Tutorial: Magnetic Resonance Segmentation That Detects Glioblastoma Multiforme

Keith Bova

Abstract—This document proposes an investigation into the development of open-source image processing libraries that can be used to improve the early detection of glioblastoma multiforme (GBM), a very lethal form of brain cancer. Supervised and unsupervised machine learning techniques have been explored in the past. The novel *UNETR++* transformer is presented, along with an in-depth discussion of the *BraTS* dataset, which originated from The Cancer Imaging Archive (TCIA). *UNETR++* is a high-performance code that was optimized for NVidia hardware; however, designing the code around one hardware vendor is not ideal for long-term performance. Modifications to the training and testing scripts should improve image segmentation metrics and performance portability. This will be validated by compiling the code for different architectures, and observing changes in run time, performance, and accuracy.

Index Terms—Glioblastoma Multiforme, GBM, Exascale, Instance Segmentation, Magnetic Resonance, Performance-Portability

1 Introduction

Magnetic resonance (MR) imaging is the optimal process to detect diseases that affect soft tissue, such as most cancers. Cancer therapy has improved substantially in recent years; however, early diagnosis affects the outcome of the prognosis. A brain tumor is among the most lethal forms of cancer, and glioblastoma multiforme (GBM) is a grade four brain tumor. Fewer than 5% of people diagnosed with GBM survive more than five years [TJ17]. In the last year, image segmentation has proven to be the key backbone of automated GBM detection [Ba24]. Computers can viably segment GBM using supervised and unsupervised techniques, with the former outperforming the latter in terms of accuracy [JA+15].

2 Motivation

Wide-spread image processing algorithms (such as those machine learning techniques discussed in Module 8 and beyond) that improve the early detection of GBM will reduce the lethality of the disease, because they will provide doctors with more time to treat their patients before the malignant tumor has spread to other parts of their brain. Metrics for evaluating the performance of image segmentation include *intersection over union* (IoU), *dice coefficient*, *precision and recall*, *pixel accuracy*, and *Hausdorff Distance*. *Hausdorff Distance* is defined as the longest distance that a person can be forced to traverse between two finite sets, whereby they must travel from one set into another [HKR93]. The *Hausdorff Distance* is a necessary calculation that draws the boundaries between cancerous and noncancerous cells in GBM segmentation.

Bonada et al. discuss some of the ethical issues with the routine use of deep learning for GBM segmentation [Ba24]. Artificial intelligence and machine learning (AI/ML) driven decision-making must be approached with caution in clinical settings. The integrity of medical training data sets is extremely important to ensure accurate models that can make unbiased inferences. Porz et al. discuss how traditional GBM segmentation metrics typically focus on tumor volume, rather than characteristics of the tumor itself [Por+14]; this is not ideal, however, the development of GBM volumetric segmentation algorithms is necessary and can be adapted. The goal of the project is a fast, efficient and portable technique for segmenting GBM in 3D space.

3 Expected Contribution

Fully automated cancer segmentation techniques have existed for many years; however, the increasing strength of MR machines also substantially increases the amount of data that must be processed with each scan. For example, the new *Siemens MAGNETOM Terra* MR scanner is rated for seven Teslas [Hea25]. Fortunately, many image processing algorithms can be computed in parallel. Existing parallel methods that are written for the CPU can often be rewritten using open source tools and compiled for SIMD or GPU memory spaces [Tro+22] with massive improvements in training and inference times. Many open-source image processing algorithms have high-performance portability layers built in; examples of this are the *YOLO* [Red+16] and *Detectron2* [Wu+19] algorithms. Unfortunately, *YOLO* and *Detectron2* are fast and efficient 2D segmentation algorithms and will not be sufficient for the MR scans that exist in 3D.

Software portability layers are essential for long-term performance [CETS14] because portability layers safeguard code against underlying changes in hardware architecture. Programmers who write their code with portability in mind open the door to a diverse landscape of parts that can be used to create high-performance machines [SMI20] [Com19]. Compiler optimizations that decrease the run time of these critical codes will improve their deployment in hospitals and other oncology clinics.

