from pre-crawled databases is not a substantial quantity for datasets with less than 100 patients. However, when scaled up to larger TCIA datasets such as OPC-Radiomics¹³ (n=606) and NLST¹⁴ (n=26254), it can save researchers 24.7 core-hours and 1072 core-hours, respectively (Figure 2b). While it may vary depending on the research infrastructure utilized, these databases can result in significant savings in billing.

Comparing manual vs automatic processing of the liver metastasis dataset

97 patients with abdominal CT scans and their associated contours of liver and gross tumour volumes (GTV), data access described in data availability section below.

The liver metastasis dataset was processed using the Slicer API to export a CT scan along with segmentations of the liver and GTVs for each patient. In the initial DICOM dataset, each patient had a single RTSTRUCT file along with one or more CT series, one of which was referenced by the RTSTRUCT. The first step of the process was to load the entire DICOM dataset into the Slicer DICOM database *via* the graphical user interface (GUI), to make the data available inside the Slicer scripting environment. Then, we wrote a script to export an initial set of candidate segmentations for the liver and GTVs for each patient, along with the referenced CT scan. This script leveraged the Slicer API to identify the CT series that was referenced by the patient's RTSTRUCT and used an *ad hoc* string filtering function to identify candidate segmentations. Finally, we iteratively refined the set of exported RTSTRUCT contours on a patient-by-patient basis, based on visual checking, and physician feedback. Although this step was time-consuming, there is, at present, no substitute for manual verification to ensure data quality and correctness. On the other hand, the initial database construction and export script using the Slicer API achieved results roughly analogous to the automatic output of AutoPipeline. The export script is approximately 170 lines long and took 7.5 hours to run on the whole dataset-this script is available on our GitHub repository. In contrast, on the same 6-core machine (16GB RAM, Windows 10), AutoPipeline took only 2.3 hours with 1 command line, mainly accelerated by parallelization (Figure 2c).

Comparing manual vs automatic processing of the RADCURE dataset

3,219 patients with head and neck CT scans and their associated contours of organs at risk (OAR) for radiotherapy, pending public release on TCIA.

The RADCURE dataset is a large dataset of 3,219 head and neck cancer patients and their radiotherapy planning data. The dataset was extracted from two separate treatment planning software systems, meaning the directories and DICOM metadata were structured differently. The directories from each system were restructured using general heuristics, and any abnormal cases were flagged and manually organized. We used SimpleITK and PyDicom to extract the imaging and contour data from the DICOM files, which underwent a similar iterative process as the liver metastatic dataset. The script is over 1000 lines long and takes 30-40 hours to run on the whole dataset using a single core-this script is available on our GitHub repository. AutoPipeline scales dramatically based on the number of cores available, which enables the entire dataset to be processed in 40 minutes using 32 cores, automatically managing the directories from different systems and extracting the contours (Figure 2d).

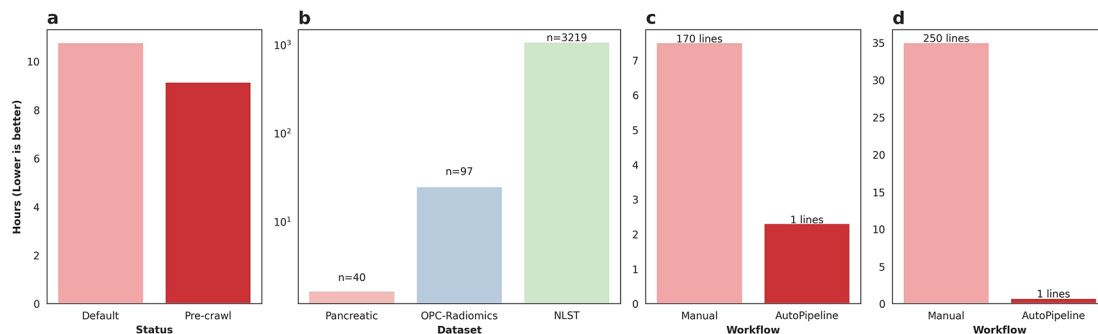


Figure 2. a) Time taken to process the Pancreatic-CT-CBCT-SEG dataset with (Pre-crawl) and without (Default) the pre-crawled databases. b) Amount of core-hours saved by using pre-crawled databases across various public TCIA datasets. c) Time taken to process the liver metastasis dataset manually vs AutoPipeline. d) Time taken to process the RADCURE dataset manually vs AutoPipeline.

