Computer Methods and Programs in Biomedicine 231 (2023) 107374

Contents lists available at ScienceDirect

Computer Methods and Programs in Biomedicine

journal homepage: www.elsevier.com/locate/cmpb

PyRaDiSe: A Python package for DICOM-RT-based auto-segmentation pipeline construction and DICOM-RT data conversion

Elias Rüfenacht[a,*], Amith Kamath[a], Yannick Suter[a], Robert Poel[b], Ekin Ermiş[b], Stefan Scheib[c], Mauricio Reyes[a]

[a] ARTORG Center for Biomedical Engineering Research, University of Bern, Murtenstrasse 50, Bern 3008, Switzerland
[b] Department of Radiation Oncology, Inselspital, Bern University Hospital and University of Bern, Bern, Switzerland
[c] Varian Medical Systems Imaging Laboratory GmbH, Baden, Switzerland

ARTICLE INFO

Article history:
Received 15 October 2022
Revised 9 January 2023
Accepted 23 January 2023

MSC:
68T45
68U05
68U10
62P10
92C55

PACS:
87.53.Tf
89.70.+c

Keywords:
Auto-segmentation
Deep learning
Radiotherapy
DICOM
DICOM RT structure sets
DICOM RTSS conversion

ABSTRACT

Background and objective: Despite fast evolution cycles in deep learning methodologies for medical imaging in radiotherapy, auto-segmentation solutions rarely run in clinics due to the lack of open-source frameworks feasible for processing DICOM RT Structure Sets. Besides this shortage, available open-source DICOM RT Structure Set converters rely exclusively on 2D reconstruction approaches leading to pixelated contours with potentially low acceptance by healthcare professionals. PyRaDiSe, an open-source, deep learning framework independent Python package, addresses these issues by providing a framework for building auto-segmentation solutions feasible to operate directly on DICOM data. In addition, PyRaDiSe provides profound DICOM RT Structure Set conversion and processing capabilities; thus, it applies also to auto-segmentation-related tasks, such as dataset construction for deep learning model training.

Methods: The PyRaDiSe package follows a holistic approach and provides DICOM data handling, deep learning model inference, pre-processing, and post-processing functionalities. The DICOM data handling allows for highly automated and flexible handling of DICOM image series, DICOM RT Structure Sets, and DICOM registrations, including 2D-based and 3D-based conversion from and to DICOM RT Structure Sets. For deep learning model inference, extending given skeleton classes is straightforwardly achieved, allowing for employing any deep learning framework. Furthermore, a profound set of pre-processing and post-processing routines is included that incorporate partial invertibility for restoring spatial properties, such as image origin or orientation.

Results: The PyRaDiSe package, characterized by its flexibility and automated routines, allows for fast deployment and prototyping, reducing efforts for auto-segmentation pipeline implementation. Furthermore, while deep learning model inference is independent of the deep learning framework, it can easily be integrated into famous deep learning frameworks such as PyTorch or Tensorflow. The developed package has successfully demonstrated its capabilities in a research project at our institution for organs-at-risk segmentation in brain tumor patients. Furthermore, PyRaDiSe has shown its conversion performance for dataset construction.

Conclusions: The PyRaDiSe package closes the gap between data science and clinical radiotherapy by enabling deep learning segmentation models to be easily transferred into clinical research practice. PyRaDiSe is available on https://github.com/ubern-mia/pyradise and can be installed directly from the Python Package Index using `pip install pyradise`.

1. Introduction

The clinical adoption of computer-aided diagnosis and intervention planning systems in radiotherapy has increased treatment quality and reduced the duration until treatment commencement

* Corresponding author.
E-mail address: elias.ruefenacht@unibe.ch (E. Rüfenacht).

https://doi.org/10.1016/j.cmpb.2023.107374

[1]. Both aspects are associated with prolonged, progression-free survival in some aggressive tumor types (e.g., glioblastoma [2]); thus, further reducing delays until treatment commencement could potentially improve the progression-free survival of numerous cancer patients [3]. One possible path for further reducing delays in the clinical radiotherapy workflow is automating laborious and error-prone workflow steps using deep learning (DL) techniques [4]. Recently, DL-based methods have shown massive potential for many real-world applications [5,6]. Similarly, many studies have shown that DL techniques provide the potential for accurately segmenting anatomical structures [7,8] and tumor sites (i.e., target structures) [9,10] as is required in radiotherapy. In addition, study results indicate that the gap in segmentation quality is consistently reducing with evolving DL techniques compared to human experts [7]. However, differences in data formats for medical image segmentation introduce challenges in adopting DL techniques. While data science relies typically on discretized image formats (e.g., NIfTI, NRRD, MHA), radiotherapy workflows depend on the Digital Imaging and Communications in Medicine (DICOM) [11] format, precisely the DICOM Radiotherapy Structure Set (DICOM-RTSS) format, containing lists of contour control points. The main reason for using discrete medical image formats in data science is that most DL-based segmentation architectures (e.g., U-Net [12]) operate exclusively on data aggregated on a grid. However, the processing of continuously spaced contour points on the state-of-the-art segmentation architectures is not foreseen. Thus, accurate data conversion between discrete and continuous spaces is necessary for working with the clinical DICOM-RTSS data format.

In addition to the data format challenges, building auto-segmentation solutions involves pre-processing the provided DICOM image series and post-processing the generated segmentation masks. Pre-processing the patient's images is necessary to clean and adjust their properties (e.g., intensity value normalization, resolution, registration to an atlas) to fit the model's input. Hence, pre-processing images for DL model inference often consists of many image analysis operations that affect intensity value properties (e.g., intensity value range, amount of noise) and spatial properties (i.e., origin, direction, spacing, size). While the intensity value properties are no longer of interest after DL model inference, spatial properties typically propagate to the generated segmentation masks to correspond with the model's input. However, these spatial properties often differ from the ones of the original DICOM image series, and restoring the original DICOM image series spatial properties is inevitable for having segmentation masks aligned with the original DICOM image series. Post-processing predominantly concerns segmentation mask modifications and quality enhancements, such as excluding minor false-positive segmentation errors for single instance structures. In summary, the route to developing radiotherapy-oriented, DL-based auto-segmentation solutions that operate directly on clinical DICOM data contains numerous challenges that must be overcome before clinic deployment is feasible. However, to the best of our knowledge, there is currently no Python package that addresses these challenges holistically.

Despite this, some projects address specific challenges in building auto-segmentation solutions. While some projects address DICOM-RTSS-specific challenges, others aim at closing the gap between data science and its clinical application in a holistic approach by providing a profound general-purpose DL inference framework for medical applications. For converting discrete medical images to DICOM-RTSS and vice versa, the RT-Utils [13] and the DicomRTTool [14] packages provide essential functionality. However, both packages are constrained to a simple 2D-based conversion algorithm that causes DICOM-RTSS contours to appear synthetic (i.e., pixelated contours), which may limit the acceptance of generated DICOM-RTSS contours in clinics. Regarding the data loading, pre-processing, and post-processing, multiple medical imaging frameworks are available, such as the Insight Toolkit (ITK) [15] or its simplified version, the Simple Insight Toolkit (SimpleITK) [16]. These imaging frameworks are on a technically mature level, provide a rich feature set, and are well-accepted by the community. However, while these frameworks are usually able to load DICOM image series, they lack importing functionality for DICOM-RTSS. Nevertheless, their tremendous functionality and robust design provide a well-established basis for implementing new packages.

Specific packages for data pre-processing are also publicly available, such as TorchIO [17], which focuses on data pre-processing and augmentation for DL model training with PyTorch [18]. In addition, TorchIO provides functionality for DL model inference and image transformation invertibility but lacks processing capabilities for DICOM-RTSS. Although TorchIO may be extendable for processing DICOM-RTSS data, structured handling of DICOM image series metadata would nonetheless be challenging because the required data structures and mechanisms are missing.

MONAI (Medical Open Network for AI) [19] follows a more holistic approach by providing DL model training and inference functionality, including training routines, neural network architectures, and loss functions. In addition, MONAI provides a bundle called MONAI Deploy suited for deploying DL-based applications in a clinical production environment. However, MONAI Deploy is currently unable to export DICOM-RTSS files limiting its applicability to radiotherapy segmentation tasks. Furthermore, the MONAI framework, while getting to be a de-facto standard in DL for medical imaging, is specifically designed for the PyTorch DL framework, thus, limiting fast switching between DL frameworks and deploying existing DL models to radiotherapy clinics.

A further project following a holistic approach is DeepNeuro [20], which provides a TensorFlow-based [21] DL model deployment system suited for segmentation that lacks DICOM-RTSS conversion capabilities. Unfortunately, this project is not actively maintained anymore as of October 2022.

In summary, many open-source projects aim to resolve challenges in bringing DL techniques into clinics. However, holistic projects, such as MONAI or DeepNeuro, lack DICOM-RTSS conversion functionality that is challenging to extend due to metadata propagation. On the other hand, projects focusing exclusively on generating DICOM-RTSS provide solely 2D-based conversion approaches leading to pixelated structures that may limit result acceptance by physicians and do not provide additional functionality for DL model inference.

In this paper, we present PyRaDiSe, an open-source Python (Py) package for developing deployable, radiotherapy-oriented (Ra), DICOM-based (Di) auto-segmentation (Se) solutions. PyRaDiSe is DL framework-independent but can easily integrate most DL frameworks, such as PyTorch or TensorFlow. The package addresses the following challenges for building radiotherapy-oriented auto-segmentation solutions: handling DICOM data, managing and converting DICOM-RTSS data (incl. a 2D-based and a 3D-based conversion algorithm), invertible pre-processing, and post-processing. In addition to building auto-segmentation solutions, PyRaDiSe allows for converting and curating DICOM image series and DICOM-RTSS data to simplify segmentation training dataset construction. Therefore, PyRaDiSe is highly flexible, allows for fast prototyping, and facilitates a fast transition of data science research results into clinical radiotherapy research.

2. Methods

The intended use of PyRaDiSe is to close the gap between DL and clinical radiotherapy research by providing a versatile and holistic Python package to build deployable DICOM-based auto-

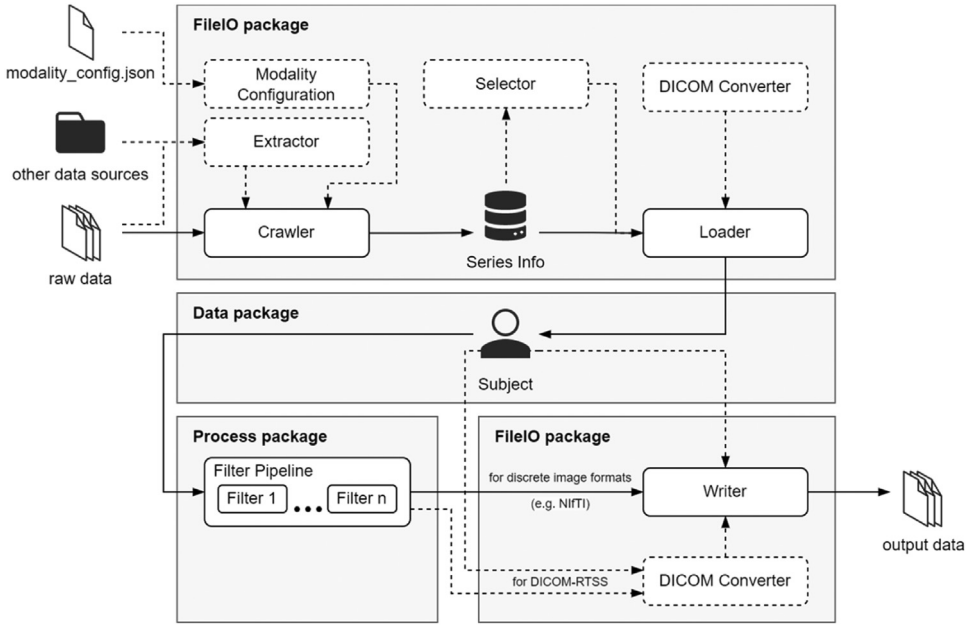


Fig. 1. Schematic illustration of PyRaDiSe and its packages. The typical workflow is shown with solid lines and alternative and optional processing routes are depicted by dashed lines.

segmentation solutions and to evaluate corresponding research results in a clinical environment. In addition, the functional scope of PyRaDiSe provides specific solutions such as converting clinical DICOM and DICOM-RTSS data to various discrete medical image file formats used for research purposes (e.g., NIfTI) or preprocessing DICOM data for training or dataset construction of DL models. In combination with other tools such as Docker [22] and GitHub[1], PyRaDiSe facilitates and speeds up the development of portable and version-controlled auto-segmentation solutions and renders fast evolution cycles feasible. Furthermore, due to its holistic approach, simple integration capabilities, and flexible extensibility, PyRaDiSe-based auto-segmentation solutions can be integrated into existing segmentation solutions or made available via the cloud to a broader community of users. In summary, PyRaDiSe addresses three main challenges of auto-segmentation in clinical radiotherapy research: image data handling (incl. DICOM), conversion from and to DICOM-RTSS, and processing of clinical research data with DL methods.