Method	Architecture	Proposed
pytorch-3dunet	CNN	Transformers should outperform CNN in terms of accuracy
unetr++	Transformer	Training is only done using FP16 precision, which is not ideal with modern GPU technology

TABLE 1
Expected contribution against competitive methods

3.1 Against Competitive Methods

Fast and efficient open-source volumetric segmentation algorithms already exist that can detect GBM; for example, the *PyTorch 3D UNET* is a convolutional neural network (CNN) based model that is used for volumetric segmentation [Wol+20]. While the CNN architecture is good for structured data, transformers can often outperform the accuracy of CNNs at the cost of computational expense. *UNETR++* [Sha+24] is an example of a volumetric segmentation algorithm that uses transformers with good results. Table 1 shows this expected contribution. For the purposes of this project, these algorithms will be built upon–rather than trying to reinvent something from scratch.

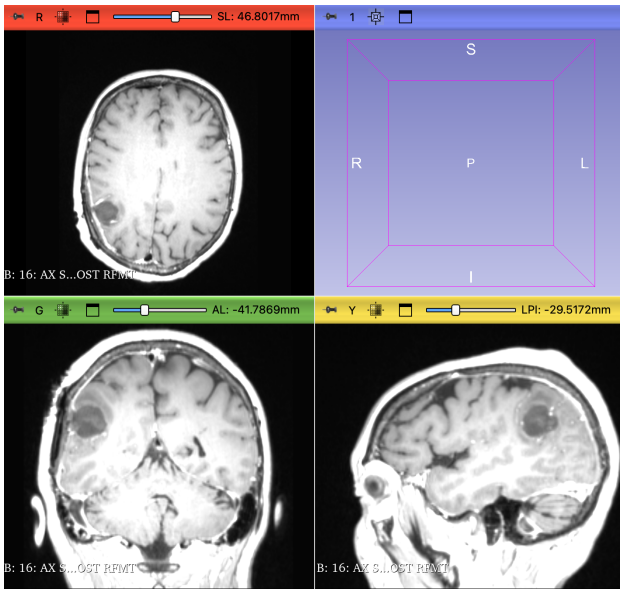

Fig. 1. Example of volumetric segmentation data labeling using the 3D slicer application.

4 Code

The source code from this project will come from the official *UNETR++* library. *UNETR++* contains training and testing scripts for their large transformer; however, as mentioned in Table 1, the FP16 precision is not ideal. The source code will have to be modified so that the transformer uses FP64 (double) precision. It is unknown wether this change will alter the training time, but it is hypothesized that it will cause an increase of accuracy.

MR scans contain a point cloud of the magnetic field strength that can be stored in three dimensional arrays; these arrays, when sliced in the transverse plane, can be converted into .JPEG files. The .JPEG file format is ubiquitous and an ideal format that many image processing libraries support; however, 2D image segmentation is not sufficient for a complete GBM detection algorithm. Nonetheless, for completeness, Appendix A contains code that can generate these ”.JPEG” files.

As an aside, computing the delaunay tesselation of a CT scan using the process proposed in [Rie21] will result in a ”.stl” solid model. These ”.stl” files can be viewed in a wide variety of 3D rendering engines, such as Kitware ParaView [Hen02]. It is possible to label the surfaces of these 3D models using Blender [Com18] and train surface segmentation algorithms like MedMeshCNN [Sch+21]. Appendix B provides the code used to generate such a ”.stl” file from a CT scan.

5 Dataset Requirements

The *UNETR++* GitHub repository contains a link to a large GBM dataset that was annotated by medical professionals; this dataset is named ”brain tumor segmentation” (BraTS). *The Cancer Imaging Archive* (TCIA) [Cla+13] hosts the original, unannotated dataset used for BraTS. TCIA is a service provided by the *National Cancer Institute*. TCIA operates out of the University of Arkansas for Medical Sciences, and has a wide variety of medical datasets for digital imaging processing. The MR scans are stored in a *digital imaging and communications in medicine* (.DICOM) format that is not compatible with most image processing libraries. The data from TCIA must be preprocessed before it can be used to train a neural network; however, the data from BraTS is already annotated and ready for training. As a note, if more MR data was acquired, the scans could be labeled using open-source tools, such as [Fed+12]. The 3D-Slicer application should be able to label data for *PyTorch 3D UNET* and *UNETR++*. An example of this data labeling process is shown in Figure 1.