One caveat of these comparisons is that the iterative process of filtering appropriate contours, which can add weeks to months of cooperation between the researchers and the physicians, were already conducted in the original processing steps. Various data cleaning steps that require human intervention, such as sorting contour names or selecting specific subseries acquisitions, cannot be fully automated for the foreseeable future. The Med-ImageTools team aims to add features to the package that will assist researchers in these steps, such as adding a flag to visualize all unique contours without requiring code, adding subseries detection to the crawler, and publishing a set of regular expressions (regex) that can be used to automatically choose contours from prominent head and neck datasets on TCIA. Also, these comparisons do not take into account the time it takes for researchers to develop the manual processing scripts. Hence, the actual amount of time saved for the researcher may be greater than the reported times.

Discussion

Although ML and AI promise to revolutionize the way we leverage medical imaging data for improving care, they require large datasets to train computational models that can be implemented in clinical practice. However, processing large and complex medical imaging datasets remains an open challenge. To address this issue, we developed Med-ImageTools, a new open-source software package to automate data curation and processing while allowing researchers to share their data processing configurations more easily, lowering the barrier for other researchers to reproduce published works. The AutoPipeline feature will improve the accessibility of raw clinical datasets on public archives, such as TCIA, allowing machine learning researchers to process analysis-ready formats without requiring deep domain knowledge.

While our package aims to address challenges encountered across a few medical imaging labs, we acknowledge that there may be infinite other issues that may arise in DICOM datasets. This is one of the key reasons why our package is open-source for community involvement and contribution. Also, as stated previously, there are certain onerous tasks that cannot be automated and must undergo human supervision. These aspects of researcher-clinician collaboration are an inevitable part of medical imaging research and are subject to delay.

No single solution can completely solve the reproducibility crisis of medical deep learning research, due to a variety of issues ranging from ambiguous data processing techniques to stochasticity of model training. However, community-centered open-source solutions and increased clinical adherence to data standards, such as contour nomenclature,¹⁵ can incrementally improve research quality and reproducibility, and make medical deep learning research more accessible for everyone.

Data availability

The Pancreatic-CT-CBCT-SEG dataset is available on The Cancer Imaging Archive at: <https://doi.org/10.7937/TCIA.ESHQ-4D90>.

The pre-crawled Med-ImageTools database of the Pancreatic-CT-CBCT-SEG dataset is available on GitHub at: https://github.com/bhklab/med-imagetools/raw/tcia-crawls/csvs/imgtools_Pancreatic-CT-CBCT-SEG.csv.

The liver dataset was collected as part of the study titled *Radiomics artificial intelligence modelling for prediction of local control for colorectal liver metastases treated with radiotherapy*¹⁶ with Institutional Review Board and informed consent. Interested individuals should contact the senior author for access to the dataset, which will be made available following standard institutional rules for IRB and HIPAA.

The RADCURE dataset is pending public release on TCIA and the article will be updated upon release. Data published on TCIA are subject to TCIA data usage policies and restrictions.

Software availability

Source code available from: github.com/bhklab/med-imagetools also available on the PyPI software repository: pypi.org/project/med-imagetools.

Scripts for use cases are available from: github.com/bhklab/med-imagetools/tree/F1000Research

Archived source code at time of publication: <https://doi.org/10.5281/zenodo.7212760>¹⁷

License: [MIT](https://creativecommons.org/licenses/by/4.0/)

Its associated documentation and tutorials are available at: med-imagetools.readthedocs.io

