Petr Strakoš Ph.D

Best Practice Guide

Medical image processing as a service

Best Practice Guide

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

Medical image processing as a service

Petr Strakoš Ph.D

Best Practice Guide

Contents:

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

List of abbreviations:

MRI	Magnetic Resonance Imaging
CT	Computed Tomography
DL	Deep Learning
GPU	Graphics Processing Unit
AI	Artificial Intelligence
HPC	High Performance Computing
SDK	Software Development Kit
SSH	Secure Shell
AIAA	AI-Assisted Annotation
MONAI	Medical Open Network for AI
MMAR	Medical Model Archive
BYOC	Bring Your Components
HTTP	Hypertext Transfer Protocol
VTK	Visualization Toolkit
ITK	Insight Toolkit
CTK	The Common Toolkit
CUDA	Compute Unified Device Architecture
DICOM	Digital Imaging and Communications in Medicine
FDA	Food and Drug Administration

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

1. Basic idea

Medical image processing can help to find and understand irregularities in the human body. It can detect or even predict diseases. As a data source, Magnetic Resonance Imaging (MRI) or Computed Tomography (CT) is often used. The current state-of-the-art in medical image processing and analysis is in machine learning and, more specifically, in deep learning (DL) and the use of deep neural networks [1,2,3,4]. DL methods can reduce the time to high-quality medical diagnosis and thus improve healthcare in general.
Since deep learning approaches hold the state-of-the-art in medical image processing and analysis, the aim was to develop a user-friendly solution for medical doctors that would allow to use the DL algorithms and provide an automatic tissue segmentation from medical images. Before any capable DL algorithm is trained, a large amount of data and computing power is needed. This is especially true if you train the models from scratch. Such models, e.g., for performing the 3D image segmentation, usually combine many labelled datasets with training on multiple GPUs to provide models of desired quality and in a reasonable time. Therefore, the idea in this project was (i) to provide a service that would use the state-of-the-art algorithms for automatic segmentation of desired tissues as an AI-based annotation service and (ii) collect the data after the automatic segmentation and validation by medical doctors and provide an HPC based training of new models or enhance the existing models by fine-tunning them and use the new or improved models again in the step (i). In such a way a complex, but a useful and user-friendly concept for medical image processing and analysis is being created.

2. Our approach

The solution that we have created consists of several building blocks that provide the desired functionality. The whole idea is depicted in Fig. 1. Two main parts can be distinguished there. One runs at a medical doctor's site in a local hospital, and the other operates in our facility at IT4Innovations National Supercomputing Center.

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

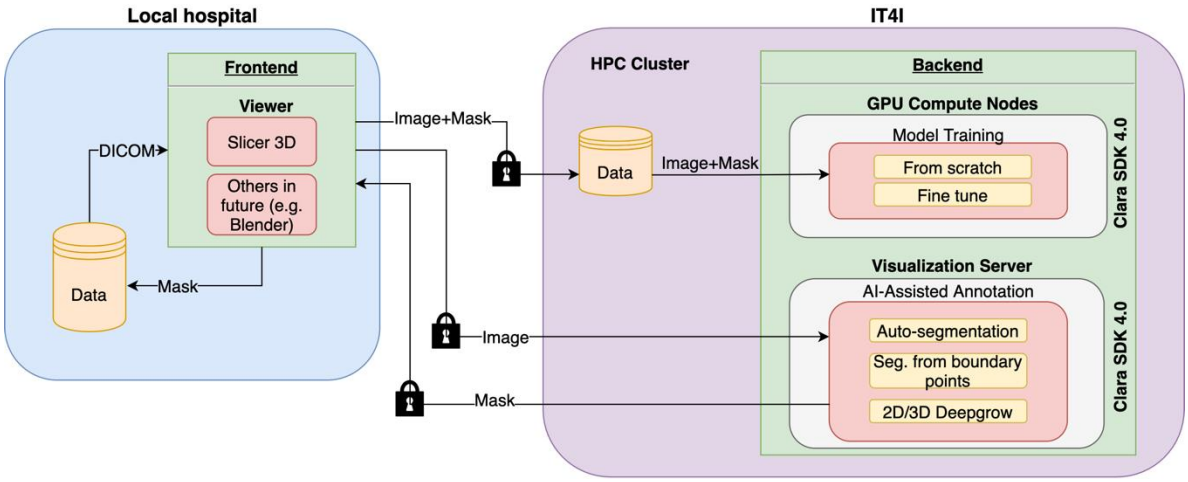

Fig. 1 – The main concept of the tool for medical image processing and analysis

The part at the local hospital is represented by a frontend that mediates the interaction between the doctor and the data. A software representing this part should allow the user to load the data, view them in a raw form and provide access to the computational resources that will process the images and send the results back. The hardware equipment the viewer runs on can therefore be lightweight since all the computation is provided remotely. For medical doctors, such as radiologists, this tool must have a similar environment to the one they are used to in their regular practice. We have chosen an open-source 3D Slicer [5] to serve as the viewer with expected capability and to develop the necessary extension for accessing the backend part. More details regarding the viewer and its modifications needed for the communication with the backend part are described in section 3.2.

The backend part that provides the computational power and other required features is held at IT4Innovations National Supercomputing Center. The main idea here is to allow training of deep learning (DL) models for automatic tissue segmentation and then provide easy use of the models for DL inference. This twofold nature led us to the utilization of various High-Performance Computing (HPC) resources. As shown in Fig. 1, we use GPU compute nodes for training and a GPU based visualization server to provide model inference using AI-Assisted annotation. The reason for this is that the training process is a computationally intensive task that can utilize multiple resources for a long period of time. It can take from hours to days, depending on the complexity of the trained model and the amount of the training data. Although the training is an intensive process, it will occur rather occasionally only if a new set of training data is collected. This is an ideal task for the utilization of cluster resources in the form of multi-GPU nodes.

On the other hand, the model inference does not require high computing power; it can be provided in a short time (a matter of minutes) by just a single GPU. The computational resources for the model inference should be quickly available. Therefore, we dedicate this task to the visualization server. Within the IT4Innovation's infrastructure, several visualization

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

servers are sparsely used; therefore, such resource is ideal for being used as an on-demand inference server. The computational part of the concept responsible for model training and inference should be compatible with both types of resources. A powerful and general tool to facilitate this task has been found in NVIDIA Clara Train SDK [6]. A more detailed description of the tool is provided in section 3.1.
Regarding the data sent between the hospital and IT4Innovations, only anonymized image data in NIFTI format [7] and binary label masks are used. Apart from that, all the connections are secured by secure encrypted communication (SSH).