The TCIA dataset contains 230 cases of GBM across 468 different studies. The dataset is distributed under the TCIA restricted license and explicitly prohibits the identification of any patients, or the training of any facial recognition algorithms. The dataset also provides a metadata .CSV file with a table that contains the collection, data description, file name and size, manufacturer, modality, and patient identifier.

Preprocessing requires first iterating over the entire dataset and loading the DICOM files into memory. Once loaded, the headers are checked to classify if the media is either a MR or a CT scan. The data has to be separated into two discrete groups: one that contains the set of all CT scans, and another that contains the set of all MR scans. TCIA asserts that all the patients in the dataset were diagnosed with GBM, so, theoretically, a GBM segmentation algorithm should be capable of returning 100% accuracy on this particular dataset.

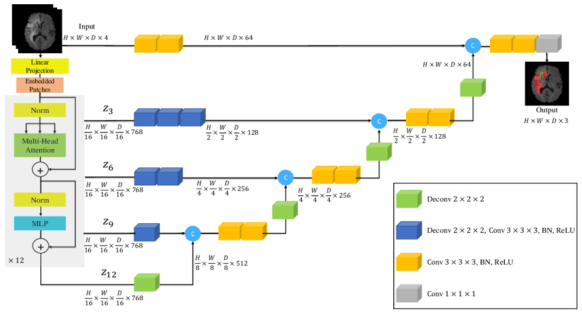

Fig. 2. System diagram for the *UNETR++*, as defined by [Sha+24]

5.1 Image Sizes (voxels for 3D slices)

The voxels for the 3D slices are stored in a matrix, whose size will vary with the resolution of each CT and MR machine. The wide variety of machines contained in the dataset causes substantial variance in the size of each image. It is expected that a seven tesla *Siemens MAGNETOM Terra* will create a substantially larger matrix than one from a previous generation, and as such, any numerical approach must be able to scale to larger resolutions. The resolution of each CT and MR machine increases the complexity of the underlying numerical algorithms for GBM segmentation because the patients in the *BraTS* dataset were measured using a wide variety of scanners.

6 Data Training and Testing

The MR data from 222 of the patients are used to train the *UNETR++* neural network. 65 of the patients are used to test that the training was done properly. These numbers can be found explicitly by reading the source code for *UNETR++*. The training process involves first cloning the repository. After the repository has been downloaded, the necessary python dependencies need to be installed. Downloading and extracting the *BraTS* dataset, and running the training bash script will start the training and testing processes. The training scripts explicitly specify the CUDA GPU memory space for *UNETR++*. The implementation was well done, but defining the code for a particular hardware vendor is not good for long term performance portability.

6.1 Network Architecture

As mentioned in Section 3.1, *UNETR++* is a transformer model. The training scripts contain a variable learning rate that starts at $l_r = 0.01$ and slowly decreases to $l_r = 0$ at the end of 1000 epochs. The neural network has over 40 billion parameters that include drop out rate, decay, learning rate, batch size, dice and cross entropy loss, among others. The system diagram is shown in Figure 2. Hyperparemeter tuning is defined in the training scripts.

7 Method Validation Metrics

After the training and testing phases, the results will be validated by inferencing on a MR scan that the network has not seen. Provided everything works properly, the question arises: how does the performance vary accross different computer architectures? If the portability layers of each code were written properly, it is expected that they should be able to compile for every tested computer architecture with no loss of accuracy. This is counterintuitive, because conventional wisdom would suggest that a solution that takes ten times as long to calculate using a CPU should be an order of magnitude more accurate than one computed using a GPU. If the thesis is correct, for the same data, compiler optimizations will accelerate the detection of GBM in the open-source implementations with negligible loss in accuracy.

The assumption is that parallel algorithms designed with performance portability in mind from the start will be able to leverage new upcoming architectures. In reality, the accuracy of the solution is much more important than the time it takes to obtain the solution. When designing a computer around critical medical codes, the *code* needs to guarantee that the design of the upcoming architectures prioritizes an increase in accuracy with every generation. If the accuracy does not change from the compiler optimizations, the GBM segmentation technique will be portable.

8 Discussion

GBM is a formidable disease that is very difficult to treat medically. The location of the tumor makes it difficult to remove completely without harming the patient; additionally, GBM is notorious for its fast growth and ability to spread throughout the brain. Numerical models and software techniques that can detect GBM in its early stages will assist in reducing its lethality, because the tumor is less likely to have spread to other parts of the brain.