References

1. Brant WE, Helms CA: **Fundamentals of Diagnostic Radiology**. 2012.
2. Mildenerger P, Eichelberg M, Martin E: **Introduction to the DICOM standard**. *Eur. Radiol.* 2002; **12**: 920–927.
[PubMed Abstract](#) | [Publisher Full Text](#)
3. Clark K, et al.: **The Cancer Imaging Archive (TCIA): maintaining and operating a public information repository**. *J. Digit. Imaging.* 2013; **26**: 1045–1057.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
4. Law MYY, Liu B: **DICOM-RT and Its Utilization in Radiation Therapy**. *RadioGraphics.* 2009; **29**: 655–667.
[PubMed Abstract](#) | [Publisher Full Text](#)
5. Deng J, et al.: **ImageNet: A large-scale hierarchical image database**. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009; pp. 248–255.
[Publisher Full Text](#)
6. Vallières M, Kay-Rivest E, et al.: **Data from Head-Neck-PET-CT**. *The Cancer Imaging Archive*. 2017.
[Publisher Full Text](#)
7. Sabottke CF, Spieler BM: **The Effect of Image Resolution on Deep Learning in Radiography**. *Radiol. Artif. Intell.* 2020; **2**: e190015.
8. Pinter C, Lasso A, Wang A, et al.: **SlicerRT: radiation therapy research toolkit for 3D Slicer**. *Med. Phys.* 2012; **39**: 6332–6338.
[PubMed Abstract](#) | [Publisher Full Text](#)
9. Fedorov A, et al.: **3D Slicer as an image computing platform for the Quantitative Imaging Network**. *Magn. Reson. Imaging.* 2012; **30**: 1323–1341.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
10. Yli-Heikkilä M: **Data Science Languages**. *DSNS '19*. University of Helsinki; 2019.
11. Kazmierski M, Kim S, Ramanathan V, et al.: **bhklab/med-imagetools: v4.4.0**. *Zenodo*. 2022.
[Publisher Full Text](#)
12. Camp B: **Pancreatic-CT-CBCT-SEG - The Cancer Imaging Archive (TCIA) Public Access - Cancer Imaging Archive Wiki**.
[Publisher Full Text](#)
13. Kwan JYY, et al.: **Radiomic Biomarkers to Refine Risk Models for Distant Metastasis in HPV-related Oropharyngeal Carcinoma**. *Int. J. Radiat. Oncol. Biol. Phys.* 2018; **102**: 1107–1116.
[PubMed Abstract](#) | [Publisher Full Text](#)
14. Clark K: **National Lung Screening Trial**.
[Publisher Full Text](#)
15. Mayo CS, et al.: **American Association of Physicists in Medicine Task Group 263: Standardizing Nomenclatures in Radiation Oncology**. *Int. J. Radiat. Oncol. Biol. Phys.* 2018; **100**: 1057–1066.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
16. Hu R, Chen I, et al.: **Radiomics artificial intelligence modelling for prediction of local control for colorectal liver metastases treated with radiotherapy**. *Phys. Imaging Radiat. Oncol.* 2022; **24**: 36–42.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
17. Cranmer K, Kreiss S: **decouple software associated to arXiv: 1401.0080**. [Code] *Zenodo*. 2014.
[Publisher Full Text](#)

Open Peer Review

Current Peer Review Status: ? ?

Version 1

Reviewer Report 17 October 2023

<https://doi.org/10.5256/f1000research.139617.r189038>

© 2023 Faouzi J. This is an open access peer review report distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.



Johann Faouzi

ENSAI, Rennes, France

The authors present their Python package for data processing pipelines and curating medical imaging data in this manuscript. Overall the manuscript is well written and clearly presents the main contributions of this package to the community. Nonetheless, some points on the software side could be improved. Please find my comments below:

* How does the output format used compare to other formats such as other popular formats such as BIDS (Brain Imaging Data Structure, which is maybe less general since it's intended for brain imaging data)?

* Figure 2b (and the corresponding text): Raw saved running time, even though useful, is irrelevant without knowing the total running time. Please provide also the saved time as a percentage of the total running time.

* Figures 2c and 2d: "1 lines" => "1 line"

* The authors should consider renaming the "master" branch as the "main" branch (see <https://sfconservancy.org/news/2020/jun/23/gitbranchname/>).

* Is there a reason for Python 3.8 not being tested in the continuous integration (CI) (maybe a limit on the number of jobs)? By the way, Python 3.7 is no longer supported and Python 3.11 has been released for over 9 months (<https://devguide.python.org/versions/>). The Python versions tested in the CI should probably be updated.

* Flake8 is installed in the CI but never used. I ran the command locally (flake8 | wc -l) and obtained 2163 errors or warnings, which is not good. It would be nice to clean the source code a bit.

* There is no code coverage associated to the tests, which is an important limitation. Please add code coverage to your project. There are "only" 16 tests for the moment, so I suspect that a

substantial part of the source code is not tested.

* I don't understand the point of using the "-s" option with pytest. I don't find the output (when using this option) to be clear or useful to identify tests that would fail for instance. In particular, I don't think that printing stuff when running the tests in the CI is relevant.