3. Building blocks

3.1. NVIDIA Clara Train SDK

We have adopted an available tool of NVIDIA's Clara Train SDK [6]. It contains several APIs, such as those for AI-Assisted Annotation (AIAA) and a training framework denoted as Clara Training Framework for the DL-based model training. The Clara Train SDK is in version 4.0, and it is solely PyTorch-based compared to a previous version which was TensorFlow based. The Clara Training Framework uses an open-source Medical Open Network for AI (MONAI) framework [8]. MONAI has been specifically denoted to deep learning in healthcare imaging. Therefore, besides the standard Clara tools proven from previous versions, such as Medical Model Archive (MMAR) and Bring Your Components (BYOC), Clara combines these with the components of MONAI.
As for the MMAR, this concept helps the user organize and configure models and data and covers the whole development cycle of the model. Several MMARs have been made available to give developers a reference. These have also been used in our concept, and we have extended their capabilities to better match the needs of medical doctors.
The BYOC is a way to write your components in Python language and then use them in the training/inference pipeline. Components from MONAI can also be used here as templates and then serve within BYOC.

3.1.1. AI-Assisted Annotation

Clara's AI-Assisted Annotation (AIAA) is a client-server-based architecture that provides C++ or Python client API. In this way, many medical image viewers can be interfaced with to provide AIAA services offered by the server. Based on the pre-loaded models on the server, the AIAA can make automatic segmentation of specific tissues and share the results with the interfaced viewer on the client's side. It can significantly reduce medical doctors' workload. If

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

combined with the viewer's capabilities in terms of editing and post-processing the resulting masks, it can speed up the creation of high-quality annotated datasets that can be used to train robust DL algorithms.
Since this solution is given to researchers and developers by NVIDIA, the prerequisite to use it is to have an NVIDIA GPU card available.
A schematic view of the AIAA's architecture is in Fig. 2. Communication between the server and client runs over the HTTP protocol. To add the necessary security level in the communication between the client and the server, we encapsulate the connection within secure encrypted communication (SSH).

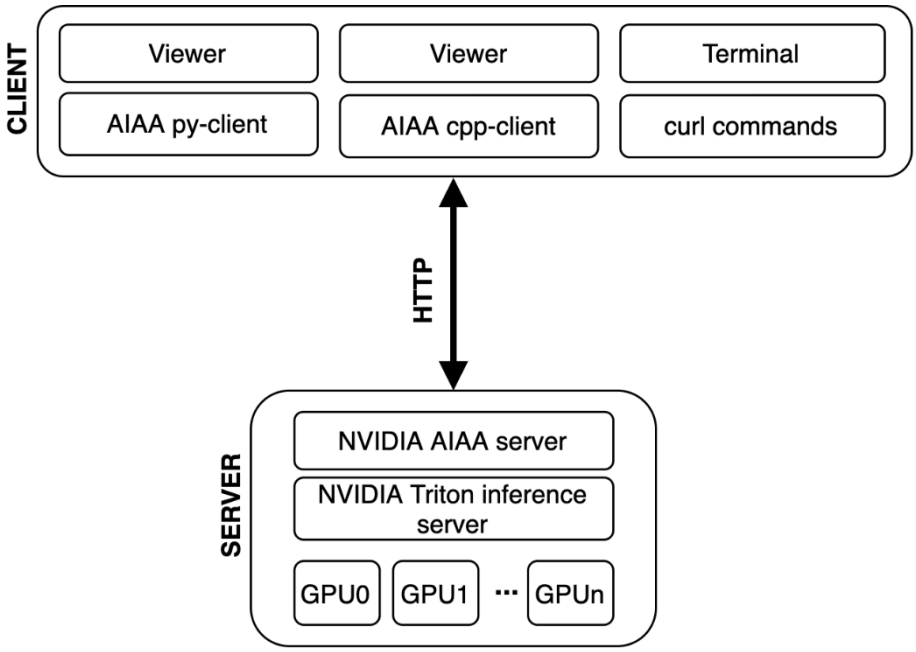

Fig. 2 – Overview of AI-Assisted Annotation (AIAA) concept

The key features in the AIAA tool are the possibilities to use segmentation, annotation, or deep grow methods, which help in labelling the data sets.

3.1.2. Clara Training Framework

The training framework is part of the Clara Train SDK. It is based on Python and provides a fast way to create deep learning models for medical data processing. Users can start with pre-trained models built by NVIDIA. Those models are packed in the so-called Medical Model Archive (MMAR) format. This format helps to start the process of creating the models and reduces the time to model if the pre-trained models are used, and the user just needs to fine-

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

tune them on his/her data. The MMAR format also brings enough freedom, e.g., users can extend several segmented labels in a model quite easily or add different data transformations. In Fig. 3, it is shown how the training framework is built on top of the PyTorch engine and the MONAI deep learning framework.

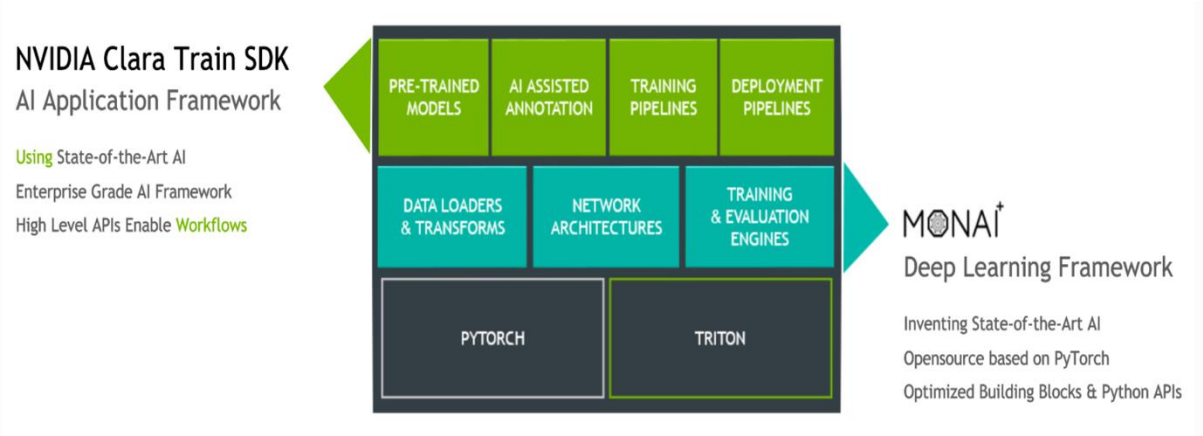

Fig. 3 – Clara Training Framework overview [9]

3.2. Frontend

In our medical image processing concept, we have a frontend solution represented by capable software that provides viewing and processing of the data, see Fig. 1. We have selected 3D Slicer as a viewer that can easily be extended to an AIAA client and bring additional functionality to medical doctors, see Fig. 2. A detailed description of 3D Slicer is in the following section.

3.2.1. 3D Slicer

3D Slicer is a free open-source and multi-platform software dedicated to medical image computing [5]. It supports various visualizations and provides advanced functionalities for image segmentation and registration in various application domains. To some extent, it is like a radiology workstation, mainly in terms of visualizations. Still, in contrast to that, 3D Slicer offers a higher level of extensibility and independence on specific hardware. On the other hand, 3D Slicer is not an FDA-approved software, so it cannot be used other than a clinical research application.