While this proposal is focused on the early detection of GBM, these techniques should be applicable to the detection of other forms of cancer. Using commercially available, previous generation (CAPG) hardware, it is possible to train a neural network that can segment GBM. It is valuable to test the deployment of the segmentation algorithms across more recent computer architectures in the cloud, including the NVidia GH200 and other DGX machines. Not only is testing performance on the latest-and-greatest hardware with benchmarks representative of real-world problems a valuable investigation into computer engineering, but it will also provide insight into the scalability of medical codes that implement highly parallel numerical algorithms.

9 Conclusion

This proposal presented an investigation into open-source image processing frameworks that may be useful in the identification of GBM, and other forms of cancer. The *UNETR++* neural network is an impressive accomplishment and should be thoroughly evaluated and built upon; however, designing the system entirely around NVidia hardware is not good for long term scalability. Early results using CAPG hardware suggest that the algorithms should be able to inference with competitive frame-rates using modern exascale technology; if this is the case, the algorithms should be capable of processing extremely high resolution scans, such as the *Siemens Magetom Terra*, in a reasonable time-frame.

Appendix A
Python code for computing .JPEG files from MR data

Listing 1. Example .jpeg generation applied to DOI: 10.7937/TCIA.T905-ZQ20

```
import os
import pydicom
import pydicom.data
import numpy as np
from PIL import Image

def checkIfDcmFileIsMRI(pathToDCMFile):
    if(pathToDCMFile):
        if(pathToDCMFile.endswith(".dcm")):
            slice = [pydicom.dcmread(pathToDCMFile)]
            if(str(slice[0]['SOPClassUID']).endswith
                ("MR Image Storage")):
                return True
    return False

def checkIfDcmFileIsCT(pathToDCMFile):
    if(pathToDCMFile):
        if(pathToDCMFile.endswith(".dcm")):
            slice = [pydicom.dcmread(pathToDCMFile)]
            if(str(slice[0]['SOPClassUID']).endswith
                ("CT Image Storage")):
                return True
    return False

def openCT(file_path):
    slices = [pydicom.dcmread(file_path + '/' + s)
        for s in os.listdir(file_path) if s.endswith
        (".dcm")]
    print(slices[0])
    #print(slices)
    slices.sort(key = lambda sl: int(sl.
        InstanceNumber))
    locations = np.stack([sl.SliceLocation for sl in
         slices])
    images = np.stack([sl.pixel_array for sl in
        slices])
    slope = slices[0].RescaleSlope
    intercept = slices[0].RescaleIntercept
    images = slope * images.astype(np.float64)
    images += intercept

    slice_thickness = slices[0].SliceThickness
    spacing_bw_rows = float(slices[0].PixelSpacing
        [0])
    spacing_bw_cols = float(slices[0].PixelSpacing
        [1])
    return images, slice_thickness, spacing_bw_rows,
         spacing_bw_cols, locations

def openMRI(file_path):
    slices = [pydicom.dcmread(file_path + '/' + s)
        for s in os.listdir(file_path)  if s.
        endswith(".dcm")]
    #print(slices)
    slices.sort(key = lambda sl: int(sl.
        InstanceNumber))
    locations = np.stack([sl.SliceLocation for sl in
         slices])
    images = np.stack([sl.pixel_array for sl in
        slices])
    slice_thickness = slices[0].SliceThickness
    spacing_bw_rows = float(slices[0].PixelSpacing
        [0])
    spacing_bw_cols = float(slices[0].PixelSpacing
        [1])

    return images, slice_thickness, spacing_bw_rows,
         spacing_bw_cols, locations

def getPathsToAllMRIScansFromADirectory(path = "
    dataMRI"):
    listOfMRIScans = []

    for root, dirs, files in os.walk(path):
        for file in files:
            if(file.endswith(".dcm")):
                currentFile = os.path.join(root,file
                    )
                currentFileIsAMRIScan =
                    checkIfDcmFileIsMRI(currentFile)
                if(currentFileIsAMRIScan):
                    listOfMRIScans.append(
                        currentFile)
    return listOfMRIScans

def main()->None:
    os.makedirs("JpegImages", exist_ok = True)
    listOfMRIScans =
        getPathsToAllMRIScansFromADirectory("data")
    print(listOfMRIScans)
    i = 0
    for dcmFile in listOfMRIScans:
        ds = pydicom.dcmread(dcmFile)
        image = ds.pixel_array.astype(float)
        rescaled_image = (np.maximum(image,0)/image.
            max())*255
        final_image = np.uint8(rescaled_image)
        pillowImage = Image.fromarray(final_image)
        pillowImage.save(f"JpegImages/scan_{i + 1}.
            jpeg")
        i+=1
    return

if __name__ == '__main__':
    main()
```