Is the rationale for developing the new software tool clearly explained?

Yes

Is the description of the software tool technically sound?

Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?

Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?

Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?

Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Open source software, Python programming, machine learning, medical applications

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Author Response 05 Apr 2024

Benjamin Haibe-Kains

We thank Dr Faouzi for his feedback regarding our manuscript titled 'Med-ImageTools: An open-source Python package for robust data processing pipelines and curating medical imaging data'. We have added our responses to their comments here in subdivided sections of comments/questions (Q), in bold, our corresponding responses/answers (A), and any relevant sections of the manuscript as an italicized bullet point, with changes emphasized in bold.

Q: The authors present their Python package for data processing pipelines and curating medical imaging data in this manuscript. Overall the manuscript is well written and clearly presents the main contributions of this package to the community. Nonetheless, some

points on the software side could be improved. Please find my comments below:

A: We thank the reviewer for their positive understanding of the authors' work and constructive feedback, especially on the technical aspects of the package.

Q: How does the output format used compare to other formats such as other popular formats such as BIDS (Brain Imaging Data Structure, which is maybe less general since it's intended for brain imaging data)?

A: We thank the reviewer for bringing the BIDS standard to the authors' attention. The BIDS standard is very effective in standardizing brain imaging datasets by adhering to an intuitive structure describing various modalities of brain MRI. Med-ImageTools is mainly geared towards use cases that are broader and less specific than BIDS and its implementation would require fundamental reworkings of the package. However, the benefit of having a BIDS-native output cannot be understated and this feature will be added to the development team's ongoing development goals. We have revised the relevant section from 'Current Workflows' as follows:

- *While Med-ImageTools has many modular functions for image, contour, and dose input/output (IO) built on popular frameworks such as SimpleITK, TorchIO and PyDicom, such functionalities are redundant and available in other open-source packages as well. **We have tailored our core features towards broader use cases and development workflows, instead of modality or disease type specific workflows, such as BIDS.** Our main contribution is in the development of AutoPipeline and will mainly discuss its functionalities and implementation.*

Q: Figure 2b (and the corresponding text): Raw saved running time, even though useful, is irrelevant without knowing the total running time. Please provide also the saved time as a percentage of the total running time.

A: We appreciate the reviewer's point about Figure 2b. However, the reported time saved is reported in core-hours, which is a comprehensive unit of total running time. This is because the crawling occurred independently from the end-to-end processing of the datasets to ensure an accurate measure of core-hours saved. Reporting core-hours saved allows the reader to estimate their time/cost savings based on their specific hardware availabilities, either local machine, server-based, or cloud infrastructure, agnostic to the hardware used by the authors. We have emphasized this point in our manuscript as follows:

- *However, when scaled up to larger TCIA datasets such as OPC-Radiomics 13 (n=606) and NLST 14 (n=26254), it can save researchers 24.7 core-hours and 1072 core-hours, respectively (Figure 2b). **The resources saved are reported in core-hours to allow a hardware-agnostic estimation of time and cost savings.** While it may vary depending on the research infrastructure utilized, these databases can result in significant savings in billing.*

Q: Figures 2c and 2d: "1 lines" => "1 line"

Q: The authors should consider renaming the "master" branch as the "main" branch (see <https://sfconservancy.org/news/2020/jun/23/gitbranchname/>).

Q: Is there a reason for Python 3.8 not being tested in the continuous integration (CI) (maybe a limit on the number of jobs)? By the way, Python 3.7 is no longer supported and Python 3.11 has been released for over 9 months (<https://devguide.python.org/versions/>). The Python versions tested in the CI should probably be updated.

A: We thank the reviewer for their comments. The figures and repository have been updated to address these points.

Q: Flake8 is installed in the CI but never used. I ran the command locally (flake8 | wc -l) and obtained 2163 errors or warnings, which is not good. It would be nice to clean the source code a bit.

A: We thank the reviewer for raising the concern about the source code's sanitation. The authors have reviewed the flake8 output and incorporated it into our codebase. You can find all changes in the following commit: <https://github.com/bhklab/med-imagetools/commit/a8c0c3be53b4c9b43da12603b222d261d417a18c>.