The three main components of PyRaDiSe are the *fileio* package for data import and export, the *data* package for data handling, and the *process* package for pre-processing, DL model inference, and post-processing (Fig. 1).

2.1. FileIO package

The *fileio* package provides highly automated, versatile, and easy-to-use functionality for data import, export, and DICOM-RTSS conversion. First, automated because importing and converting medical image data, particularly DICOM data, often is complex, confusing, and time-consuming due to references among the entities (e.g., processing DICOM registration data with references to two DICOM image series) and varying data structures. Second, versatile because loading, converting, and serializing medical image data requires many computational resources, and thus a streamlined import and export process featuring low computational complexity is essential. Furthermore, versatile because the *fileio* package can handle input and output data of various forms and structures. Third, easy-to-use because data ingestion, conversion, and serialization should be as simple as possible for developers, reducing development time and improving code readability Listing 1.

Data import. In contrast to the standard way of loading data, where data loading is necessary before accessing series-associated metadata is possible, PyRaDiSe allows for retrieving and selecting pre-loading information (so-called `SeriesInfo`) before data loading. This approach reduces memory requirements (depending on the data to be loaded) and speeds up the subsequent loading procedure. In order to retrieve the pre-loading information, a data `Crawler` first searches for loadable files in the target filesystem location. Then it extracts the required information using its associated metadata retrieval routines to generate the pre-loading information. Afterward, retaining selected pre-loading information is achieved using optional selection routines. For example, if retrieving data from filesystem hierarchies containing additional data, this selection step effectively allows for excluding unwanted files, such as data from failed acquisitions where patients had moved or data not required for auto-segmentation (e.g., unused magnetic resonance (MR) imaging sequences). Subsequently, an appropriate `Loader` ingests the selected data based on the pre-loading information and applies the required data conversion procedures. Finally, the loaded data is available for further processing using the data model described in the *data* package.

DICOM metadata retrieval. The retrieval procedures for constructing the pre-loading information are highly automated; however, some metadata may not automatically be deducible. In general, if working with DICOM data, all metadata is accessible and deducible except for cases where multiple uni-modal DICOM image series (e.g., two MR sequences) should be loaded. In these cases, discrimination between the uni-modal images is impossible because the DICOM standard provides just a coarse modality identification (e.g., MR for magnetic resonance imaging) and limited standardized modality-specific details, such as the X-ray tube current for computed tomography (CT) image series. Unfortunately, the limited standardization of this metadata and clinic-specific and

[1] https://github.com/.

```python
import pyradise.fileio as fio

def main(input_dir_path: str,
         output_dir_path: str
         ) -> None:
    # Assumption: The modality configuration file is
    # existing for the subject to load

    # Crawl for data and generate the pre-loading
    # information
    crawler = fio.SubjectDicomCrawler(input_dir_path)
    series_info = crawler.execute()

    # (optional) Make a selection on the pre-loading
    # information
    selection = fio.ModalityInfoSelector(keep=('T1',))
    series_info = selection.execute(series_info)

    # Load the DICOM series data to a subject instance
    loader = fio.SubjectLoader()
    subject = loader.load(series_info)

    # Write the subject to disk as NIfTI files
    # Note: NIfTI is the default writer setting
    writer = fio.SubjectWriter()
    writer.write(output_dir_path, subject)


if __name__ == '__main__':
    input_path = 'PATH_TO_SUBJECT_INPUT_DIR'
    output_path = 'PATH_TO_SUBJECT_OUTPUT_DIR'
    main(input_path, output_path)
```

Listing 1. DICOM-RTSS to NIfTI conversion example using PyRaDiSe.

machine vendor-dependent naming of DICOM attributes introduces ambiguities in these cases and hinders robust discriminability. In order to resolve the issue of restricted access to modality and acquisition details, the *fileio* package provides two approaches: either use a separate modality configuration file per patient or implement a metadata extraction method. The approach selection is up to the user and may depend on the specific application and the data access. Below these two approaches are described in more detail.

- *Modality Configuration File Approach.*
  The modality configuration file approach requires a separate JSON configuration file per patient located in the patient's data directory. This file specifies the modality name for each series of DICOM images and contains additional data for manual identification of the corresponding DICOM image series. Nevertheless, the skeleton of this configuration file can be generated automatically by the appropriate `Crawler`. However, it must be modified by the user manually or via a separate script (example in Appendix A). After configuration, the `Crawler` uses this configuration file to identify and discriminate the corresponding data. Therefore, this approach is better suited for applications where modality details are inaccessible through a programmable method. Furthermore, this approach is preferable to generate reproducible results if processing the same data multiple times is needed, such as during dataset construction and curation for DL training.
- *Modality Extractor Approach.*
  The second approach, relying on a `ModalityExtractor`, requires the implementation of a prototype class that uses rule-based deduction, external data sources, or an artificial intelligence technique to identify the corresponding modality details upon runtime. In contrast to the modality configuration file approach, this approach is better suited for applications where the required information is guaranteed to possess a standardized structure or is accessible via a third-party system by calling the appropriate API (e.g., an external database or a DL-based sequence identification algorithm).

Discrete image metadata retrieval. In the case of working with discrete image data (e.g., NIfTI images), the required metadata (i.e., modality name, organ name, annotator name) is not retrievable automatically because these image formats typically do not contain content-related metadata. However, in practice, this metadata is often accessible in varying formats, either in the filename, a lookup table, or another external data source such as a database. In order to render this metadata retrievable, the *fileio* package provides separate extractor prototype classes for each metadata type (i.e., `Modality`, `Organ`, `Annotator`). The implementation of the extractor prototypes[2] is straightforward and allows for maximum flexibility.

Data conversion. In addition to metadata such as references to DICOM images, DICOM-RTSS files consist of 3D point coordinate lists describing contour control points in the physical space, with each list describing a contour of one anatomical structure on one image slice. The combination of multiple corresponding coordinate lists provides the spatial segmentation of an anatomical structure in space. Because most DL segmentation models and image analysis methods operate exclusively on data in discrete spaces, converting DICOM-RTSS data into discretized segmentation masks is required. In PyRaDiSe, the necessary converting routines are an extended version of the ones from the RT-Utils package and are called automatically during the loading of DICOM-RTSS data. A necessity for the DICOM-RTSS conversion is the availability of the referenced DICOM image series that hold additional information about the spatial orientation of the contours and references to the image slices. In PyRaDiSe, this is checked automatically by the provided conversion functionality, and the user does not need to assign the referenced DICOM image series to the corresponding converter. In addition, the DICOM-RTSS conversion routines check for possible DICOM registrations (i.e., image transforms) assigned to the referenced DICOM image series such that they are applied to the DICOM-RTSS too. Besides converting DICOM-RTSS to segmentation masks, the *fileio* package also provides functionality to convert one or multiple binary segmentation masks into a multi-structure DICOM-RTSS dataset that is readable by standard RT planning tools. Similar to the opposite conversion direction, this procedure requires a DICOM image series to establish references in the DICOM-RTSS file. In this regard for this conversion, PyRaDiSe provides two different conversion algorithms with distinctive advantages that operate on the slices (2D) or the volume (3D)(see Fig. 2). The 2D-based algorithm is an extended version of the RT-Utils packages algorithm that, among other things, allows for smooth contours in-plane. However, the extended algorithm can not compensate for fast contour changes in neighboring slices. On the other hand, the volume-operating converter (3D) can compensate for these remaining errors by mesh-based

[2] The API reference (see https://pyradise.readthedocs.io/) provides implementation examples for each extractor type.

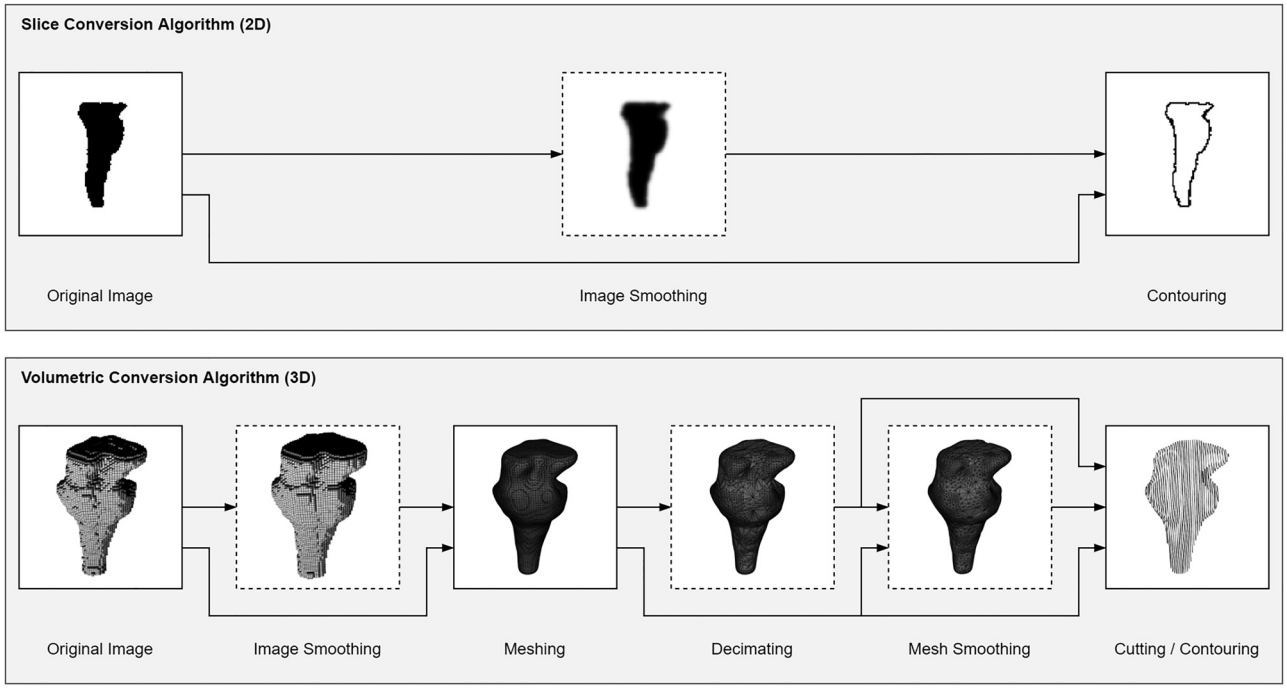


Fig. 2. Schematic illustration of the available segment-to-contour conversion algorithms on a brain stem segmentation from an internal dataset. Optional processing steps are emphasized with dashed borders.