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

3D Slicer can serve as a programming platform within which new algorithms and methods for medical image analysis can be developed. Such approaches can be developed as simple concepts and can be fully integrated into the user interface in the form of extensions. 3D Slicer builds on a modular and layered architecture. It uses languages (C++, Python, JavaScript) and libraries (VTK, ITK, CTK) that provide high-level functionality and abstraction. The work environment of 3D Slicer is depicted in Fig. 4.

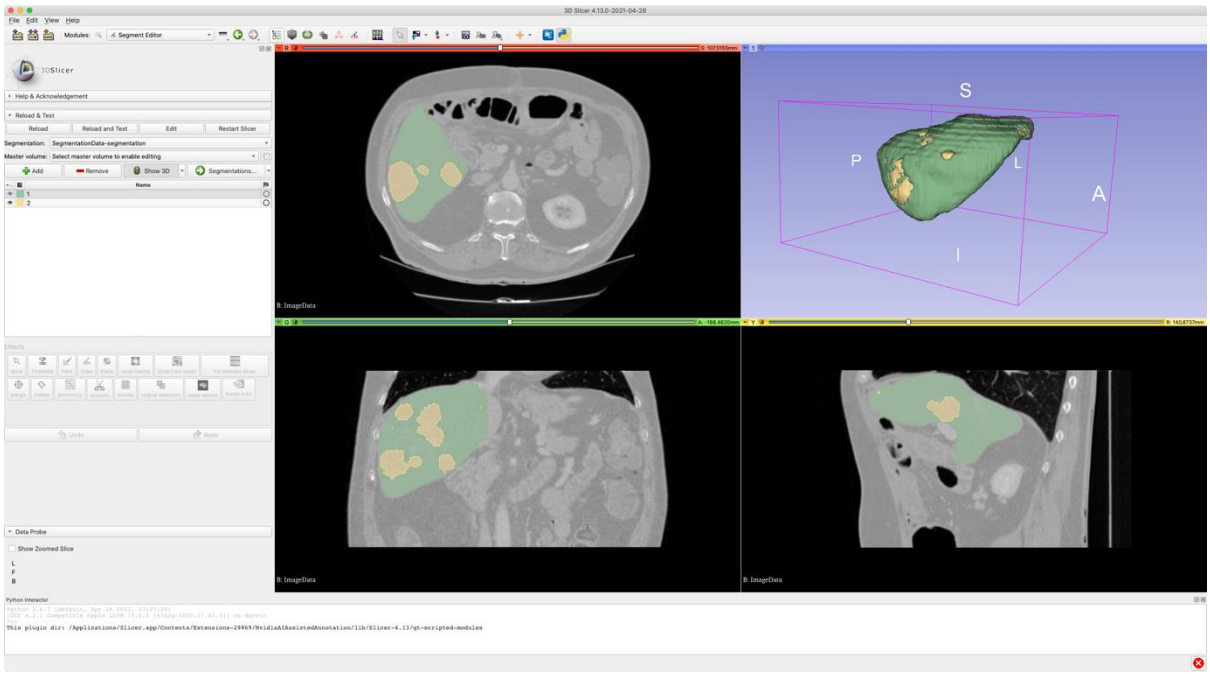

Fig. 4 – 3D Slicer working environment

We have selected Slicer mainly for the accessibility of advanced functions and easy extensibility that can be reached with Python programming. Although the core functionalities of Slicer are written in C++, APIs of core classes are accessible via Python using Python wrapping mechanisms. This interconnection with the Python language makes it possible to develop functional extensions in a simplified development process. The extensions can also use Python-based libraries and combine them with the functionality of Slicer.

3.2.2. API for AIAA clients

NVIDIA's AIAA provides two Client APIs, either in C++ or in Python. Both provide the same functionality and can be used to integrate AIAA Client in medical image viewers. We describe the Python version of the API here, but it also describes a C++ version since the implemented functionality is the same. The Client APIs are available at [10].

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

Python version of the AIAA Client API is written in a single Python file *client_api.py*, and it can be found at [11]. The script contains the main *AIAAClient* class, which defines methods to connect to the AIAA server. You can also list available models pre-loaded on the server and thus know what type of segmentation you can call and what tissue it applies to. Regarding the processing operations that can be called on the data, there are four main methods:

- Segmentation (*segmentation(self, model, image_in, image_out, session_id=None)*) – this method performs automatic segmentation by using MMAR segmentation type of model (in Slicer it corresponds to Auto-segmentation option).
- Dextr3d (*dextr3d(self, model, point_set, image_in, image_out, pad=20, roi_size='128x128x128', pre_process=True, session_id=None)*) – this method performs segmentation from boundary points by using MMAR annotation type of model (in Slicer it corresponds to Segment from boundary points option).
- Deepgrow (*deep grow(self, model, foreground, background, image_in, image_out, current_point=None, spatial_size=None, session_id=None)*) – this method performs segmentation based on user selection of foreground and background points. It can provide either 2D or 3D segmentation, and it uses deep grow models from MMAR model types (in Slicer, it corresponds to the Deepgrow option).
- Inference (*inference(self, model, params, image_in, image_out, session_id=None)*) – this method provides generic inference for given input image and model.

Besides the mentioned methods for the model inference, API offers several other tools to process the data before or after the inference. There is a tool to, e.g., resample the image, convert the 3D mask to polygons, update polygons, etc.

3.2.3. Slicer add-on

Official add-ons for Slicer can be downloaded and installed using the Slicer's Extension manager. The regular AIAA add-on provided by NVIDIA can also be installed this way. By searching for NvidiaAIAssistedAnnotation in the Extension Manager, you can install the AIAA extension.

Standard AIAA add-on uses the Python version of AIAA API (section 3.2.2) and provides client connection to AIAA server running at some visible IP address in the network. In terms of data exchange that is copied between the client and the server, raw image data is sent for processing to the server, and the result in the form of the image binary mask is sent back. The standard situation is depicted in Fig. 2.

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

Our extension to this Slicer add-on broadens its possibilities so that a user can securely ask for computational resources on the HPC cluster directly and run there the Clara AIAA server to provide model inference. To connect to the cluster resources, the frontend user must have a regular user account on the IT4Innovations infrastructure, and a project has to be assigned to the user with some computational resources available. If so, then the necessary conditions are met, and the user can use the cluster for the assisted annotation. The situation is depicted in Fig. 5, where the typical data flow to/from the inference server is denoted by blue colour.