Q: I don't understand the point of using the "-s" option with pytest. I don't find the output (when using this option) to be clear or useful to identify tests that would fail for instance. In particular, I don't think that printing stuff when running the tests in the CI is relevant.

A: We thank the reviewer's attention to detail in Med-ImageTools' CI/CD framework. The -s option is enabled for verbose outputs from Med-ImageTools to be carried through to the standard output of the tests in CI. In the past we have had OS-specific issues that were elucidated through the -s flag, but have removed it for future releases.

Q: There is no code coverage associated to the tests, which is an important limitation. Please add code coverage to your project. There are "only" 16 tests for the moment, so I suspect that a substantial part of the source code is not tested.

A: We thank the reviewer for highlighting the lack of code coverage association of these tests. We agree that this is a clear limitation of the Med-ImageTools' development pipeline, and have significantly increased the number of unit tests and integrated them into our CI/CD process. Future development efforts will continue to add more tests to our pipeline, including a badge detailing code coverage on our README.

Competing Interests: There are no competing interests to disclose.

Reviewer Report 17 October 2023

<https://doi.org/10.5256/f1000research.139617.r189015>

© 2023 Sparks R. This is an open access peer review report distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.



Rachel Sparks

King's College London, London, England, UK

I think the motivation and justification for why a light weight Python module to support data loading and preprocessing of radiotherapy DICOM formats was very well articulated and clear.

And I applaud the authors on making such a useful tool available to the community.

However, I have two major concerns that must be addressed prior to this article being acceptable for indexing.

First there are no details as to the design of Med-Image-Tools this includes important information such as:

(1) dependencies on other python packages

(2) what functions or other tools are being leveraged within the tool and how these are arranged.

(3) the structure of the outputs of this pipeline specifically the csv file corresponding to the 'Dataset Index', the imaging and non-imaging feature outputs. For instance for imaging data how are pre-operative scans and segmentations saved and named (is there a convention is this something the user must write)

(4) While there is a reference to the tutorial about the processing pipeline I think a high level overview listing the possible options would be appropriate to help the reader understand what this tool can offer.

My other concern is related to the use cases. While lines of code is an easy metric to report it is mostly meaningless, some lines of code may be boilerplate or easy to write while others may obviously be different. It is also easy to pad or alter depending on the level of skill of the programmer. I think reporting run time is far more meaningful to your argument and I would stick to these.

Finally, specifically, for the RADCURE dataset, it is not clear to me why you are comparing run time on a single core versus run time on a 32 core machine. I suspect the reported run time for the comparison could be significantly reduced by using a very simple parallelization which should be achievable tools available in python. I would suggest a fairer comparison for this specific example, it wont detract from the best argument you have which is there you are presenting a nice package so that other developers and researchers do not have to spend time re-writing code. Highlighting where some time savings may be gained is in some ways a secondary advantage.

Finally, as a minor note the statement that SlicerRT is not easily interfaced with python is not a fully justified statement. One can use the python interface in Slicer to run python scripts and even run from the command prompt by called s'licer.exe --python-code'. However, I think you can modify this to make a true and equally motivated statement which is that SlicerRT requires one to install Slicer/SlicerRT and stay within the slicer ecosystem to process data. Your tool lacks such dependencies so can be a more light weight and does not require running python through another software system.

Is the rationale for developing the new software tool clearly explained?

Yes

Is the description of the software tool technically sound?

No

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?

No

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?

No

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?

Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Software development; open source software; medical image computing

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Author Response 05 Apr 2024

Benjamin Haibe-Kains

We thank Dr Sparks for her feedback regarding our manuscript titled 'Med-ImageTools: An open-source Python package for robust data processing pipelines and curating medical imaging data'. We have added our responses to their comments here in subdivided sections of comments/questions (Q), in bold, our corresponding responses/answers (A), and any relevant sections of the manuscript as an italicized bullet point, with changes emphasized in bold.

Q: I think the motivation and justification for why a light weight Python module to support data loading and preprocessing of radiotherapy DICOM formats was very well articulated and clear. And I applaud the authors on making such a useful tool available to the community.

A: We thank the reviewer for their recognition of Med-ImageTools' contribution to the medical imaging community and positively evaluating the authors' works.

Q: However, I have two major concerns that must be addressed prior to this article being acceptable for indexing.