3D smoothing coupled with a reslicing operation. This volumetric conversion provides naturally appearing geometries at the cost of prolonged computation time. Both conversion algorithms allow for fine-graded parameterization.

Data export. Besides data import and conversion, the *fileio* package offers export capabilities to serialize data to various file formats, such as NIfTI or DICOM-RTSS. Especially, exporting DICOM-RTSS helps in scenarios where healthcare professionals are the intended audience of an auto-segmentation solution because they may not possess the necessary tools and knowledge to convert discrete segmentation masks into DICOM-RTSS. In addition to the different export file formats, the *fileio* package allows specific `Writers` to automate the combination of the original input and the generated data in a single output directory or a Zip file. This combination process guarantees that the original input data or the input directory content remains unchanged and is available in the output. This feature is helpful when deploying an auto-segmentation solution in a Docker container with separate input and output directories.

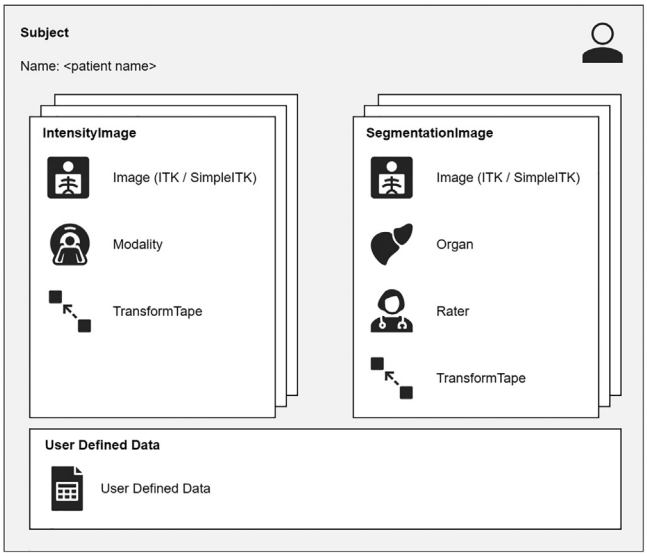


Fig. 3. Schematic illustration of the `Subject` and its associated components.

2.2. Data package

The *data* package provides a lightweight, reproducible, and extensible RT-oriented data model. First, lightweight because working with medical image data requires large amounts of memory; thus, the introduced functionality should consume a minimum of additional memory. Second, reproducible because the data model should not bias or influence the evaluation of different auto-segmentation models on the same data. Third, extensible so that additional data entities (e.g., DICOM RT Dose) can be added in the future to extend the functional scope of PyRaDiSe.

Subject. The data model's central component is the `Subject` (Fig. 3), which contains all loaded data from a single patient. The content of the `Subject` includes but is not limited to the patient's identification (i.e., the patient's name or an anonymized identifier), the patient-associated `IntensityImages` (e.g., a CT scan and multiple MR images), and `SegmentationImages` (e.g., loaded and converted segmentation masks from a DICOM-RTSS or segmentation masks generated by an auto-segmentation algorithm). Furthermore, the `Subject` allows the incorporation of additional user-defined data related to the patient.

For ease of use, the loading routines provided as part of the *fileio* package automatically construct a `Subject` from the imported data such that the user does not need to construct it from the different data entities. After data loading and `Subject` construction, all downstream processing, including data export, takes place on the `Subject` instance such that the different processing routines have access to all data on a subject level.

Images. The main contents of a `Subject` instance are the images associated with the patient. In contrast to the image types of well-known medical imaging libraries such as SimpleITK or ITK,

the PyRaDiSe image types hold additional information for identification and discrimination purposes. PyRaDiSe differentiates between `IntensityImages`, such as CT, MR, or ultrasound images (US), and `SegmentationImages`, which contain either a manually or automatically generated segmentation mask. In addition to the image data, `IntensityImages` include a `Modality` that provides information about the imaging modality (e.g., CT, MR, or US) and acquisition parameters (e.g., T1-weighted (T1w), T1-weighted post-contrast (T1c), or fluid-attenuated inverse recovery (FLAIR) for MR sequences) when the `Subject` contains multiple uni-modal `IntensityImages` such that these can be differentiated. In contrast to `IntensityImages`, `SegmentationImages` contain content-related data about the `Organ` depicted on the segmentation mask and information about the `Annotator` that generated the segmentation. Although the `Annotator`'s name is freely choosable, it can be anonymized or describe an auto-segmentation algorithm instead of a human expert. In addition to the image type-specific content, both image types contain a mechanism called `TransformTape` that records modifications and transformations of the image data to allow reproducibility and partial invertibility. Furthermore, this feature renders restoration of spatial image properties (i.e., image origin, direction, spacing, and size) feasible after applying one or multiple modification procedures to the image (e.g., image resampling and registration to an atlas before DL model inference). However, the inversion's availability is limited to filters that perform mathematically invertible operations (more details see Chapter 4 and Table 1).

Content-related image information. The type of content-related image information (i.e., `Modality`, `Organ`, and `Annotator`) depends on the image type and allows discrimination between images, as mentioned earlier. For ease of use and maximum flexibility, all image information types contain a name as a string that allows anonymization and enables the incorporation of as much information as is desired by the user.

Extensibility & flexibility. The design approach of the data model is to keep things simple such that extensibility can be assured. Thus, the flat data hierarchy, strictly considering polymorphism principles and utilizing type hints throughout the entire PyRaDiSe package, ensures the straightforward implementation of new data types, such as DICOM RT Dose. Furthermore, these design principles enable high flexibility for implementing new features.

2.3. Process package

The *process* package offers a selection of pre-processing and post-processing routines for manipulating `IntensityImages` and `SegmentationImages` that incorporate functionality from popular imaging frameworks such as SimpleITK and ITK. For DL model inference, the *process* package incorporates prototypes and implementation examples to stay DL framework independent, allowing for maximum flexibility and fast prototyping. Furthermore, the optional use of a DL framework drastically reduces the memory footprint of PyRaDiSe itself, which is beneficial if PyRaDiSe is used for conversion purposes only.

The base components of the *process* package are `Filter`, `FilterParams`, and `FilterPipeline`, which allow for the construction of reproducible and invertible processing pipelines. The current version of the package includes the most often applied `Filters` for building auto-segmentation solutions. In order to facilitate user-driven extensions and implementations of additional components, abstract base types and implementation examples are given and documented in the API reference[3] Listing 2.

[3] https://pyradise.readthedocs.io/.

```python
import pyradise.fileio as fio
import pyradise.process as proc

# Load DICOM data
crawler = fio.SubjectDicomCrawler(subject_dir_path)
series_info = crawler.execute()
subject = fio.SubjectLoader().load(series_info)

# Create the filter pipeline and warn on
# non-invertible filters
pipeline = proc.FilterPipeline(None, None, True)

# Add a resampling filter to the pipeline
size = (256, 256, 256)     # resample output size
spacing = (1.0, 1.0, 1.0)  # resample output spacing
pipeline.add_filter(proc.ResampleFilter(),
                    proc.ResampleFilterParams(size,
                                              spacing))

# Add a normalization filter to the pipeline
pipeline.add_filter(proc.ZScoreNormFilter(),
                    proc.ZScoreNormFilterParams())

# Add here additional filters for example a
# DL model inference filter

# Add a playback filter to invert the data
# transformations (i.e., resampling, normalization)
invert_params = proc.PlaybackTransformTapeFilterParams()
pipeline.add_filter(proc.PlaybackTransformTapeFilter(),
                    invert_params)

# Apply the pipeline to the subject
subject = pipeline.execute(subject)

# Write the processed subject to disk as NIfTI
fio.SubjectWriter().write(output_dir_path, subject)
```

Listing 2. `FilterPipeline` construction and application to a `Subject` instance.

Filter & filter parameters. `Filters` are data modification routines parameterized by `FilterParams`. The design of the `Filters` simplifies the processing because they operate directly on `Subject` instances, thus, reducing the amount of boilerplate code and improving code readability. However, the `Images` to which the `Filter`'s modification routine is applied are specifiable for most given `Filters`. In contrast to other medical image analysis frameworks, the provided `Filters` incorporate an invertibility feature if their operation is reversible. This feature also allows for reproducibility by logging the `Filter` applications to an `Image` on a `TransformTape` assigned to the corresponding `Image`. For inversion, the `TransformTape` can be played back on the corresponding `Image` to undo the modifications recorded. The invertibility feature allows restoring spatial image properties after processing, typically at the end of an auto-segmentation pipeline, so that the generated segmentations align with the input images (more details see Chapter 4).

Table 1 enlists all `Filters` provided in the package and the `Image` type they modify.

Table 1
Overview of `Filters` and `FilterParams` implemented in the *process* package with indicators for the image type the `Filter` modifies (int. = `IntensityImages`, seg. = `SegmentationImages`).

Module	Filter / filter parameters (* = `Filter` or `FilterParams`)	Invertible	Image type	
			int.	seg.
Intensity	`ZScoreNorm`*	✓	✓	–
	`ZeroOneNorm`*	✓	✓	–
	`RescaleIntensity`*	✓	✓	–
	`ClipIntensity`*	–	✓	–
	`Gaussian`*	–	✓	–
	`Median`*	–	✓	–
	`Laplacian`*	–	✓	–
Orientation	`Orientation`*	✓	✓	✓
Registration	`IntraSubjectRegistration`*	✓	✓	✓
	`InterSubjectRegistration`*	✓	✓	✓
Resampling	`Resample`*	✓	✓	✓
Modification	`AddImage`*	–	✓	✓
	`RemoveImageByOrgan`*	–	–	✓
	`RemoveImageByAnnotator`*	–	–	✓
	`RemoveImageByModality`*	–	✓	–
	`MergeSegmentation`*	–	–	✓
Inference	`Inference`*	–	–	✓
Post-Proc.	`SingleConnectedComponent`*	–	–	✓
	`AlphabeticOrganSorting`*	–	–	✓
Invertibility	`PlaybackTransformTape`*	N/A	✓	✓

Filter pipeline. The `FilterPipeline` is an easy-to-use and valuable component for combining multiple filters into an executable sequence, reducing boilerplate code upon `Filter` execution. Furthermore, it allows for experimenting with different variants of `Filter` combinations during development and enables a flexible processing pipeline construction process.

Extensibility & flexibility. The *process* package offers a set of selected `Filters` that are, in our opinion, sufficient for building simple to intermediate auto-segmentation solutions. This set of `Filters` includes, among others, functionality for intensity value manipulation (i.e., intensity normalization), affine registration of `IntensityImages` from the same (i.e., `IntraSubjectRegistrationFilter`) or two different `Subjects` (i.e., `InterSubjectRegistrationFilter`) using SimpleITK functionality, and segmentation mask manipulation. Nonetheless, specific applications may require extending the current capabilities; this can be achieved easily by implementing task-specific `Filters` for which examples and descriptions are provided in the API reference. These examples also illustrate the wrapping of SimpleITK and ITK filters such that the user can easily reuse and combine existing, community-proven processing routines. Furthermore, these examples and explanations also allow for straightforward implementation of other advanced registration third-party tools, such as SimpleElastix [23].