Appendix B
Python code for computing .STL files from CT data

Listing 2. Example Elyse Rier Method applied to DOI: 10.7937/TCIA.T905-ZQ20

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import threading
import pydicom
import numpy as np
import os
from skimage import feature
import pyvista as pv
import time
import scipy

def checkIfDcmFileIsCT(pathToDCMFile):
    if(pathToDCMFile):
        if(pathToDCMFile.endswith(".dcm")):
            slice = [pydicom.dcmread(pathToDCMFile)]
            if(str(slice[0]['SOPClassUID']).endswith
                ("CT Image Storage")):
                return True
    return False

def getPathsToAllFilesWithExtension(folderName = "",
     extension = ".dcm"):
    listOfFiles = []
    for root, dirs, files in os.walk(folderName):
        for file in files:
            if(file.endswith(extension)):
                currentFile = os.path.join(root,file
                    )
                listOfFiles.append(currentFile)
    return listOfFiles

def checkIfFolderContainsCTScans(folderName)->bool:
    for root, dirs, files in os.walk(folderName):
```

```
        for file in files:
            if(file.endswith(".dcm")):
                currentFile = os.path.join(root,file
                    )
                currentFileIsACTScan:bool =
                    checkIfDcmFileIsCT(currentFile)
                #print(f"{currentFile} is {
                    currentFileIsACTScan}")
                if(not currentFileIsACTScan):
                    return False
    return True

def open_CT(file_path):
    slices = [pydicom.dcmread(file_path + '/' + s)
        for s in os.listdir(file_path)]
    slices.sort(key = lambda sl: int(sl.
        InstanceNumber))
    locations = np.stack([sl.SliceLocation for sl in
         slices])
    images = np.stack([sl.pixel_array for sl in
        slices])
    slope = slices[0].RescaleSlope
    intercept = slices[0].RescaleIntercept
    images = slope * images.astype(np.float64)
    images += intercept
    slice_thickness = slices[0].SliceThickness
    spacing_bw_rows = float(slices[0].PixelSpacing
        [0])
    spacing_bw_cols = float(slices[0].PixelSpacing
        [1])
    return images, slice_thickness, spacing_bw_rows,
         spacing_bw_cols, locations

def thresh_edge_CT(images,threshold):
    num_slices= images.shape[0]
    edges = images
    for i in range(num_slices):
        edges[i] = feature.canny(images[i]>threshold
            )
    return edges

def extract_PT_Cloud(edges,sl_thickness,row_spacing,
    col_spacing,locations):
    dimensions = edges.shape
    count =1
    for sl_num in range(dimensions[0]):
        for rows in range(dimensions[1]):
            for cols in range(dimensions[2]):
                if edges[sl_num][rows][cols] == 1:
                    if count==1:
                        point_cloud = np.array([
                            float((float(cols) *
                            col_spacing) - (
                            col_spacing/2)), float((
                            float(rows) *
                            row_spacing) - (
                            row_spacing/2)), float(
                            locations[sl_num])])
                    else:
                        holder = np.array([float((
                            float(cols) *
                            col_spacing) - (
                            col_spacing/2)), float((
                            float(rows) *
                            row_spacing)- (
                            row_spacing/2)), float(
                            locations[sl_num])])
                        point_cloud = np.vstack((
                            point_cloud,holder))
                    count += 1
    return point_cloud

def create_mesh(point_cloud):
        tree= scipy.spatial.KDTree(point_cloud)
        dist, ind= tree.query(point_cloud,100)
        dist_new= dist[:,1:]
        averages= np.zeros((dist.shape[0],1))
        for i in range(dist.shape[0]):
                averages[i] = np.mean(dist[i])
        alph= np.mean(averages)
        pcd= pv.PolyData(point_cloud)
        print("poly_data_created")
        mesh= pcd.delaunay_3d(alpha=alph)
        print("delaunay_created")
        return mesh,alph