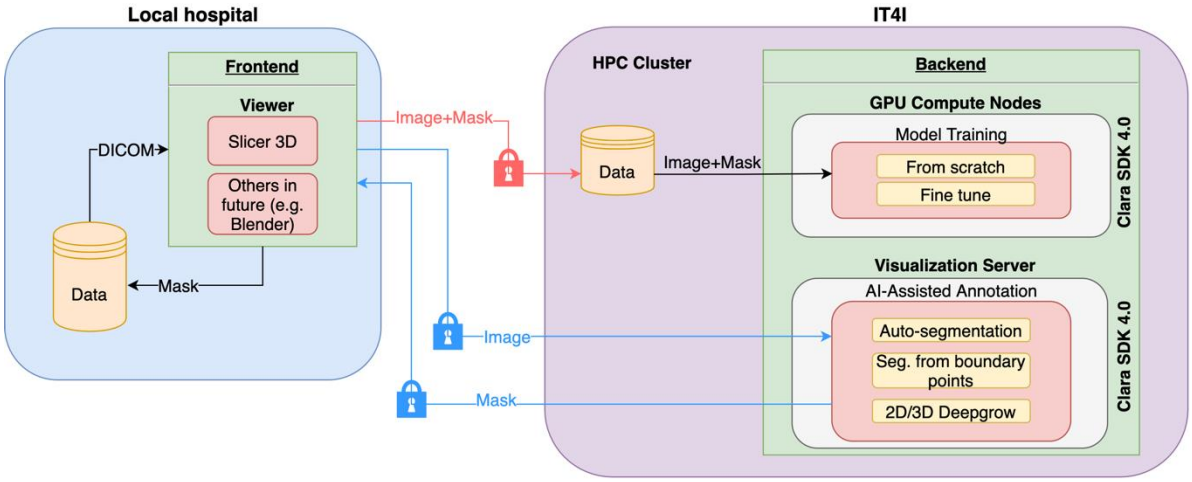

Fig. 5 – Distinction of different data flows between frontend (viewer) part and the backend

Since we also want to have the ability to use the cluster resources for model training, we needed to create a way how to upload the data from the viewer to some specific path on the cluster. Such data must represent the validated dataset suitable for training. It means a raw image and a correct binary mask validated by a doctor must be sent to the cluster. In Fig. 5, this is denoted by the red data path. Again, we use the user credentials to create a secured way to transfer the data to the cluster.

The AIAA add-on for Slicer is completely Python-based. We use Paramiko and scp modules to provide the functionality to establish and manage the connection to the cluster. These modules are not part of the regular Python distribution in Slicer and need to be installed as an extra. We have included the installation process of the modules as a part of the AIAA add-on setting. The AIAA add-on itself is installed in the following way. First, the add-on needs to be added via Application Settings, see Fig. 6.

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

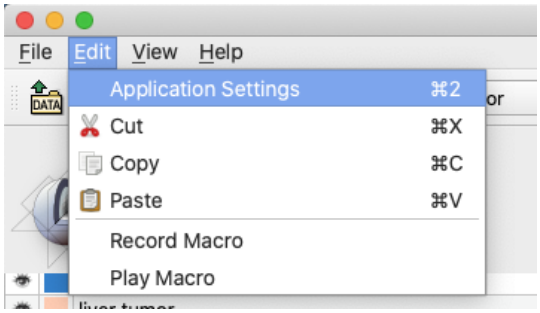

Fig. 6 – Access to Application Settings in Slicer

A user from the Application Settings window needs to add the path to the modified version of the AIAA extension from within the Modules section in the settings, see Fig. 7.

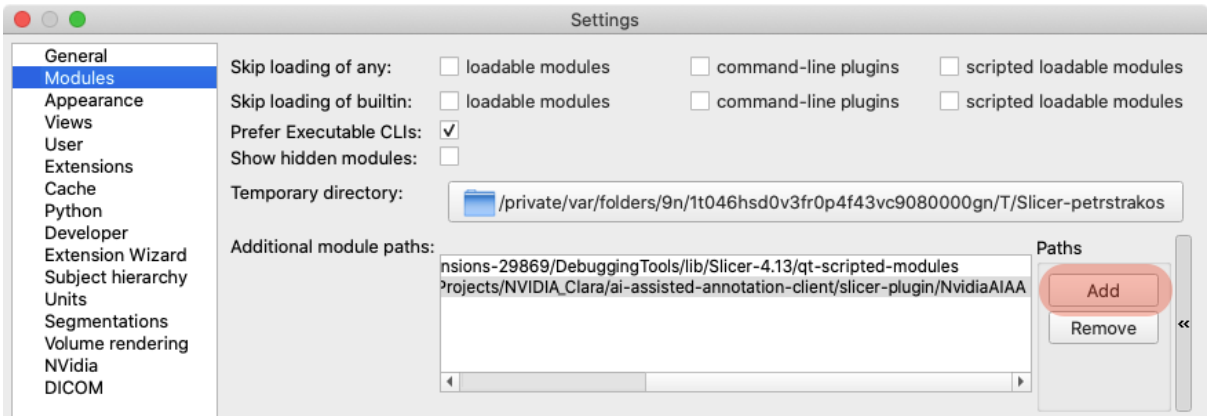

Fig. 7 – Adding the path to a user extension in the modules tab of Slicer's setting

After restarting the 3D Slicer, the user can find in the Application Settings the new item in the list named AIAA, see Fig. 8. A specific setting to access the cluster is brought to the user and needs to be filled in properly. Also, if some required Python module is missing (Paramiko, scp), the Python Interactor prints out the message "Please install required dependencies to use AIAA extension!" and the Install dependency button becomes active (otherwise greyed out if no dependency is missing). After installing the required Python modules, Slicer needs to be restarted again to make the changes effective. If Slicer is started and no dependencies must be installed, Python Interactor displays the message "AIAA extension has been activated!".