3. Results

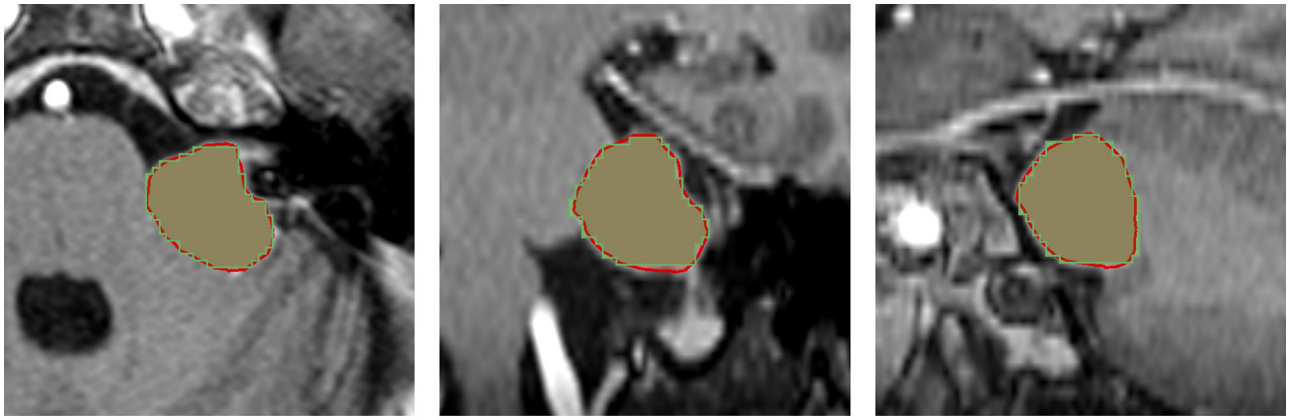
PyRaDiSe is hosted on the Python Package Index (PyPI) for straightforward installation of the latest version using the command `pip install pyradise`. In addition, the source code is publicly available on GitHub[4] under the terms of the Apache 2.0 license, and the documentation is hosted on Read the Docs[5], including descriptions of the classes and functions. Furthermore, examples in the documentation demonstrate the intended use of PyRaDiSe in small parts covering isolated functionality and more extensive examples encompassing complete use cases. The offered examples are available on GitHub or directly rendered in the documentation. In all examples, MR images and segmentations of the head of five subjects from the TCIA [24] "Segmentation of Vestibular Schwannoma from Magnetic Resonance Imaging: An Open Annotated Dataset and Baseline Algorithm" [25,26] dataset are used. Each subject comprises a T1-weighted post-contrast (Gd) image, a T2-weighted image, and manual segmentations of the tumor volume, the cochlea, and the skull. In contrast to the original dataset, the secondary DICOM-RTSS and all DICOM-RT Dose and DICOM-RT Plans were removed, such that the example data does not contain data unnecessary for the demonstrations. In addition to DICOM data, the example data also contains all data in NIfTI format, which was converted with PyRaDiSe. Hence, executing both provided conversion examples is immediately possible after downloading the example data. Furthermore, a PyTorch-based U-Net segmentation model is provided for the inference example, which segments the skull based on the images given. This model was trained for 15 epochs on images from 50 subjects of the original dataset with a learning rate of 0.0001 and a binary cross-entropy loss. This model did not converge during training and is for demonstration purposes only.

The provided examples mimic different aspects of building an auto-segmentation pipeline for brain tumor segmentation. The following sections summarize the examples that cover the dedicated functionality of PyRaDiSe.

3.1. Data conversion

The examples *DICOM to NIfTI Conversion* and *NIfTI to DICOM-RTSS Conversion* illustrate the conversion capabilities of PyRaDiSe. The first example demonstrates the conversion of DICOM images and DICOM-RTSS to NIfTI. This conversion procedure results in a set of NIfTI files that either contains the intensity values of the corresponding DICOM image or a binary segmentation mask of the related structure in the DICOM-RTSS (Fig. 4). All images possess the same spatial properties inherited from the DICOM image files, preserving the alignment between the images and structures. Fur-

[4] https://github.com/ubern-mia/pyradise.
[5] https://pyradise.readthedocs.io/.



(a) Axial view (b) Coronal view (c) Sagittal view

Fig. 4. Illustration of conversion result from DICOM-RTSS (red-filled structure) to NIfTI (green-filled structure) overlaid on the T1-weighted post-contrast image of subject VS-SEG-001 from the example dataset, as displayed by 3D Slicer [27].

thermore, this example demonstrates both metadata retrieval approaches – modality configuration file and extraction via a user-implemented `ModalityExtractor` – that are required to discriminate between the two MR.

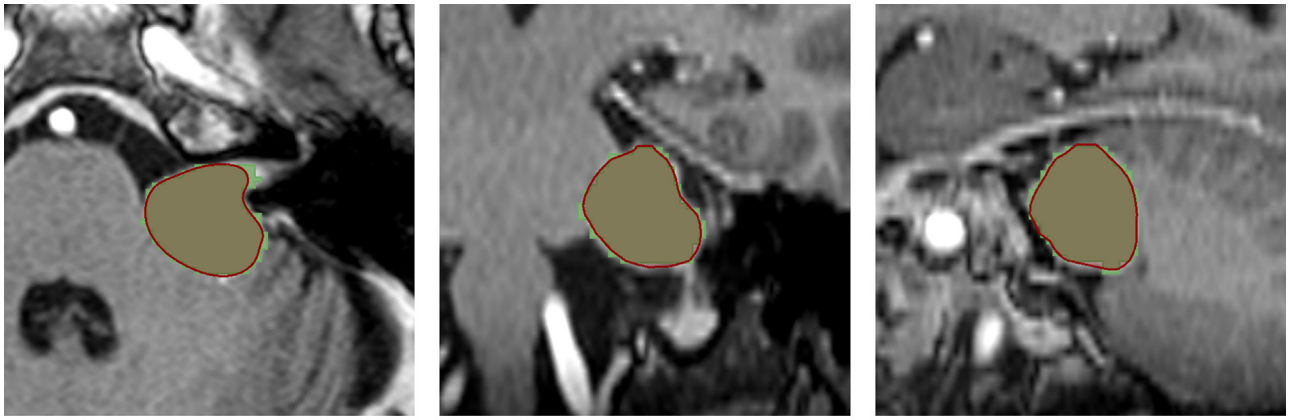
In contrast to the first conversion example, the second demonstrates the conversion from NIfTI to DICOM-RTSS (Fig. 5). Converting NIfTI to DICOM-RTSS requires partially loading the corresponding DICOM image series because DICOM-RTSS contains references to a specific DICOM image series. Therefore, this example also demonstrates the DICOM image series selection mechanism to define the image series referenced in the DICOM-RTSS. Furthermore, this example demonstrates the implementation process of `Extractor` classes required to retrieve the `Modality`, `Organ`, and `Annotator` while loading NIfTI files.

3.2. Data processing

The example *Data Processing* demonstrates selected processing capabilities of the *process* package, including the built-in invertibility feature (Fig. 6). Visualized results are provided for each demonstration in this example to facilitate maximum clearness. In a separate demonstration, building an invertible `FilterPipeline` is illustrated to exemplify a realistic use case in an auto-segmentation pipeline. The given demonstrations also highlight the limitations of the invertibility feature, raising sensitivity that the invertibility feature should be used with appropriate caution. The demonstration of experiencing an information loss after applying the invertibility feature is presented in Fig. 6, where the intensity information at the posterior side of the skull gets removed by transforming the image. The experienced information loss in this demonstration is not retrievable because the invertibility feature does not track the different image stages.

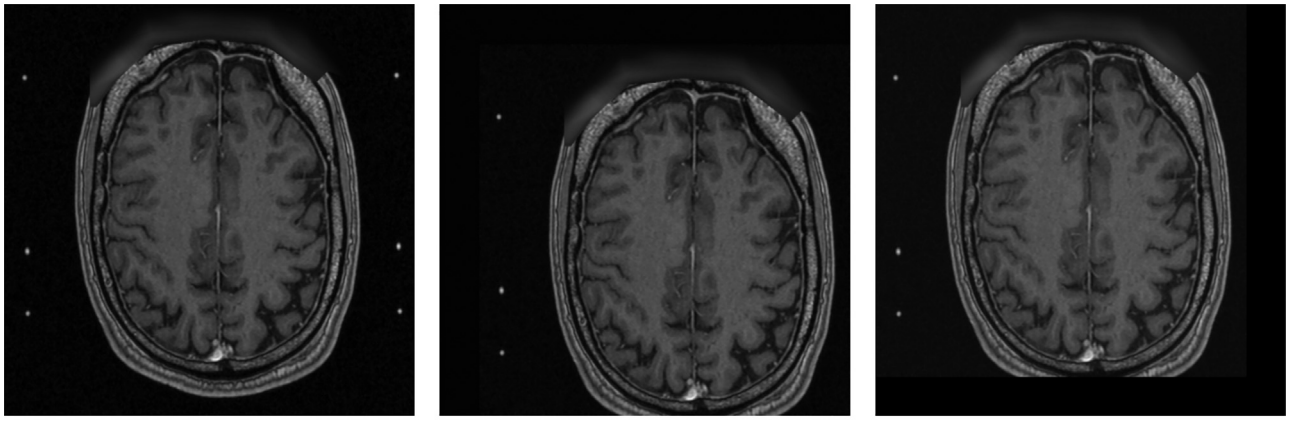
3.3. Inference

The example *DL-Model Inference Pipeline* presents a full-featured auto-segmentation pipeline for segmenting the skull based on a T1-weighted post-contrast (Gd) and a T2-weighted image. Similarly to clinical applications, the example ingests the appropriate DICOM image series and outputs a combination of the utilized DICOM image series and the newly created DICOM-RTSS such that the data is importable in a treatment planning system (Fig. 7). For segmentation, this example relies on a 2D-U-Net implemented in PyTorch that is provided in the example data GitHub repository, as mentioned earlier. In addition, the implementation of the appropriate `InferenceFilter` for applying the model to the subject's images is demonstrated and explained. Furthermore, this example includes specific pre-processing and post-processing steps, such that the image's spatial and intensity value properties correspond with the expected model's input. Finally, this example demonstrates the assignment of customizable metadata to the DICOM-RTSS, such as the institution name, the operator's name, or the series' name.



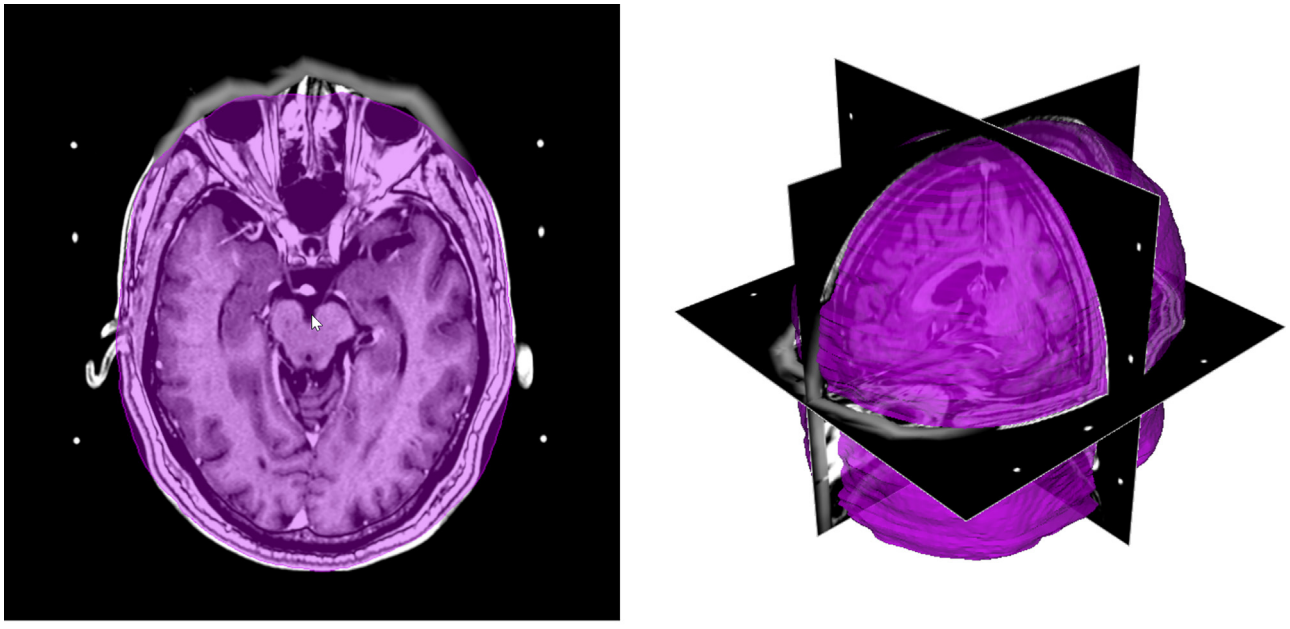
(a) Axial view (b) Coronal view (c) Sagittal view

Fig. 5. Illustration of conversion result (2D-based algorithm) from NIfTI (green-filled structure) to DICOM-RTSS (red-filled structure) with default settings ($\sigma_{smooth} = 2.0$, $kernelsize_{smooth} = 32$) overlaid on the T1-weighted post-contrast (Gd) image of subject VS-SEG-001 from the example dataset, as displayed by 3D Slicer [27].