First there are no details as to the design of Med-Image-Tools this includes important information such as:

(1) dependencies on other python packages

A: We thank the reviewer for pointing out the need to communicate the technical

dependencies of Med-ImageTools. We have updated our 'AutoPipeline' section as follows, to include information on how to find the list of package dependencies.

- *Running the 'pip install med-imagetools' command will install the latest version of Med-ImageTools and its associated dependencies **defined by the requirements.txt file.***

The 'Current workflows' section also contains a concise description of what frameworks Med-ImageTools leverages such as follows:

- *While Med-ImageTools has many modular functions for image, contour, and dose input/output (IO) built on popular frameworks such as SimpleITK, TorchIO and PyDicom...*

Q: (2) what functions or other tools are being leveraged within the tool and how these are arranged.

A: We thank the reviewer for calling attention to the need to clarify how other tools/packages function within Med-ImageTools. We have added some information to our 'AutoPipeline' section to clarify major operations performed using external tools/packages as follows. For a granular understanding of the integration behind every external tool, we defer the reader to learn about the design choices made throughout the package and its implications by gaining a deeper understanding of the source code available through our GitHub repository.

- *1. Crawl: The crawler opens every DICOM file in the dataset using Pydicom, indexing important metadata, such as unique identifiers, modality information, and references to other modalities. This produces a database of every unique image and DICOM-RT modality.*
- *3. Process: All identified samples are processed according to imaging and transformation parameters defined by the user. The user can configure parameters such as the pixel spacing in mm(s), which specific modalities to process, the number of cores they want to use for multiprocessing, and define nnU-Net specific flags as well. **The images are manipulated using SimpleITK without requiring user intervention.***
- *As the crawl and process steps are computationally intensive, all steps in AutoPipeline are automatically parallelized **using the joblib backend** to efficiently leverage all available computational resources.*

Q: (3) the structure of the outputs of this pipeline specifically the csv file corresponding to the 'Dataset Index', the imaging and non-imaging feature outputs. For instance for imaging data how are pre-operative scans and segmentations saved and named (is there a convention is this something the user must write)

A: We thank the reviewer for this question and for highlighting the lack of clarity of our description of AutoPipeline. The 'Dataset Index' is designed to be an autogenerated output of the Crawler and does not require any user interaction to be created; this is simply there to ensure convenient user experience. We have made this and other aspects of the outputs clearer by adding the following sentences to the 'AutoPipeline' section.

- ***Med-ImageTools also generates files that are not analysis-ready images, such as the autogenerated 'Dataset Index' (Figure 1), and stores them in a folder named ".imgtools" at the path of the 'input_directory'. This is to ensure a convenient user experience and intuitive folder structure by hiding extraneous components.***

Q: (4) While there is a reference to the tutorial about the processing pipeline I think a high

level overview listing the possible options would be appropriate to help the reader understand what this tool can offer.

A: We agree with the reviewer that adding this information to our description of AutoPipeline can help readers quickly understand the tool's strengths before interfacing with the GitHub repository or ReadTheDocs. We have updated the 'AutoPipeline' section accordingly.

- *3. Process: All identified samples are processed according to imaging and transformation parameters defined by the user. **The user can configure parameters such as: the pixel spacing in mm(s), which specific modalities to process, the number of cores they want to use for multiprocessing, and define nnU-Net specific flags as well. The images are manipulated using SimpleITK without requiring user intervention.***

Q: My other concern is related to the use cases. While lines of code is an easy metric to report it is mostly meaningless, some lines of code may be boilerplate or easy to write while others may obviously be different. It is also easy to pad or alter depending on the level of skill of the programmer. I think reporting run time is far more meaningful to your argument and I would stick to these.

A: We appreciate the reviewer's consideration in the metrics reported in Figure 2. However, we believe this metric emphasizes Med-ImageTools' strength allowing users to process DICOM datasets with a single command line interaction, in contrast to writing dozens of lines of Python/R/MATLAB scripts. One particular reason why we believe a single command line is useful to highlight is to express the ease-of-use for users that may not be comfortable writing their own code for DICOM to Nifti/Nrrd processing, especially if they lack deep domain knowledge.

Another, perhaps more, exciting reason is to show the potential of building automated workflows on top of Med-ImageTools. If one were to build an end-to-end automated pipeline starting from clinical DICOM datasets to outputting an inference-ready deep learning model, they could easily develop a reproducible processing step by configuring only the command line interaction of Med-ImageTools, making debugging and custom configurations simpler since the developer would not have to rely on a static script. We have elaborated on Med-ImageTools' contributions for improved scientific reproducibility in the 'Discussion' section as follows.