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

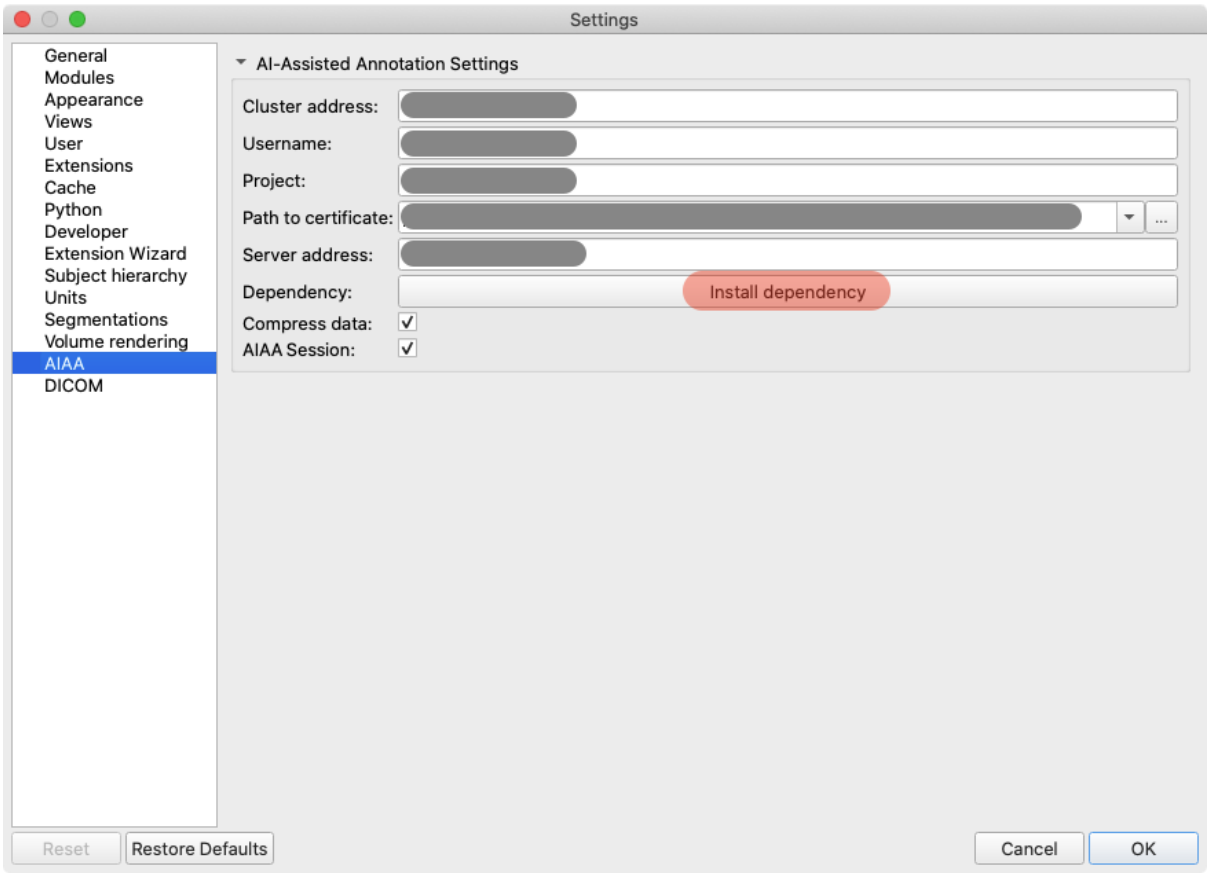

Fig. 8 – Settings of the AIAA extension in Slicer

When the AIAA add-on is added as a module, the user can find this tool in the Segment Editor of the application, see the lower right part in Fig. 9. To use the AIAA add-on, it is necessary to first load some image data by using the DATA or DCM buttons, see the top left part in Fig. 9, before entering the Segment Editor.

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

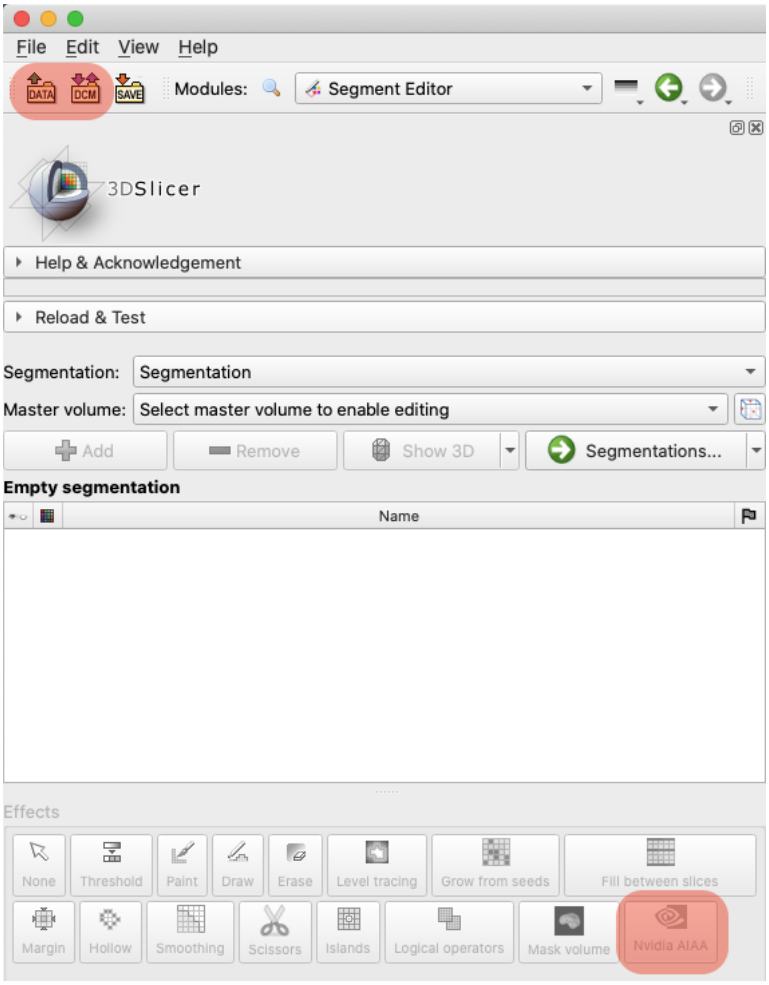

Fig. 9 – Slicer's Segment Editor with the AIAA extension

After loading some image data, entering the Segment Editor, and activating the AIAA tool, the user gets the possibility to submit the AIAA session with Clara at the backend on the HPC cluster and connect to it, see Fig. 10. The AIAA tool has three different modes to provide the user with assistance in medical image segmentation. The modes correspond to different types of pre-trained models as defined by MMAR. Users can train and use models for fully automatic segmentation of a tissue. Such model corresponds to the MMAR type denoted as segmentation. The other type of model provides segmentation based on boundary points. Such a model is marked as an annotation model in the MMAR. Then the user needs to select the proper model for the structure of interest and then specify input points near the edges of the structure, one on each side, therefore six points in total. The last type of model is specified as deep grow in MMAR type, and it performs segmentation based on user selection of foreground and background points. Users can either use 2D or 3D deep grow segmentation.

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

The 2D segmentation operates slice by slice over added points. The 3D segmentation operates in all three dimensions based on the foreground and background points added.

The first two segmentation modes in the AIAA module, the auto-segmentation and segment from boundary points, are organ-specific, meaning that to properly segment a tissue, a pre-trained model for segmentation of such tissue needs to be used. The deep grow segmentation is an option based on a general pre-trained model capable to segment any tissue of the user selection. The last option in the AIAA module represents the extension that allows the compression of the image data and resulting segmentation mask and sends it securely to the HPC cluster. The user is notified about the status of a data sending operation. The data are stored in the specific project folder on the cluster where it can be used by the training framework of the Clara train SDK that runs on the cluster in a singularity container where it can utilize allocated GPU resources for training or fine-tuning of models.