(a) Original image (b) Resampled image (c) Inverted image

Fig. 6. Illustration of the invertibility feature experiences an information loss on the posterior side of the head. For the resampled image, a transformation via a `ResampleFilter` is applied on VS-SEG-001 of the example dataset.



(a) Axial view (b) 3D reconstruction

Fig. 7. Illustration of the segmentation result generated with the example *DL-Model Inference Pipeline* on subject VS-SEG-001 of the example dataset overlaid on the T1-weighted post-contrast (Gd) image, as displayed in 3D Slicer [27].

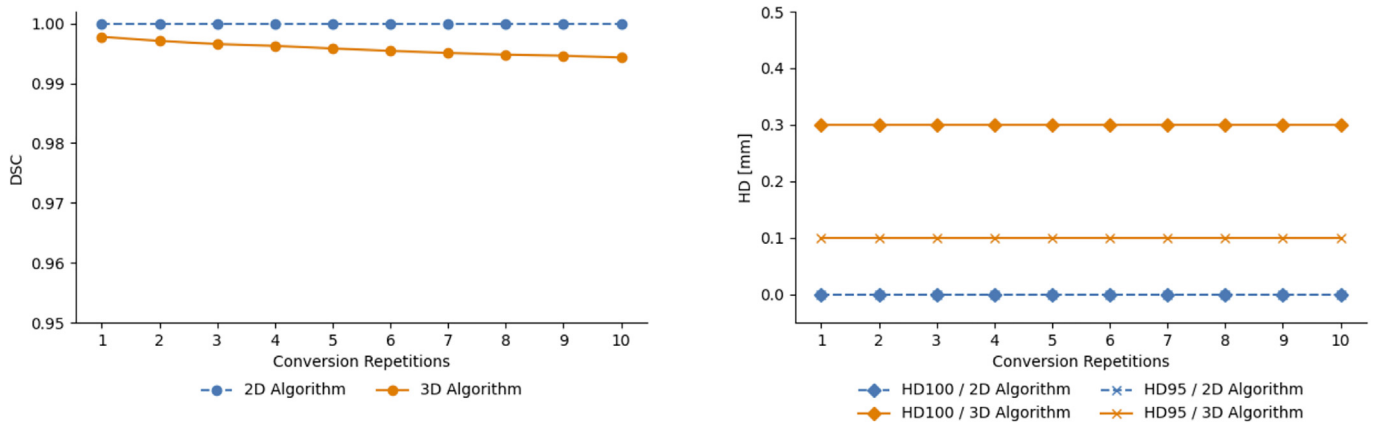
In the introduction, dockerization of auto-segmentation solutions is mentioned as an easy-to-use and versatile deployment option that allows for versioning and minimal setup on the target machine. In the tutorial *Container-based Inference Pipeline*, a recommended workflow for dockerization is suggested that provides detailed descriptions and refers to valuable resources such that typical dockerization challenges can be overwhelmed without frustration.

3.4. Tutorials

In addition to the examples, step-by-step tutorials for converting well-known DICOM-RTSS-based segmentation datasets to discrete medical images are provided on the documentation website to reduce efforts for PyRaDiSe users. These tutorials include metadata handling (i.e., modality configuration file generation) and basic conversion so that users can adapt the conversion scripts to their specific scenarios. Hence, these tutorials also provide a good starting point for building more elaborate conversion pipelines for the respective dataset that, for example, include resampling to a common size and spacing.

3.5. Synthetic conversion experiment

In addition to the aforementioned results, a set of experiments was performed to demonstrate the conversion performance of PyRaDiSe by repeatedly converting different data between DICOM-RTSS and NIfTI. In the first experiment, a synthetic sphere of 20mm diameter was converted for ten cycles using both provided segmentation-to-DICOM-RTSS converters (i.e., `SegmentToRTSSConverter2D` and `SegmentToRTSSConverter3D`). The selection of the sphere as a representative geometrical primitive is motivated by the fact that it contains large curvature in all dimensions that correspond with the geometrical properties of anatomical structures. Furthermore, converting a geometrical body with a high curvature is more challenging than it is for one with a slight curvature. In addition, the sphere diameter has been selected to be similar to many small segmentation structures, such as the eye globes,



(a) Results in DSC.

(b) Results in HD95 and HD100.

Fig. 8. Graphical illustration of the repetition experiment results over 10 repetition cycles.

which are more sensitive to the selected evaluation metrics than structures with larger volumes. For the evaluation of the arising segmentation masks, we chose the Dice coefficient (DSC) [28], the 95th-percentile Hausdorff distance (HD95) [29], and the 100th-percentile Hausdorff distance (HD100) to account for overlap and distance to the ground truth segmentation mask.

For constructing the DICOM-RTSS at each conversion cycle, an artificial DICOM image series was created with a native size of [500 × 500 × 500] voxels and spacing of [0.1 mm × 0.1 mm × 0.1 mm]. For the conversion procedure using the 2D-based conversion algorithm (i.e., `SegmentToRTSSConverter2D`), the default smoothing of the input segmentation mask was disabled such that precise contours are generated that remain unmodified for visualization. Similarly, all smoothing operations for the conversion procedure using the 3D-based conversion algorithm (i.e., `SegmentToRTSSConverter3D`) were disabled.

The obtained quantitative results (Fig. 8) reveal that the 2D-based conversion algorithm produces perfect results that do not experience any quantitative deterioration in any metric used. However, the qualitative checking of the generated DICOM-RTSS showed that contours contain pixelation characteristics (i.e., connected straight line segments from a finite set of angles) due to the omitted smoothing of the segmentation mask before contour extraction. On the other hand, using the 3D-based conversion algorithm resulted in a slight and propagating deterioration of the segmentation masks over multiple conversion cycles (DSC: 0.994, HD95: 0.100 mm, HD100: 0.300 mm after ten conversion cycles). However, over the ten conversion cycles, only the DSC was affected by the propagating deterioration, while the distance-based metrics (i.e., HD95 and HD100) experienced degradation only during the first conversion cycle. Correspondingly, the visual examination of the converted segmentation masks revealed that repetitive conversion with the 3D-based algorithm causes tiny changes located exclusively at the segmentation's boundary. Hence, this demonstrates that discretization and meshing lead to irreversible information loss increasing over multiple conversion cycles. Nonetheless, the degradation of the segmentation masks is slight and slowly increasing, demonstrating for practical applications the high conversion performance of PyRaDiSe's 3D-based conversion algorithm. Detailed results for the 3D-based conversion algorithm are provided in Table 2.

Table 2
Results overview of the synthetic conversion experiment using the 3D-based conversion algorithm.

Metric	Unit	Repetition				
		1	**2**	**3**	**4**	**5**
DSC	–	0.998	0.997	0.997	0.996	0.996
HD95	mm	0.100	0.100	0.100	0.100	0.100
HD100	mm	0.300	0.300	0.300	0.300	0.300
		6	**7**	**8**	**9**	**10**
DSC	–	0.995	0.995	0.995	0.995	0.994
HD95	mm	0.100	0.100	0.100	0.100	0.100
HD100	mm	0.300	0.300	0.300	0.300	0.300

3.6. 2D-based conversion tool comparison

In a second experiment, the 2D-based conversion algorithm of PyRaDiSe was compared with the conversion algorithm of the RT-Utils package that initially provided the basis for the 2D-based conversion algorithm in PyRaDiSe. Similar to the first experiment, the data was converted for ten cycles between DICOM-RTSS and NIfTI, and the segmentation performance was compared using the same segmentation metrics (i.e., DSC, HD95, HD100). However, in contrast to the first experiment, manually generated segmentations of the brain stem, the left cochlea, and the left hippocampus from ten arbitrarily chosen glioblastoma patients, of an in-house DICOM-RTSS dataset, were used to demonstrate the conversion performance on clinical data. These anatomical structures were selected due to their volume and shape diversity.

Because this experiment used raw clinical DICOM data, the selected data contained slight variations in the number of slices per patient but identical slice image sizes of [256 × 256] voxels. Furthermore, patients were selected so that all associated DICOM images have a similar spacing of [1 mm × 1 mm × 1 mm].

The quantitative results (Tables B.3–B.5) obtained with both conversion algorithms demonstrated a near-lossless and identical conversion performance in all selected metrics for the left cochlea and the left hippocampus. However, using the RT-Utils algorithm for converting the brain stem resulted in higher HD100 distances than the PyRaDiSe algorithm. The analysis of the qualitative results unraveled that for one subject, the RT-Utils algorithm generated a single voxel segmentation in the initial conversion from DICOM-RTSS to NIfTI due to a small artifact in the original DICOM-RTSS. Due to its small size, the resulting single voxel segmentation diminished in the subsequent conversion cycles. However, the initial segmentation mask containing the single-voxel segmentation was used as the reference for evaluating the segmentation masks because the computation of the metrics requires discretized segmentation masks. Thus, the segmentation masks generated during the repetitive conversion (those without the single-voxel segmen-

tation) were compared to the initial segmentation mask containing the single-voxel segmentation, which ultimately led to an increase in HD100. The PyRaDiSe conversion algorithm, on the other hand, contains an optional post-processing routine for hole-filling and single-voxel-segmentation-removal that successfully removed the artifact during conversion from DICOM-RTSS to NIfTI. This removal explains the lower HD100 values obtained with the PyRaDiSe algorithm. However, if the mentioned post-processing routine remains inactivated, the increase in HD100 is identical to that experienced with the RT-Utils algorithm.

In order to understand the reason for the minimal deterioration in DSC and HD100, we analyzed the segmentation masks visually. Our analysis showed that segmentation errors, besides the debated case, are exclusively located at the boundary of the segmentation and arise after the first conversion cycle. These errors do not exacerbate beyond the first conversion cycle, indicating an error saturation effect. Furthermore, our observations indicate that the errors arise due to the information loss experienced during the discretization of the continuously-spaced DICOM-RTSS contours. Nonetheless, the conversion performance of both examined conversion algorithms is nearly perfect and allows precise conversion from and to DICOM-RTSS.

3.7. 3D-based conversion tool comparison

Similar to the second experiment comparing two different 2D-based conversion algorithms on clinical data, the third compares two 3D-based conversion algorithms on a subset of the clinical data used in the previous experiment. For this third experiment, the conversion performance of PyRaDiSe's 3D-based algorithm was compared with the performance of the 3D Slicer [27] and its extension for processing DICOM-RTSS called SlicerRT [30]. In contrast to PyRaDiSe, 3D Slicer is a full-featured medical image analysis application intended for interactive use, thus, providing limited API capabilities. Due to this limitation, the ten conversion procedures were executed manually on a reduced set of five glioblastoma patients using the same anatomical structures as in the second experiment.

The 3D Slicer application, combined with the SlicerRT extension, provides capabilities for converting binary segmentation masks directly to DICOM-RTSS or using 3D meshes to convert to DICOM-RTSS. Because we aimed to simulate a 3D-based conversion, we meshed the binary segmentation masks before conversion to DICOM-RTSS. This procedure is similar to the procedure implemented in PyRaDiSe, thus, representing a fair comparison of both algorithms.