def viewMesh(mesh):
        mesh.plot(show_edges=True)
        return

def save_mesh_stl(mesh,name):
        surf=mesh.extract_surface().clean()
        surf.save(name)
        return

def generateSTLFromFolderOfCTDicoms(fileDirectory =
    "", nameForSTLFileThisFunctionIsCreating:str = "
    "):
    if(not nameForSTLFileThisFunctionIsCreating):
        print("ERROR_FUNCTION_CALL_NEEDS_FILE_NAME")
        exit()
    print(f"running_elyse_rier_method_for_{
        nameForSTLFileThisFunctionIsCreating}")
    image_stack, sl_thickness, row_spacing,
        col_spacing , sl_locations = open_CT(
        fileDirectory)
    print(f"{fileDirectory}->{
        nameForSTLFileThisFunctionIsCreating}")
    edge_image = thresh_edge_CT(image_stack,600)
    surface_points = extract_PT_Cloud(edge_image,
        sl_thickness,row_spacing,col_spacing,
        sl_locations )
    #plot_PT_Cloud(surface_points)
    mesh,alpha = create_mesh(surface_points)
    #viewMesh(mesh)
    save_mesh_stl(mesh,
        nameForSTLFileThisFunctionIsCreating)
    return

def main():
    fileDirectory = "data"

    listOfDirsThatContainCTScans = []

    os.makedirs("Results", exist_ok = True)

    for root, dirs, files in os.walk(fileDirectory):
        for dir in dirs:
            currentDir = os.path.join(root,dir)
            dirContainsCTScans:bool =
                checkIfFolderContainsCTScans(
                currentDir)
            if(dirContainsCTScans):
                listOfDirsThatContainCTScans.append(
                    currentDir)

    print(listOfDirsThatContainCTScans)

    x:int = 0
    for folderThatContainsCTScans in
        listOfDirsThatContainCTScans:
        nameForSTLFile:str = f"Results/new_mesh_{x}.
            stl"
        try:
            generateSTLFromFolderOfCTDicoms(
                folderThatContainsCTScans,
                nameForSTLFile)
        except Exception as ex:
            print(ex)
        x+=1
    return