- *The AutoPipeline feature will improve the accessibility of raw clinical datasets on public archives, such as TCIA, allowing machine learning researchers to process analysis-ready formats without requiring deep domain knowledge. **Another exciting potential of Med-ImageTools lies in building automated workflows using AutoPipeline. For a researcher to build an end-to-end automated pipeline starting from clinical DICOM datasets to outputting an inference-ready deep learning model, they could easily develop a reproducible processing step by configuring only the command line interaction of Med-ImageTools, making debugging and custom configurations simpler since the developer would not have to rely on a static script.***
- *No single solution can completely solve the reproducibility crisis of medical deep learning research, due to a variety of issues ranging from ambiguous data processing techniques to stochasticity of model training. However, community-centered open-source solutions and increased clinical adherence to data standards, such as contour nomenclature,¹⁵ can incrementally improve research quality and reproducibility, and make medical deep learning research more accessible for everyone.*

Q: Finally, specifically, for the RADCURE dataset, it is not clear to me why you are comparing run time on a single core versus run time on a 32 core machine. I suspect the reported run time for the comparison could be significantly reduced by using a very simple parallelization which should be achievable tools available in python. I would suggest a fairer comparison for this specific example, it wont detract from the best argument you have which is there you are presenting a nice package so that other developers and researchers do not have to spend time re-writing code. Highlighting where some time savings may be gained is in some ways a secondary advantage.

A: We thank the reviewer for their comment about Figure 2d. The goal of the comparison was not only to show a reduction in processing time based on parallelization, but to show that our package's design does not bottleneck any multiprocessing backends and brings meaningful acceleration on large datasets. In other words, we wanted to show that a 32x increase in computational hardware was going to bring at least 32x improvement in performance as observed in Figure 2d. This is in contrast to Figure 2c which does not show similar levels hardware acceleration, but its usefulness in parsing private, unseen data. We have expanded on these ideas in the manuscript as follows:

- *In contrast, on the same 6-core machine (16GB RAM, Windows 10), AutoPipeline took only 2.3 hours with 1 command line, mainly accelerated by parallelization (Figure 2c). **The authors involved in validating Med-ImageTools' effectiveness on this private dataset had no active involvement in the development of the package before its application. This highlights the package's robustness on unseen data and its potential utility for multi-centre collaborations to ensure consistent processing. One use case might be to enable federated learning platforms to automatically process each node's datasets without requiring any user intervention.***
- *AutoPipeline scales dramatically based on the number of cores available, which enables the entire dataset to be processed in 40 minutes using 32 cores, automatically managing the directories from different systems and extracting the contours (Figure 2d). **These results demonstrate Med-ImageTools' design does not bottleneck any multiprocessing backends and brings meaningful acceleration on very large datasets, such as RADCURE.***

Q: Finally, as a minor note the statement that SlicerRT is not easily interfaced with python is not a fully justified statement. One can use the python interface in Slicer to run python scripts and even run from the command prompt by called 'slicer.exe --python-code'. However, I think you can modify this to make a true and equally motivated statement which is that SlicerRT requires one to install Slicer/SlicerRT and stay within the slicer ecosystem to process data. Your tool lacks such dependencies so can be a more light weight and does not require running python through another software system.

A: We thank the reviewer for this point. We have revised the relevant section from 'Current Workflows' as follows to better justify our statement.

- *Despite its broad adoption, **batch data processing with Slicer requires custom scripting in Python, to be executed in the Slicer ecosystem. Rather than simply installing a package within their Python environment, users must install the Slicer application and add any other dependencies, not provided by Slicer, into the application environment. As a result, machine learning projects relying on Python***

for data preprocessing will have their code fragmented across multiple environments—the Slicer environment for data processing, and another Python environment for data analysis and machine learning.

Competing Interests: There are no competing interests to disclose.

The benefits of publishing with F1000Research:

- Your article is published within days, with no editorial bias
- You can publish traditional articles, null/negative results, case reports, data notes and more
- The peer review process is transparent and collaborative
- Your article is indexed in PubMed after passing peer review
- Dedicated customer support at every stage

For pre-submission enquiries, contact research@f1000.com

F1000Research