The meshing of segmentation masks and the potential subsequent smoothing are sensitive procedures for obtaining high-quality conversion results. Thus PyRaDiSe provides a profound set of settings for controlling these procedures. At the same time, we could not identify such settings for the standard way of meshing in 3D Slicer, which demonstrates a limitation for the comparison. Therefore, we kept the default settings of the PyRaDiSe algorithm for this experiment which may not be optimal for producing the best possible conversion results.

The quantitative results (Tables C.6–C.8) obtained in this experiment showed that the PyRaDiSe algorithm provides better results in each metric and for all anatomical structures when compared to the results obtained with 3D Slicer. However, limitations apply due to missing settings for the 3D Slicer algorithm that may reduce its conversion performance. The qualitative analysis of the segmentation masks yielded that the 3D Slicer application generally produced smoother segmentations. Therefore, the 3D Slicer segmentations lost some shape details still observable in the segmentations generated by PyRaDiSe's algorithm. Furthermore, the 3D Slicer-generated segmentations experience easily observable changes in volume over multiple conversion cycles, which are noticeable as an increase in HD100. These volume changes were predominantly observable in areas of high curvature where smoothing significantly affects the shape accuracy. As a result of the repetitive smoothing, the anatomical shapes will eventually develop into geometric primitives of constant curvature, such as spheres. The same effect with much less magnitude was observable for the PyRaDiSe algorithm, which applies less smoothing on the mesh using default settings. However, changing the smoothing settings will result in the same effect.

In addition, we observed that after three conversion cycles of the left cochlea using the 3D Slicer application, the cochlea's volume collapsed for all five subjects. The PyRaDiSe algorithm, in contrast, preserved the segmentation of the left cochlea for all five subjects and over ten conversion cycles. This volume collapse experienced with 3D Slicer is assumed to be caused by repetitive smoothing leading to a volume reduction.

For some conversions with 3D Slicer, we observed artifacts in the discrete segmentation masks, such as small holes in the middle or partially missing slices. However, these errors propagated only partially due to the meshing and the intense mesh smoothing that removed some of these errors or established connections between partially missing slices to construct a coherent mesh volume. Such errors were not observable with PyRaDiSe.

While PyRaDiSe's algorithm provided a better conversion performance in this experiment, it remains still unclear if 3D Slicer could be competitive when smoothing would be reducible to a comparable amount. However, under current circumstances, PyRaDiSe provides a significantly better conversion performance. Furthermore, the amount of smoothing is controllable for the PyRaDiSe algorithm, so conversion accuracy can be compensated for generating more naturally appearing conversion results.

4. Discussion

We developed PyRaDiSe, a Python package that aims to facilitate the development of automated medical image analysis tools, explicitly targeting the challenges of working with DICOM-RTSS data for radiotherapy treatment. Specifically, the PyRaDiSe package offers a highly customizable set of tools to implement DL framework-independent auto-segmentation solutions for clinical research allowing shorter deployment cycles and easy prototyping. Thus, this new package aims to close the gap between DL research and its clinical research application. Furthermore, our development provides profound methods for data conversion from and to DICOM-RTSS, allowing easy and reproducible construction of curated datasets, for example, for training DL models or other medical image analysis tasks. Therefore, our package includes partially invertible and reproducible processing routines for medical image modification to guarantee consistent results.

The *fileio* package enables flexible, highly automated, and versatile importation and exportation of DICOM image series, DICOM-RTSS, and other discretized medical image formats. Its flexibility manifests in the multitude of methods provided to retrieve pre-loading and metadata, exclude non-required pre-loading information, load a variety of image file formats, and export the data in various ways, including copying the source data to a target directory. Furthermore, the given functionality is highly automated such that managing the complexity of DICOM data is reduced to a minimum, and PyRaDiSe users do not require in-depth DICOM knowledge. However, the modular design of the fileio package also allows versatility and extensibility by providing a well-designed architecture that does not need to compensate for automation due to its design. Therefore, the fileio package integrates smoothly into other frameworks to render DICOM image series and DICOM-RTSS importable and exportable.

The data model provided in the *data* package is specifically designed for the demands of the radiotherapy workflow that commonly works with a multitude of DICOM images and multiple organs and target segmentations. Collecting all images in a joint structure (i.e., a `Subject` instance) allows for handling an arbitrary number of patient-associated images and segmentations while also managing additional user-defined data. In contrast to the standard image types found in other medical imaging libraries, the data package provides distinctive image types for intensity images and segmentation masks, enabling type-specific metadata that is helpful for image identification. Although the data model drastically extends the image types' functionality, it adds a neglectable overhead to the underlying image data, allowing for high memory efficiency and fast data modification. In addition, extending the given image types is achieved easily due to the consideration of polymorphous principles during development so that other image entities, such as DICOM RT Dose images, can be integrated into the data model.

The processing functionality implemented in the *process* package enables the construction of reproducible and partially invertible image analysis pipelines that incorporate pre-processing, DL model inference, and post-processing steps. However, in contrast to other medical imaging libraries, PyRaDiSe allows applying filters (e.g., `ResampleFilter`) at the subject level on all or a subset of the subject-associated images. Hence, it reduces boilerplate code and speeds up the development of new pipelines. Furthermore, the implemented filter design allows for partial inversion depending on the filter's operation. For example, the `ZScoreNormalizationFilter` stores the original intensity value range that can be inverted such that the original image can be restored. However, not all filters are invertible (e.g., `SingleConnectedComponentFilter`), and attempting inversion on non-invertible filters will result in the identity operation with an optional warning to keep the user informed. In addition, the `process` package provides prototype and utility classes for DL framework-independent model inference that allows the implementation of existing DL segmentation models into a PyRaDiSe-based auto-segmentation solution. Therefore, PyRaDiSe users benefit from maximum flexibility and low effort when integrating their model into a deployable segmentation pipeline.

The PyRaDiSe package was successfully used on research projects at our institution for automated segmentation of brain structures in brain tumor patients and expedited algorithm deployment for industrial projects (Fig. 9). For industrial projects, the auto-segmentation tools were made available on the cloud as a Docker container that interacts with an interactive web-based interface. This interface allows clinical research collaborators to upload DICOM image series, predict and correct the generated segmentations, and download the segmentation results as a DICOM-RTSS file. Furthermore, PyRaDiSe was applied in combination with pymia [31] in multiple research projects for end-to-end data curation, data conversion, and HDF5 dataset construction, allowing high-speed data access during DL model training. However, these projects are currently unavailable to the public due to licensing restrictions or pending publication states.

Based on the experience at our institution, we consider the current state of PyRaDiSe as stable and helpful for building auto-segmentation solutions that are deployable to clinical research facilities and can efficiently be operated by clinical research personnel.

Future plans for the PyRaDiSe package include extending the package to other radiotherapy-oriented DICOM Service-Object Pairs (SOPs) classes (e.g., RT Dose Storage), implementing Standardized Uptake Value (SUV) support for Positron Emission Tomogra-

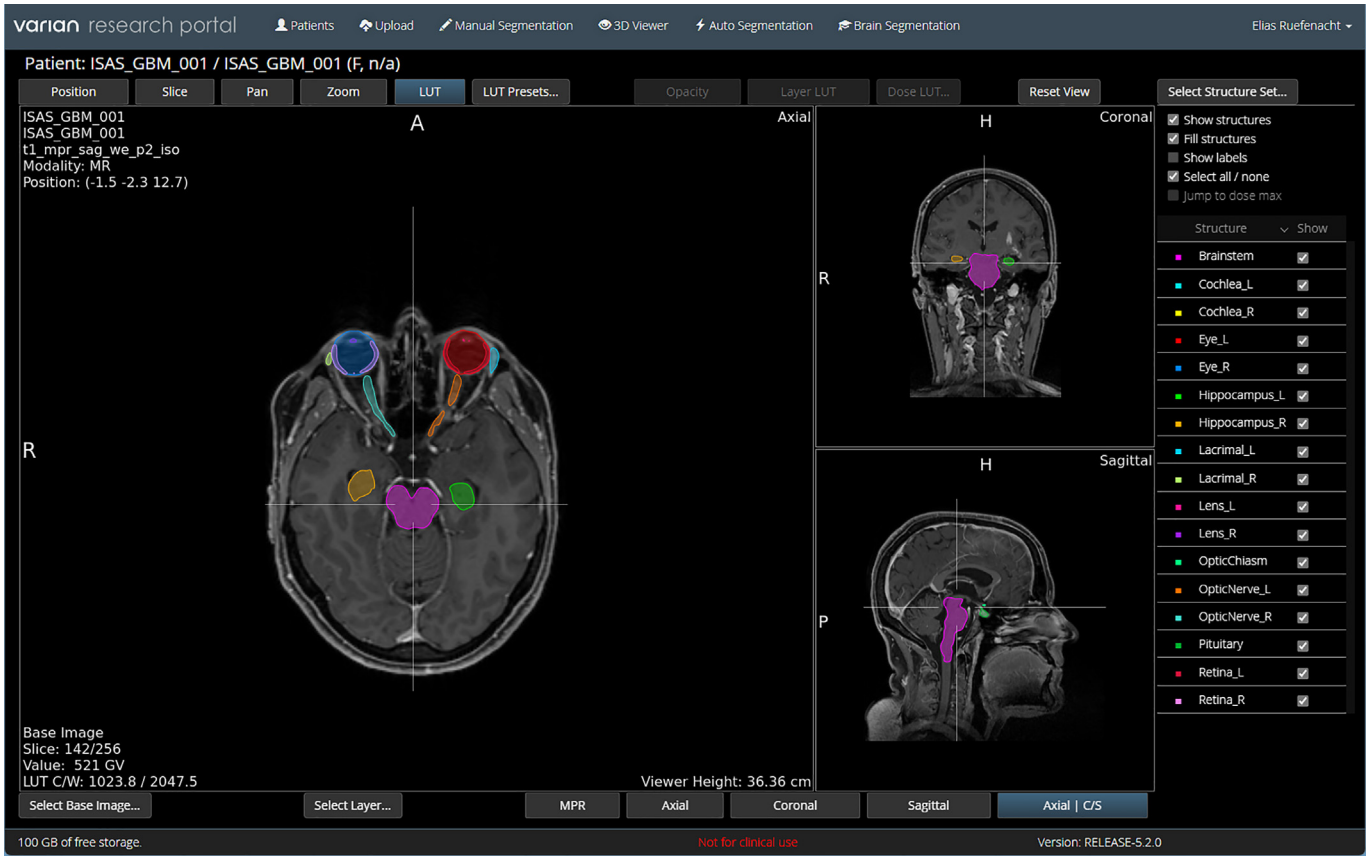


Fig. 9. Web interface of a PyRaDiSe-based auto-segmentation solution deployed on the cloud using Docker containers.

phy (PET) images, extending the set of built-in filters, increasing code coverage by unit tests, reducing memory footprint by profiling memory usage, and extending the examples for other DL frameworks than PyTorch. However, with the expected increase in PyRaDiSe users, feature requests will undoubtedly emerge, but we aim to keep simplicity and modularity in mind for future releases. For instance, it would be beyond the scope of this package to implement neural network segmentation architectures, and training functionality as projects like MONAI do. However, extending the set of built-in filters would be valuable for the intended use of PyRaDiSe.

In conclusion, PyRaDiSe was developed to close the gap between medical image analysis and clinical radiotherapy research by speeding up the transition process and making the deployment of auto-segmentation solutions straightforward. Furthermore, PyRaDiSe facilitates fast and straightforward curation of DICOM-RTSS data for DL dataset establishment.

Statements of ethical approval

Not applicable.

Declaration of Competing Interest

Authors declare that they have no conflict of interest.

Acknowledgments

The authors thank all the contributors to PyRaDiSe and acknowledge the valuable feedback from Fabian Balsiger and Alain Jungo. This research was partially supported by the Swiss Innovation Agency (InnoSuisse) under the project number 31274.1 IP-LS.