if __name__ == "__main__":
    main()
```

Acknowledgments

The author thanks Alison Carrington and James Snover for sharing their extensive knowledge.

References

Articles

[HKR93] D.P. Huttenlocher, G.A. Klanderman, and W.J. Rucklidge. "Comparing images using the Hausdorff distance". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15.9 (1993), pp. 850–863. DOI: 10.1109/34.232073.

[Fed+12] Andriy Fedorov et al. "3D Slicer as an image computing platform for the Quantitative Imaging Network". In: *Magnetic Resonance Imaging* 30.9 (2012), pp. 1323–1341. DOI: 10.1016/j.mri.2012.05.001. URL: https://www.slicer.org/.

[Cla+13] Kenneth W. Clark et al. "The Cancer Imaging Archive (TCIA): Maintaining and Operating a Public Information Repository." In: *J. Digit. Imaging* 26.6 (2013), pp. 1045–1057. URL: http://dblp.uni-trier.de/db/journals/jdi/jdi26.html#ClarkVSFKKMPMPTP13.

[CETS14] H. Carter Edwards, Christian R. Trott, and Daniel Sunderland. "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns". In: *Journal of Parallel and Distributed Computing* 74.12 (July 2014). ISSN: ISSN 0743-7315. DOI: 10.1016/j.jpdc.2014.07.003. URL: https://www.osti.gov/biblio/1106586.

[Por+14] N. Porz et al. "Multi-modal glioblastoma segmentation: man versus machine". In: *PLoS One* 9.5 (2014), e96873. DOI: 10.1371/journal.pone.0096873. URL: https://doi.org/10.1371/journal.pone.0096873.

[JA+15] Javier Juan-Albarracín et al. "Automated glioblastoma segmentation based on a multiparametric structured unsupervised classification". In: *PloS One* 10.5 (2015), e0125143. DOI: 10.1371/journal.pone.0125143. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC4433123/.

[TJ17] A. F. Tamimi and M. Juweid. "Epidemiology and Outcome of Glioblastoma". In: (2017). Ed. by S. De Vleeschouwer. PMID: 29251870. URL: https://www.ncbi.nlm.nih.gov/pubmed/29251870.

[Com19] Penguin Computing. "SNL/NNSA CTS-1 Attaway - Tundra Extreme Scale, Xeon Gold 6140 18C 2.3GHz, Intel Omni-Path". In: (2019). URL: https://top500.org/system/179777/.

[SMI20] David Michael Smith, David J. Martinez, and Trevor Irwin. "Cooling Performance Testing of Attaway's Negative Pressure CDU". In: (June 2020). DOI: 10.2172/1763558. URL: https://www.osti.gov/biblio/1763558.

[Wol+20] Adrian Wolny et al. "Accurate and versatile 3D segmentation of plant tissues at cellular resolution". In: *eLife* 9 (2020). Ed. by Christian S Hardtke et al., e57613. ISSN: 2050-084X. DOI: 10.7554/eLife.57613. URL: https://doi.org/10.7554/eLife.57613.

[Rie21] Elyse Rier. "GENERATION AND SEGMENTATION OF 3D MODELS OF BONE FROM CT IMAGES BASED ON 3D POINT CLOUDS". In: (2021). URL: http://hdl.handle.net/11375/27270.

[Sch+21] Lisa Schneider et al. "MedmeshCNN - Enabling meshcnn for medical surface models". In: *Computer Methods and Programs in Biomedicine* 210 (2021), p. 106372. ISSN: 0169-2607. DOI: https://doi.org/10.1016/j.cmpb.2021.106372. URL: https://www.sciencedirect.com/science/article/pii/S0169260721004466.

[Tro+22] Christian R. Trott et al. "Kokkos 3: Programming Model Extensions for the Exascale Era". In: *IEEE Transactions on Parallel and Distributed Systems* 33.4 (2022), pp. 805–817. DOI: 10.1109/TPDS.2021.3097283.

[Ba24] Marta Bonada and et al. "Deep Learning for MRI Segmentation and Molecular Subtyping in Glioblastoma: Critical Aspects from an Emerging Field". In: *Biomedicines* 12.8 (Aug. 2024), p. 1878. DOI: 10.3390/biomedicines12081878.

Software-Packages

[Hen02] A. Henderson. *ParaView Guide, A Parallel Visualization Application*. 2002. URL: https://www.paraview.org/.

[Fed+12] Andriy Fedorov et al. "3D Slicer as an image computing platform for the Quantitative Imaging Network". In: *Magnetic Resonance Imaging* 30.9 (2012), pp. 1323–1341. DOI: 10.1016/j.mri.2012.05.001. URL: https://www.slicer.org/.

[Red+16] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91. URL: https://arxiv.org/abs/1506.02640.

[Com18] Blender Online Community. *Blender - a 3D modelling and rendering package*. Stichting Blender Foundation, Amsterdam: Blender Foundation, 2018. URL: http://www.blender.org.

[Wu+19] Yuxin Wu et al. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

[Wol+20] Adrian Wolny et al. "Accurate and versatile 3D segmentation of plant tissues at cellular resolution". In: *eLife* 9 (2020). Ed. by Christian S Hardtke et al., e57613. ISSN: 2050-084X. DOI: 10.7554/eLife.57613. URL: https://doi.org/10.7554/eLife.57613.

[Sch+21] Lisa Schneider et al. "MedmeshCNN - Enabling meshcnn for medical surface models". In: *Computer Methods and Programs in Biomedicine* 210 (2021), p. 106372. ISSN: 0169-2607. DOI: https://doi.org/10.1016/j.cmpb.2021.106372. URL: https://www.sciencedirect.com/science/article/pii/S0169260721004466.

[Sha+24] Abdelrahman Shaker et al. *UNETR++: Delving into Efficient and Accurate 3D Medical Image Segmentation*. 2024. arXiv: 2212.04497 [cs.CV]. URL: https://arxiv.org/abs/2212.04497.

WEB RESOURCES

[Hea25] Siemens Healthineers. *MAGNETOM Terra*. 2025. URL: https://www.siemens-healthineers.com/en-us/magnetic-resonance-imaging/7t-mri-scanner/magnetom-terra.