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

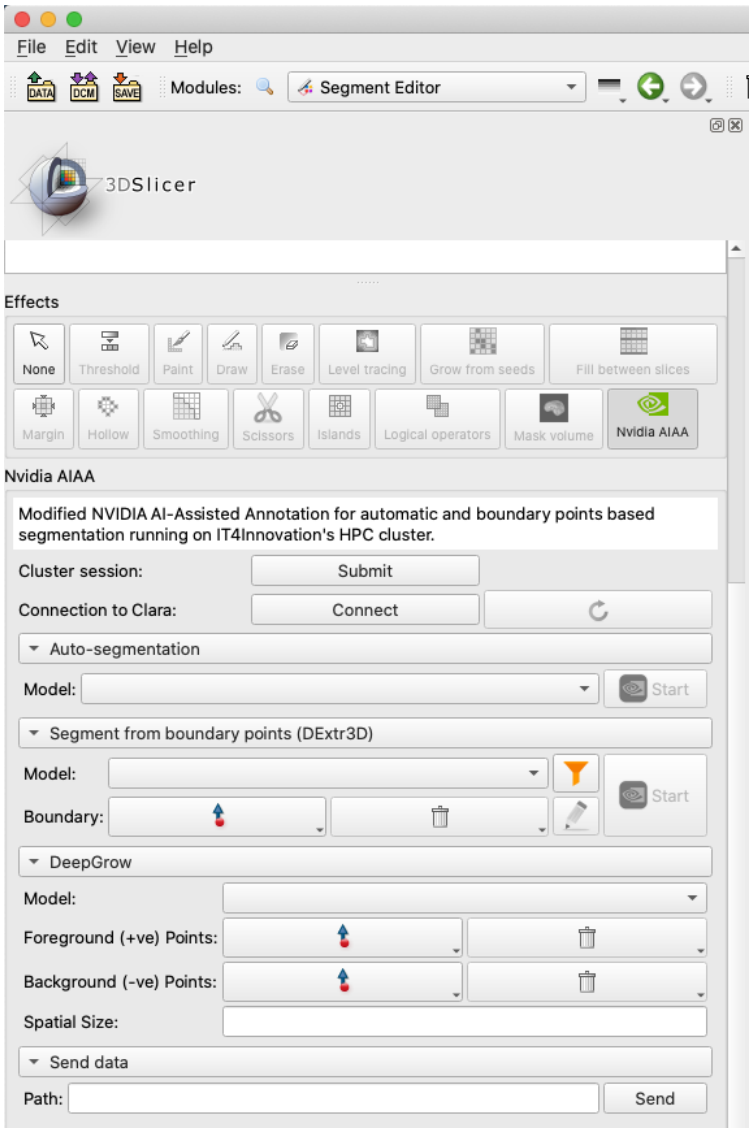

Fig. 10 – Slicer's Segment Editor with the AIAA extension

3.3. Backend

3.3.1. Cluster part

3.3.1.1. Singularity container with Clara

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

Singularity [12] provides virtualization at the level of the operating system. This technique is also known as containerization. Containers can be easily moved from system to system and thus bring reproducibility to a new level. One of the main uses of Singularity as software for running containers was a need to bring reproducibility to scientific computing represented by High-Performance Computing (HPC).

To run Clara Training Framework on the HPC cluster, it is first necessary to create a Singularity container. To use Singularity that is available as a module on all the IT4Innovations' clusters, one needs to use the following command:

```
[user1234@vizserv1.karolina ~]$ ml Singularity
```

Clara Train SDK 4.0 is distributed as a Docker container. It is possible to build a Singularity container directly from the Docker repository as:

```
$ singularity build clara-train-sdk-4.simg docker://nvcr.io./nvidia/clara-train.sdk:v4.0
```

To run the container with Clara Training Framework on a pre-allocated computational resource with GPU accelerators, it is necessary to execute:

```
[user1234@vizserv1.karolina ClaraTrain~]$ singularity exec -B Clara_4_0:/workspace
-B /apps/all/CUDAcore/11.2.2/:/usr/local/cuda --nv Clara_4_0/clara-train-sdk-4.simg /bin/bash
```

To access directories on the HPC cluster from within the Singularity, the directories must be mounted. To mount a directory, use the option *-B <source_dir>:<mount_dir>*. We mount a working directory as */workspace* and a directory with CUDA from the core system to a specific */usr/local/cuda* folder in the Singularity. Mounting CUDA solves possible driver-related errors in Singularity.

If the execution of the Clara container is successful, you will see Singularity in the terminal.

```
Singularity>
```

Inside the */workspace* folder, you can check that the data are correctly mounted.

```
Singularity> ls /workspace/
clara-train-sdk-4.simg  data  docker-compose.env  docker-compose.yml  mmars
```

It is recommended to create two folders in the working directory:

- *data* – to store datasets,
- *mmars* – to store Medical Model Archive (DL models for training).

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

In the next chapter, the focus is on preparing data and models from MMAR (Medical Model Archive). We will look at how to download pre-trained models and describe the development environment that enables training, validating, or bringing your models.

3.3.1.2. Clara Training Framework

This chapter describes the basics of the Clara Training Framework. It provides instructions on how to prepare your data and how to work with MMARs. Also, descriptions of how to train, validate and export segmentation or classification model are presented. The section describes techniques to train deep learning models from scratch or fine-tune already existing pre-trained models.

Preparing the data

The dataset is separated into training, testing and validation folders. You can split the data into a 70:20:10 ratio. If the data are in DICOM format or the resolution is not isotropic, you should use a data converter. The converted data are in NIfTI format, and 1x1x1mm resolution is used.

The following Clara command converts all DICOM images in a folder:

```
$ medl-dataconvert -d your/data/directory -r 1 -s .dcm -e .nii -o your/output/directory
```

If you convert images with labels, use the command with -*l* flag, which uses the nearest neighbour interpolator to correctly convert label data.

```
$ medl-dataconvert -d your/data/directory -r 1 -s .dcm -l -e .nii -o your/output/directory
```

Models

Some pre-trained models are publicly available in NVIDIA NGC Catalog [13]. The models are used for organ segmentation, annotation and classification based on the state-of-the-art networks (3D U-Net, AH Net, DenseNet, ResNet, Dextr3D). The models are packaged as MMARs (Medical Model Archive). Each MMAR contains all the configurations needed to train the model and scripts needed for model development tasks.