Appendix A. Example modality configuration file

Listings 3–4.

```
[
   {
        "SOPClassUID": "1.2.840.10008.5.1.4.1.1.4",
        "StudyInstanceUID": "1.3.6.1.4.1.1556635.6.0",
        "SeriesInstanceUID": "1.3.6.1.4.1.155761.29.0",
        "SeriesDescription": "t1_mpr_sag_we_p2_iso",
        "SeriesNumber": "7",
        "DICOM_Modality": "MR",
        "Modality": "MR"
  },
  {
        "SOPClassUID": "1.2.840.10008.5.1.4.1.1.2",
        "StudyInstanceUID": "1.3.6.1.4.1.15576.105.0",
        "SeriesInstanceUID": "1.3.6.1.4.1.15346.129.0",
        "SeriesDescription": "t2_fl_sag_p2_iso",
        "SeriesNumber": "2",
        "DICOM_Modality": "MR",
        "Modality": "MR"
   }
]
```

Listing 3. Modality configuration file skeleton generated by the `DicomCrawler`.

```
[
   {
        "SOPClassUID": "1.2.840.10008.5.1.4.1.1.4",
        "StudyInstanceUID": "1.3.6.1.4.1.1556635.6.0",
        "SeriesInstanceUID": "1.3.6.1.4.1.155761.29.0",
        "SeriesDescription": "t1_mpr_sag_we_p2_iso",
        "SeriesNumber": "7",
        "DICOM_Modality": "MR",
        "Modality": "T1c" <-- ADJUSTED MODALITY
  },
  {
        "SOPClassUID": "1.2.840.10008.5.1.4.1.1.2",
        "StudyInstanceUID": "1.3.6.1.4.1.15576.105.0",
        "SeriesInstanceUID": "1.3.6.1.4.1.15346.129.0",
        "SeriesDescription": "t2_fl_sag_p2_iso",
        "SeriesNumber": "2",
        "DICOM_Modality": "MR",
        "Modality": "FLAIR" <-- ADJUSTED MODALITY
   }
]
```

Listing 4. Manually modified modality configuration file ready for loading the data.

Appendix B. Results 2D-based conversion tool comparison

B1. Segmentation performance results in DSC

Table B.3
Results in DSC for the conversion tool comparison using the 2D-based conversion algorithms of PyRaDiSe and RT-Utils.

Structure	Algorithm	Repetition / mean & SD in DSC ($n = 10$)				
		1	**2**	**3**	**4**	**5**
Brainstem	**RT-Utils**	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
Brainstem	**PyRaDiSe 2D**	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
Cochlea left	**RT-Utils**	0.998 ± 0.003	0.998 ± 0.003	0.998 ± 0.003	0.998 ± 0.003	0.998 ± 0.003
Cochlea left	**PyRaDiSe 2D**	0.998 ± 0.003	0.998 ± 0.003	0.998 ± 0.003	0.998 ± 0.003	0.998 ± 0.003
Hippocampus left	**RT-Utils**	1.000 ± 0.001	1.000 ± 0.001	1.000 ± 0.001	1.000 ± 0.001	1.000 ± 0.001
Hippocampus left	**PyRaDiSe 2D**	1.000 ± 0.001	1.000 ± 0.001	1.000 ± 0.001	1.000 ± 0.001	1.000 ± 0.001
		6	**7**	**8**	**9**	**10**
Brainstem	**RT-Utils**	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
Brainstem	**PyRaDiSe 2D**	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
Cochlea left	**RT-Utils**	0.998 ± 0.003	0.998 ± 0.003	0.998 ± 0.003	0.998 ± 0.003	0.998 ± 0.003
Cochlea left	**PyRaDiSe 2D**	0.998 ± 0.003	0.998 ± 0.003	0.998 ± 0.003	0.998 ± 0.003	0.998 ± 0.003
Hippocampus left	**RT-Utils**	1.000 ± 0.001	1.000 ± 0.001	1.000 ± 0.001	1.000 ± 0.001	1.000 ± 0.001
Hippocampus left	**PyRaDiSe 2D**	1.000 ± 0.001	1.000 ± 0.001	1.000 ± 0.001	1.000 ± 0.001	1.000 ± 0.001

B2. Segmentation performance results in HD95

Table B.4
Results in HD95 for the conversion tool comparison using the 2D-based conversion algorithms of PyRaDiSe and RT-Utils.

Structure	Algorithm	Repetition / mean & SD in HD95 ($n = 10$)				
		1	**2**	**3**	**4**	**5**
Brainstem	**RT-Utils**	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Brainstem	**PyRaDiSe 2D**	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Cochlea left	**RT-Utils**	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Cochlea left	**PyRaDiSe 2D**	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Hippocampus left	**RT-Utils**	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Hippocampus left	**PyRaDiSe 2D**	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
		6	**7**	**8**	**9**	**10**
Brainstem	**RT-Utils**	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Brainstem	**PyRaDiSe 2D**	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Cochlea left	**RT-Utils**	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Cochlea left	**PyRaDiSe 2D**	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Hippocampus left	**RT-Utils**	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Hippocampus left	**PyRaDiSe 2D**	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000

B3. Segmentation performance results in HD100

Table B.5
Results in HD100 for the conversion tool comparison using the 2D-based conversion algorithms of PyRaDiSe and RT-Utils.

Structure	Algorithm	Repetition / mean & SD in HD100 ($n = 10$)				
		1	**2**	**3**	**4**	**5**
Brainstem	**RT-Utils**	1.471 ± 1.771	1.471 ± 1.771	1.471 ± 1.771	1.471 ± 1.771	1.471 ± 1.771
Brainstem	**PyRaDiSe 2D**	0.900 ± 0.300	0.900 ± 0.300	0.900 ± 0.300	0.900 ± 0.300	0.900 ± 0.300
Cochlea left	**RT-Utils**	0.400 ± 0.490	0.400 ± 0.490	0.400 ± 0.490	0.400 ± 0.490	0.400 ± 0.490
Cochlea left	**PyRaDiSe 2D**	0.400 ± 0.490	0.400 ± 0.490	0.400 ± 0.490	0.400 ± 0.490	0.400 ± 0.490
Hippocampus left	**RT-Utils**	0.600 ± 0.490	0.600 ± 0.490	0.600 ± 0.490	0.600 ± 0.490	0.600 ± 0.490
Hippocampus left	**PyRaDiSe 2D**	0.600 ± 0.490	0.600 ± 0.490	0.600 ± 0.490	0.600 ± 0.490	0.600 ± 0.490
		6	**7**	**8**	**9**	**10**
Brainstem	**RT-Utils**	1.471 ± 1.771	1.471 ± 1.771	1.471 ± 1.771	1.471 ± 1.771	1.471 ± 1.771
Brainstem	**PyRaDiSe 2D**	0.900 ± 0.300	0.900 ± 0.300	0.900 ± 0.300	0.900 ± 0.300	0.900 ± 0.300
Cochlea left	**RT-Utils**	0.400 ± 0.490	0.400 ± 0.490	0.400 ± 0.490	0.400 ± 0.490	0.400 ± 0.490
Cochlea left	**PyRaDiSe 2D**	0.400 ± 0.490	0.400 ± 0.490	0.400 ± 0.490	0.400 ± 0.490	0.400 ± 0.490
Hippocampus left	**RT-Utils**	0.600 ± 0.490	0.600 ± 0.490	0.600 ± 0.490	0.600 ± 0.490	0.600 ± 0.490
Hippocampus left	**PyRaDiSe 2D**	0.600 ± 0.490	0.600 ± 0.490	0.600 ± 0.490	0.600 ± 0.490	0.600 ± 0.490

Appendix C. Results 3D-based conversion tool comparison

C1. Segmentation performance results in DSC

Table C.6
Results in DSC for the conversion tool comparison using the 3D-based conversion algorithms of PyRaDiSe and 3D Slicer.

Structure	Algorithm	Repetition / mean & SD in DSC ($n = 5$)				
		1	**2**	**3**	**4**	**5**
Brainstem	**3D Slicer**	0.983 ± 0.002	0.976 ± 0.003	0.971 ± 0.004	0.968 ± 0.004	0.965 ± 0.004
Brainstem	**PyRaDiSe 3D**	0.994 ± 0.000	0.991 ± 0.001	0.989 ± 0.001	0.987 ± 0.001	0.985 ± 0.001
Cochlea left	**3D Slicer**	0.728 ± 0.033	0.365 ± 0.128	0.122 ± 0.116	n/a	n/a
Cochlea left	**PyRaDiSe 3D**	0.982 ± 0.002	0.973 ± 0.004	0.969 ± 0.007	0.967 ± 0.010	0.966 ± 0.012
Hippocampus left	**3D Slicer**	0.946 ± 0.003	0.923 ± 0.007	0.902 ± 0.009	0.884 ± 0.012	0.868 ± 0.017
Hippocampus left	**PyRaDiSe 3D**	0.987 ± 0.001	0.979 ± 0.002	0.973 ± 0.003	0.968 ± 0.003	0.964 ± 0.004
		6	**7**	**8**	**9**	**10**
Brainstem	**3D Slicer**	0.963 ± 0.004	0.962 ± 0.004	0.959 ± 0.005	0.956 ± 0.005	0.955 ± 0.005
Brainstem	**PyRaDiSe 3D**	0.984 ± 0.002	0.983 ± 0.002	0.982 ± 0.002	0.981 ± 0.002	0.980 ± 0.002
Cochlea left	**3D Slicer**	n/a	n/a	n/a	n/a	n/a
Cochlea left	**PyRaDiSe 3D**	0.965 ± 0.013	0.964 ± 0.015	0.964 ± 0.015	0.964 ± 0.015	0.964 ± 0.015
Hippocampus left	**3D Slicer**	0.854 ± 0.021	0.839 ± 0.026	0.826 ± 0.031	0.813 ± 0.036	0.801 ± 0.039
Hippocampus left	**PyRaDiSe 3D**	0.961 ± 0.005	0.958 ± 0.006	0.955 ± 0.007	0.953 ± 0.008	0.951 ± 0.009

C2. Segmentation performance results in HD95

Table C.7
Results in HD95 for the conversion tool comparison using the 3D-based conversion algorithms of PyRaDiSe and 3D Slicer.

Structure	Algorithm	Repetition / mean & SD in HD95 ($n = 5$)				
		1	**2**	**3**	**4**	**5**
Brainstem	**3D Slicer**	1.000 ± 0.000	1.146 ± 0.293	1.247 ± 0.494	1.449 ± 0.708	1.290 ± 0.580
Brainstem	**PyRaDiSe 3D**	0.000 ± 0.000	0.000 ± 0.000	0.600 ± 0.490	1.000 ± 0.000	1.000 ± 0.000
Cochlea left	**3D Slicer**	1.000 ± 0.000	1.839 ± 0.131	3.018 ± 0.472	n/a	n/a
Cochlea left	**PyRaDiSe 3D**	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.200 ± 0.400	0.200 ± 0.400
Hippocampus left	**3D Slicer**	1.000 ± 0.000	1.000 ± 0.000	1.166 ± 0.203	1.712 ± 0.262	2.142 ± 0.116
Hippocampus left	**PyRaDiSe 3D**	0.000 ± 0.000	0.000 ± 0.000	0.400 ± 0.490	1.000 ± 0.000	1.000 ± 0.000
		6	**7**	**8**	**9**	**10**
Brainstem	**3D Slicer**	1.083 ± 0.166	1.000 ± 0.000	1.903 ± 1.606	2.832 ± 2.093	2.832 ± 2.093
Brainstem	**PyRaDiSe 3D**	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
Cochlea left	**3D Slicer**	n/a	n/a	n/a	n/a	n/a
Cochlea left	**PyRaDiSe 3D**	0.200 ± 0.400	0.200 ± 0.400	0.200 ± 0.400	0.200 ± 0.400	0.200 ± 0.400
Hippocampus left	**3D Slicer**	2.426 ± 0.340	2.845 ± 0.322	3.238 ± 0.560	3.634 ± 0.443	4.053 ± 0.614
Hippocampus left	**PyRaDiSe 3D**	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000

C3. Segmentation performance results in HD100

Table C.8
Results in HD100 for the conversion tool comparison using the 3D-based conversion algorithms of PyRaDiSe and 3D Slicer.

Structure	Algorithm	Repetition / mean & SD in HD100 ($n = 5$)				
		1	**2**	**3**	**4**	**5**
Brainstem	**3D Slicer**	5.633 ± 1.635	5.482 ± 1.464	7.360 ± 1.904	7.533 ± 1.859	8.442 ± 1.155
Brainstem	**PyRaDiSe 3D**	1.312 ± 0.280	1.776 ± 0.217	2.094 ± 0.116	2.184 ± 0.169	2.232 ± 0.142
Cochlea left	**3D Slicer**	1.579 ± 0.329	2.474 ± 0.280	3.843 ± 0.477	n/a	n/a
Cochlea left	**PyRaDiSe 3D**	1.000 ± 0.000	1.000 ± 0.000	1.083 ± 0.166	1.083 ± 0.166	1.083 ± 0.166
Hippocampus left	**3D Slicer**	1.903 ± 0.424	3.133 ± 0.885	4.017 ± 1.125	4.964 ± 1.167	5.735 ± 1.054
Hippocampus left	**PyRaDiSe 3D**	1.000 ± 0.000	1.000 ± 0.000	1.166 ± 0.203	1.331 ± 0.166	1.541 ± 0.156
		6	**7**	**8**	**9**	**10**
Brainstem	**3D Slicer**	8.591 ± 1.093	8.237 ± 0.870	7.804 ± 1.581	7.295 ± 2.716	7.316 ± 2.735
Brainstem	**PyRaDiSe 3D**	2.232 ± 0.142	2.474 ± 0.280	2.625 ± 0.330	2.770 ± 0.360	2.858 ± 0.480
Cochlea left	**3D Slicer**	n/a	n/a	n/a	n/a	n/a
Cochlea left	**PyRaDiSe 3D**	1.083 ± 0.166	1.083 ± 0.166	1.083 ± 0.166	1.083 ± 0.166	1.083 ± 0.166
Hippocampus left	**3D Slicer**	6.620 ± 1.085	7.417 ± 0.997	8.136 ± 0.901	8.826 ± 0.840	9.533 ± 0.722
Hippocampus left	**PyRaDiSe 3D**	1.696 ± 0.353	1.876 ± 0.281	1.876 ± 0.281	1.876 ± 0.281	1.876 ± 0.281

References

[1] M. Hussein, B. Heijmen, D. Verellen, A. Nisbet, Automation in intensity modulated radiotherapy treatment planning—A review of recent innovations, Br. J. Radiol. 91 (1092) (2018), doi:10.1259/bjr.20180270.

[2] I. Valduvieco, E. Verger, J. Bruna, L. Caral, T. Pujol, T. Ribalta, T. Boget, L. Oleaga, E. Pineda, F. Graus, Impact of radiotherapy delay on survival in glioblastoma, Clin. Transl. Oncol. 15 (4) (2013) 278–282, doi:10.1007/s12094-012-0916-x.

[3] Z. Chen, W. King, R. Pearcey, M. Kerba, W.J. Mackillop, The relationship between waiting time for radiotherapy and clinical outcomes: a systematic review of the literature, Radiother. Oncol. 87 (2008) 3–16, doi:10.1016/j.radonc.2007.11.016.

[4] P. Meyer, V. Noblet, C. Mazzara, A. Lallement, Survey on deep learning for radiotherapy, Comput. Biol. Med. 98 (2018) 126–146, doi:10.1016/j.compbiomed.2018.05.018.

[5] G. Litjens, T. Kooi, B.E. Bejnordi, A.A.A. Setio, F. Ciompi, M. Ghafoorian, J. van der Laak, B. van Ginneken, C.I. Sanchez, A survey on deep learning in medical image analysis, Med. Image Anal. 42 (2017) 60–88, doi:10.1016/j.media.2017.07.005.

[6] A.S. Lundervold, A. Lundervold, An overview of deep learning in medical imaging focusing on MRI, Z. Med. Phys. 29 (2) (2019) 102–127, doi:10.1016/j.zemedi.2018.11.002.

[7] S. Nikolov, S. Blackwell, A. Zverovitch, R. Mendes, M. Livne, J. De Fauw, Y. Patel, C. Meyer, H. Askham, B. Romera-Paredes, C. Kelly, A. Karthikesalingam, C. Chu, D. Carnell, C. Boon, D. De Souza, S.A. Moinuddin, B. Garie, Y. McQuinlan, S. Ireland, K. Hampton, K. Fuller, H. Montgomery, G. Rees, M. Suleyman, T. Back, C. Hughes, J.R. Ledsam, O. Ronneberger, Deep learning to achieve clinically applicable segmentation of head and neck anatomy for radiotherapy, (2021). arXiv.1809.04430.

[8] E. Cha, S. Elguindi, I. Onochie, D. Gorovets, J.O. Deasy, M. Zelefsky, E.F. Gillespie, Clinical implementation of deep learning contour autosegmentation for prostate radiotherapy, Radiother. Oncol. 159 (2021) 1–7, doi:10.1016/j.radonc.2021.02.040.

[9] E. Ermis, A. Jungo, R. Poel, M. Blatti-Moreno, R. Meier, U. Knecht, D.M. Aebersold, M.K. Fix, P. Manser, M. Reyes, E. Herrmann, Fully automated brain resection cavity delineation for radiation target volume definition in glioblastoma patients using deep learning, Radiat. Oncol. 15 (1) (2020) 100, doi:10.1186/s13014-020-01553-z.

[10] M. Havaei, A. Davy, D. Warde-Farley, A. Biard, A. Courville, Y. Bengio, C. Pal, P.-M. Jodoin, H. Larochelle, Brain tumor segmentation with deep neural networks, Med. Image Anal. 35 (2017) 18–31, doi:10.1016/j.media.2016.05.004.

[11] NEMA PS3 / ISO 12052, Digital imaging and communications in medicine (DICOM) standard, National Electrical Manufacturers Association, Rosslyn, VA, USA, 2022. http://www.dicomstandard.org/

[12] O. Ronneberger, P. Fischer, T. Brox, U-Net: convolutional networks for biomedical image segmentation, in: Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015, in: Lecture Notes in Computer Science, Springer International Publishing, 2015, pp. 234–241, doi:10.1007/978-3-319-24574-4_28.

[13] A. Shrestha, RT-Utils: a minimal Python library to facilitate the creation and manipulation of DICOM RTStructs, 2022, (GitHub). https://github.com/qurit/rt-utils.

[14] B.M. Anderson, K.A. Wahid, K.K. Brock, Simple Python module for conversions between DICOM images and radiation therapy structures, masks, and prediction arrays, Pract. Radiat. Oncol. 11 (3) (2021) 226–229, doi:10.1016/j.prro.2021.02.003.

[15] M. McCormick, X. Liu, L. Ibanez, J. Jomier, C. Marion, ITK: enabling reproducible research and open science, Front. Neuroinform. 8 (2014), doi:10.3389/fninf.2014.00013.

[16] B. Lowekamp, D. Chen, L. Ibanez, D. Blezek, The design of SimpleITK, Front. Neuroinform. 7 (2013), doi:10.3389/fninf.2013.00045.

[17] F. Perez-Garcia, R. Sparks, S. Ourselin, TorchIO: a Python library for efficient loading, preprocessing, augmentation and patch-based sampling of medical images in deep learning, Comput. Methods Programs Biomed. 208 (2021) 106236, doi:10.1016/j.cmpb.2021.106236.

[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: an imperative style, high-performance deep learning library, in: Proceedings of the 33rd International Conference on Neural Information Processing Systems, Curran Associates Inc., Red Hook, NY, USA, 2019, pp. 1–12, doi:10.5555/3454287.3455008.

[19] The MONAI Consortium, Project MONAI, Zenodo (2020), doi:10.5281/zenodo.4323059.

[20] A. Beers, J. Brown, K. Chang, K. Hoebel, J. Patel, K.I. Ly, S.M. Tolaney, P. Brastianos, B. Rosen, E.R. Gerstner, J. Kalpathy-Cramer, DeepNeuro: an open-source deep learning toolbox for neuroimaging, Neuroinformatics 19 (1) (2021) 127–140, doi:10.1007/s12021-020-09477-5.

[21] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: large-scale machine learning on heterogeneous distributed systems, (2016). arXiv.1603.04467.

[22] D. Merkel, Docker: lightweight linux containers for consistent development and deployment, Linux J. 2014 (239) (2014), doi:10.5555/2600239.2600241.

[23] K. Marstal, F. Berendsen, M. Staring, S. Klein, SimpleElastix: a user-friendly, multi-lingual library for medical image registration, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2016, doi:10.1109/CVPRW.2016.78.

[24] K. Clark, B. Vendt, K. Smith, J. Freymann, J. Kirby, P. Koppel, S. Moore, S. Phillips, D. Maffitt, M. Pringle, L. Tarbox, F. Prior, The cancer imaging archive (TCIA): maintaining and operating a public information repository, J. Digit. Imaging 26 (6) (2013) 1045–1057, doi:10.1007/s10278-013-9622-7.

[25] J. Shapey, A. Kujawa, R. Dorent, G. Wang, S. Bisdas, A. Dimitriadis, D. Grishchuck, I. Paddick, N. Kitchen, R. Bradford, S. Saeed, S. Ourselin, T. Vercauteren, Segmentation of vestibular schwannoma from magnetic resonance imaging: an open annotated dataset and baseline algorithm [data set], Cancer Imaging Arch. (2021), doi:10.7937/TCIA.9YTJ-5Q73.

[26] J. Shapey, A. Kujawa, R. Dorent, G. Wang, S. Bisdas, A. Dimitriadis, D. Grishchuck, I. Paddick, N. Kitchen, R. Bradford, S. Saeed, S. Ourselin, T. Vercauteren, Segmentation of vestibular schwannoma from MRI, an open annotated dataset and baseline algorithm, Sci. Data 8 (1) (2021), doi:10.1038/s41597-021-01064-w.

[27] A. Fedorov, R. Beichel, J. Kalpathy-Cramer, J. Finet, J.-C. Fillion-Robin, S. Pujol, C. Bauer, D. Jennings, F. Fennessy, M. Sonka, J. Buatti, S. Aylward, J.V. Miller, S. Pieper, R. Kikinis, 3D slicer as an image computing platform for the quantitative imaging network, Magn. Reson. Imaging 30 (9) (2012) 1323–1341, doi:10.1016/j.mri.2012.05.001.

[28] L.R. Dice, Measures of the amount of ecologic association between species, Ecology 26 (3) (1945) 297–302, doi:10.2307/1932409.

[29] D.P. Huttenlocher, G.A. Klanderman, W.J. Rucklidge, Comparing images using the Hausdorff distance, IEEE Trans. Pattern Anal. Mach. Intell. 15 (9) (1993) 850–863.

[30] C. Pinter, A. Lasso, A. Wang, D. Jaffray, G. Fichtinger, SlicerRT: radiation therapy research toolkit for 3D slicer, J. Med. Phys. 39 (10) (2012) 6332–6338, doi:10.1118/1.4754659.

[31] A. Jungo, O. Scheidegger, M. Reyes, F. Balsiger, Pymia: a Python package for data handling and evaluation in deep learning-based medical image analysis, Comput. Methods Programs Biomed. 198 (2021), doi:10.1016/j.cmpb.2020.105796.