To display the list of available NGC models, use the command below (directly from Singularity, or the NGC CLI tool can be installed on any standard platform from [14]:

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

```
$ ngc registry model list nvidia/med/clara_*
```

To download selected MMAR (Medical Model Archive) from NGC site use:

```
$ ngc registry model download-version "nvidia/med/clara_pt_liver_and_tumor_ct_segmentation:1"
```

The folder structure of MMAR looks as following:

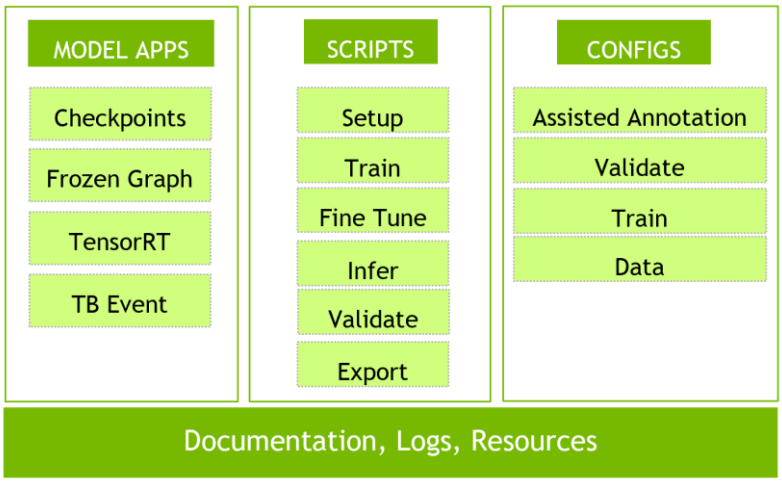

Fig. 12 – Folder structure of MMAR model architecture

Training Workflow

To train the model, open the downloaded MMAR model folder. First, navigate to the config folder and choose *the environment.json* file to set up the paths. The example is shown on the code snippet below. The user should change the following parameters:

- *DATA_ROOT* – specifies the path to the dataset
- *DATASET_JSON* – specifies the path to data list (typically called *dataset_0.json*)

```
"DATA_ROOT": "/workspace/data/Task03_Liver_nii",
"DATASET_JSON": "config/dataset_0.json",
```

Fig. 13 – Important values in *environment.json* file

Next, open the script *dataset_0.json* and modify it. The JSON file describing the data list must contain paths to all image files. As you can see in the code snippet below, the data are split into three parts - training, testing and validation. The user should set the number of images in each of the categories. The user also sets the paths to each volume image and its label. The pairs include:

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

- Image – volume image of one patient
- Label – the label corresponds to the ground truth mask


```
"num_training": 91,
"num_testing": 14,
"num_validation": 26,
"training": [
  {
    "image": "/workspace/data/Task03_Liver_nii/imagesTr/liver_4.nii",
    "label": "/workspace/data/Task03_Liver_nii/labelsTr/liver_4.nii"
  },
  {
    "image": "/workspace/data/Task03_Liver_nii/imagesTr/liver_60.nii",
    "label": "/workspace/data/Task03_Liver_nii/labelsTr/liver_60.nii"
  },
  {
    "image": "/workspace/data/Task03_Liver_nii/imagesTr/liver_125.nii",
    "label": "/workspace/data/Task03_Liver_nii/labelsTr/liver_125.nii"
  }
```


Fig. 14 – Structure of *dataset_0.json* file

If you want to change the architecture of the network, add more labels, or change parameters for training, go to *the config_train.json* file. This training config file defines the configuration of the training workflow. The config contains three sections: global variables, train, and validate. The script is used by all below mentioned training commands.

Typical global variables are shown in the code snippet below. On the left image, you can see an example of defined global variables. You can easily experiment with different training settings without modifying the config file. The train section defines the components for the training process, including *'loss'*, *'optimizer'*, *'model'*, *'metrics'* etc. The part of the file is shown in code snippets below. On the right, there is an example of a defined model from the train section. Similarly, the validate section defines the components for the validation process.

You can also use externally implemented components. For more details, see section Bring your components.


```
"cache_rate": 1.0,
"epochs": 1260,
"in_channels": 1,
"out_classes": 3,
"num_interval_per_valid": 20,
"learning_rate": 2e-4,
"multi_gpu": false,
```



```
"model": {
  "name": "UNet",
  "args": {
    "dimensions": 3,
    "in_channels": "{in_channels}",
    "out_channels": "{out_classes}",
    "channels": [16, 32, 64, 128, 256],
    "strides": [2, 2, 2, 2],
    "num_res_units": 2,
    "norm": "batch"
  }
```


This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

Fig. 15 – Setting of *config_train.json*

Navigate to the command folder and run the script *train.sh*. An example is shown on the command below. This script trains the model from scratch with settings specified by the user. The training runs on a single GPU.

```
$ cd path/to/mmars/clara_pt_liver_and_tumor_ct_segmentation_v1/commands
$ ./train.sh
```

To launch multiple GPU training run the script *train_multi_gpu.sh*. Such script can be obtained by simple modification of *train_2gpu.sh* script provided by NVIDIA.

```
$ cd path/to/mmars/clara_pt_liver_and_tumor_ct_segmentation_v1/commands
$ ./train_multi_gpu.sh
```

To continue training from an already existing model, use the script *fine-tune.sh*. The output of this command is the same as from the commands above.

```
$ cd path/to/mmars/clara_pt_liver_and_tumor_ct_segmentation_v1/commands
$ ./finetune.sh
```

When the training is finished, you should see the following files in the model's folder.

- *Model.pt* – represents the best model.
- *Final_model.pt* – when the training is complete, this file is written to disk and can be used for fine-tuning. It is usually NOT the best model.
- *Event files* – these are TensorBoard events that you can be visualized in the graph.

Export a trained model checkpoint from *the model.pt* to *model.ts*.

```
$ cd path/to/mmars/clara_pt_liver_and_tumor_ct_segmentation_v1/commands
$ ./export.sh
```

To monitor the progress of training, run the following command to run TensorBoard visualization. On the image below Fig. 16, the training accuracy is shown. You can visualize training loss, validation accuracy etc., in this way.

```
$ python3 -m tensorboard.main --logdir="/path/to/mmars/clara_ct_seg_liver_and_tumor_amp_v1/models"
```


This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

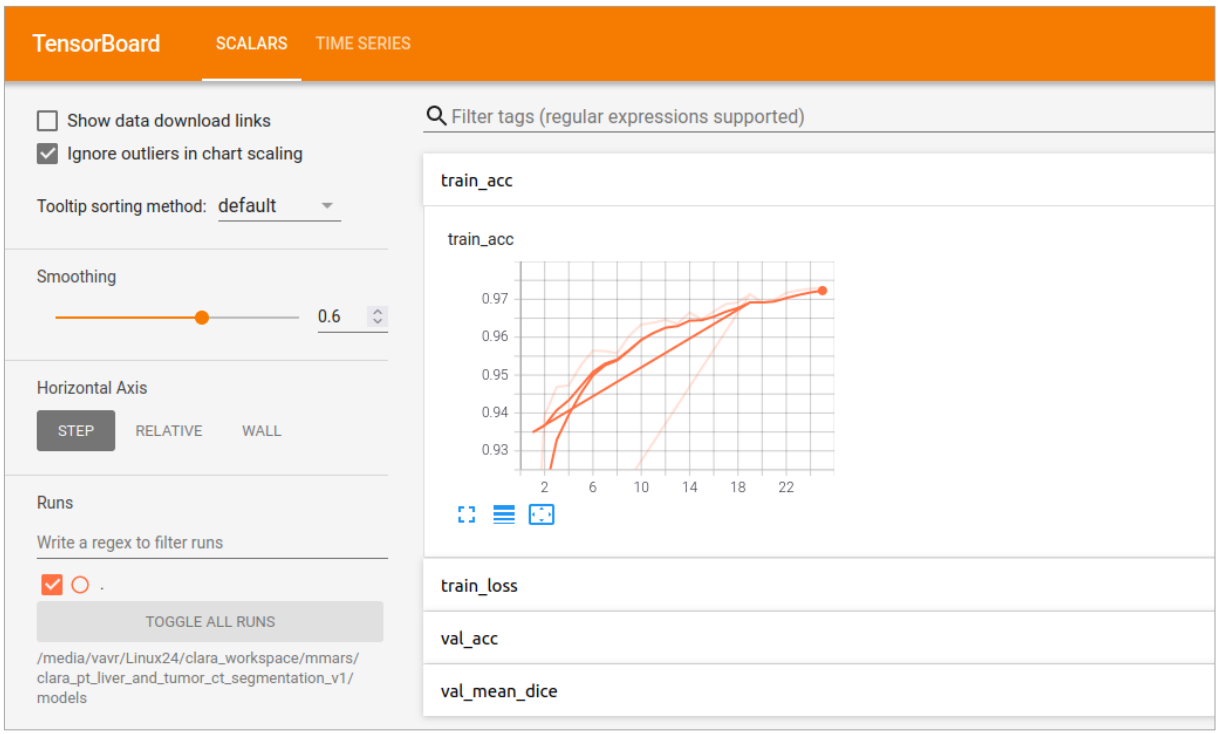

Fig. 16 – Tensorboard visualization of training progress

Validation Workflow

After the training is completed, try the validation and inference scripts. For this purpose, the configuration script *config_validation.json* is used. The same configuration file is used for both validation and inference. The validation result files are created in the *eval* folder of the MMAR after the use of *validate.sh* command.

```
$ cd path/to/mmars/clara_pt_liver_and_tumor_ct_segmentation_v1/commands
$ ./validate.sh
```

Call the command *infer.sh* to run inference on the model from Medical Model Archive. Unlike the validation, for inference, the metric value specified in the configuration file will not be computed, and no ground truth label is needed. The output of the inference is saved again in the *eval* folder.

```
$ cd path/to/mmars/clara_pt_liver_and_tumor_ct_segmentation_v1/commands
$ ./infer.sh
```

Bring your components (BYOC)

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

Clara allows researchers to solve problems and innovate by writing their components in a modular way. Users can write their components in python files and add them to the *train_config.json*.
List of components that users can add:

- **Transforms** that are applied to the input images and labels.
- **Model** of neural network.
- **Loss function.**
- **Optimizer** for finding minimal loss during training.
- **Metrics** to measure the quality of the model during training.

3.3.1.3. Cluster setup for the AIAA backend

If the Singularity container with Clara Train SDK is prepared (section 3.3.1.1), just two additional bash scripts need to be set up inside the project folder on the cluster. Both scripts have to be accessible for the cluster user. The first script, run_clara.sh is submitted as a non-interactive cluster job and runs the Singularity container. This script is submitted remotely from the Slicer frontend via Submit button.

```
#!/bin/bash

ROOT_DIR=PATH_TO_AIAA_FOLDER_ON_CLUSTER/apps
USER_DIR=PATH_TO_AIAA_FOLDER_ON_CLUSTER/users/${USER}
TEST_DIR=PATH_TO_AIAA_FOLDER_ON_CLUSTER/apps/bandwidthTest

#####################################################

ml CUDAcore/11.2.2
${TEST_DIR}/bandwidthTest

#####################################################

ml Singularity
singularity exec -B ${USER_DIR}/workspace:/workspace \
                 -B /apps/all/CUDAcore/11.2.2:/usr/local/cuda \
                 --nv ${ROOT_DIR}/clara-train-sdk-4.simg /workspace/my_scripts/start_clara_4.sh
```

Fig. 17 – Content of the run_clara.sh script

The second script, called start_clara_4.sh, is started inside the container. It sets up the important parameters for the AIAA server, such as CUDA paths, inference engine, etc., and starts the AIAA server.

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/targets/x86_64-linux:/usr/local
/cuda/targets/x86_64-linux/lib
export TZ="UTC"

start_aiaa.sh --engine AIAA --triton_proto grpc --triton_start_timeout 15 \
              --triton_model_timeout 60 --triton_verbose 1 \
              --workspace /workspace
```

Fig. 18 – Content of the start_clara_4.sh script

4. References

[1] Esteva, A., et al. (2019). A guide to deep learning in healthcare. Nature Medicine, 25(1), 24-29. doi:10.1038/s41591-018-0316-z.
[2] Lundervold, A. S., & Lundervold, A. (2019). An overview of deep learning in medical imaging focusing on MRI. Zeitschrift Fur Medizinische Physik, 29(2), 102-127. doi:10.1016/j.zemedi.2018.11.002.
[3] Hesamian, M. H., Jia, W., He, X., & Kennedy, P. (2019). Deep learning techniques for medical image segmentation: Achievements and challenges. Journal of Digital Imaging, 32(4), 582-596. doi:10.1007/s10278-019-00227-x.
[4] Sahiner, B., et al. (2019). Deep learning in medical imaging and radiation therapy. Medical Physics, 46(1), e1-e36. doi:10.1002/mp.13264
[5] Fedorov A., et al. 3D Slicer as an Image Computing Platform for the Quantitative Imaging Network. Magn Reson Imaging. 2012 Nov;30(9):1323-41. PMID: 22770690. PMCID: PMC3466397.
[6] NVIDIA Clara Imaging. https://developer.nvidia.com/clara-medical-imaging (2020).
[7] NIfTI: Neuroimaging Informatics Technology Initiative. https://nifti.nimh.nih.gov/ (2021).
[8] Project MONAI. https://monai.io/ (2021).
[9] Essential concepts – Clara Train SDK v4.0 documentation. https://docs.nvidia.com/clara/clara-train-sdk/pt/essential_concepts.html (2021).
[10] GitHub – NVIDIA, ai-assisted-annotation-client. https://github.com/NVIDIA/ai-assisted-annotation-client (2021).
[11] ai-assisted-annotation-client/client_api.py. https://github.com/NVIDIA/ai-assisted-annotation-client/blob/master/py_client/client_api.py (2021).
[12] Singularity. https://sylabs.io/singularity/ (2021).
[13] NVIDIA NGC. https://ngc.nvidia.com/catalog (2021).
[14] NGC CLI. https://ngc.nvidia.com/setup/installers/cli (2021).

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, Montenegro.