Bachelor Thesis

AutoSeg: Implementation of a Machine-Learning based Medical Image Segmentation Tool in the Visian Editor

Author
Richard Keil

Supervisor
Prof. Dr. Christoph Lippert

Co-Supervisor
Sumit Shekhar

Digital Health & Machine Learning Group

August 24, 2023

Acknowledgements

I would like to thank Prof. Dr. Christoph Lippert and everyone from the Digital Health and Machine Learning group for their support during the bachelor project. Special thanks go to my supervisor, Sumit Shekhar, for his valuable feedback and guidance, which greatly benefited this thesis. I also extend my gratitude to my bachelor team, consisting of Tony Arnold, Daniel de Jonge, Konrad Gerlach, Tim Riedel, Johanna Schlimme, and Janis Wehen, for their mutual support and productive collaboration. Moreover, thanks go to Jonas Kordt of the Visian team for his technical advice and support for the implementation of the AutoSeg tool, as well as Tobias Klein from Max-Delbrück-Centrum for his feedback. Furthermore, I am very grateful to all user study participants whose contributions were essential to this thesis. Last but not least, I thank my family and friends for their continued support throughout this journey.

Abstract

Medical images hold valuable information for clinical tasks such as patient diagnosis or treatment tracking. However, creating segmentations for these images is time-consuming and requires domain expertise. While fully automated segmentation methods exist, they are often not accurate enough for clinical applications and lack user interactivity. Visian, a modern web-based medical image editor, allows researchers and clinicians to create segmentations manually. However, Visian currently only supports algorithmic segmentation methods and does not leverage machine learning. Furthermore, Meta recently released the Segment Anything Model (SAM), a foundational deep learning model that achieves state-of-the-art results on natural image segmentation tasks. In this thesis, we implement a novel semi-automatic segmentation tool in Visian called AutoSeg, which is based on SAM. AutoSeg allows users to interactively provide a bounding box or point inputs for a target region in a medical image and receive a segmentation prediction. Besides detailing the implementation of AutoSeg, we evaluate the tool by comparing it to existing methods, conducting a user study, and performing a run-time analysis. We find that for easy segmentation tasks, AutoSeg outperforms traditional segmentation methods in Visian and other tools regarding speed and accuracy. For more complex tasks, AutoSeg provides a good initial segmentation that the user can further refine manually. We conclude that AutoSeg is a valuable addition to Visian and shows the potential of foundational models for medical image segmentation.

Zusammenfassung

Medizinische Bilder enthalten wertvolle Informationen für klinische Aufgaben wie die Patientendiagnose oder -behandlung, jedoch ist die Erstellung von Segmentierungen für diese Bilder zeitaufwändig und erfordert Fachwissen. Obwohl vollautomatische Segmentierungsmethoden existieren, sind diese für klinische Anwendungen oft nicht genau genug und bieten Nutzenden keine Interaktivität. Forschende und Mediziner:innen können hingegen manuelle Segmentierungen mit Tools wie Visian erstellen, einem modernen webbasierten Editor für medizinische Bilder. Visian unterstützt derzeit allerdings nur algorithmische Segmentierungsmethoden und nutzt kein maschinelles Lernen. Kürzlich hat Meta das Segment Anything Model (SAM) veröffentlicht, ein Deep-Learning Basismodell, welches bei der Segmentierung natürlicher Bilder Spitzenergebnisse erzielt. In dieser Arbeit implementieren wir ein neues halbautomatisches Segmentierungswerkzeug in Visian namens AutoSeg, welches auf SAM basiert. AutoSeg ermöglicht es Nutzenden, interaktiv einen Rahmen oder Fokus-Punkte für eine Zielregion in einem medizinischen Bild anzugeben und eine Segmentierungsvorhersage zu erhalten. Neben einer detaillierten Beschreibung der Implementierung von AutoSeg evaluieren wir das Tool mittels einem Vergleich mit bestehenden Methoden, einer Nutzerstudie und einer Laufzeitanalyse. Wir stellen fest, dass AutoSeg bei einfachen Segmentierungsaufgaben traditionelle Methoden in Visian und anderen Tools in Bezug auf Geschwindigkeit und Genauigkeit übertrifft. Für komplexere Aufgaben liefert AutoSeg eine gute Ausgangssegmentierung, welche Nutzende manuell weiter verfeinern können. Wir kommen zu dem Schluss, dass AutoSeg eine wertvolle Erweiterung von Visian ist und das Potenzial von Basismodellen für die Segmentierung medizinischer Bilder aufzeigt.

Contents

List of Figures

List of Tables

Acronyms

API Application Programming Interface. 31

CNN Convolutional Neural Network. 15

CT Computed Tomography. 5, 8

DICOM Digital Imaging and Communications in Medicine. 20

DL Deep Learning. 8

GUI Graphical User Interface. 5, 10

LLM Large Language Model. 23

ML Machine Learning. 8

MRI Magnetic Resonance Imaging. 5, 8

NIfTI Neuroimaging Informatics Technology Initiative. 20

NLP Natural Language Processing. 23

ONNX Open Neural Network Exchange. 5, 35

SAM Segment Anything Model. 5, 8

WASM WebAssembly. 35

1 Introduction

1.1 Motivation

Medical images hold a variety of valuable information about individual patients and the human body in general. Segmentations of such images (see figure 1.1) can be used for surgical planning, tumor detection, volumetric observation of organs, classification of blood cells, and more [5]. Hence, the segmentation of such images is a widely studied problem in medical image analysis.

However, segmenting medical images is a tedious and time-consuming task, which usually has to be performed by medical experts. Therefore, many methods have been developed to support the user in creating accurate segmentations. Many of them are based on algorithmic approaches, while recently, more and more of them leverage Machine Learning (ML) technology, mainly Deep Learning (DL) models.

DL models that assist the user during the segmentation process are usually trained on specific tasks, such as brain tumor segmentation [57] or lung segmentation [65] for Magnetic Resonance Imaging (MRI) scans. Because of their specialized architecture and training, adapting them to new tasks and datasets is challenging.

Recently, Meta released the Segment Anything Model (SAM) as a pre-trained foundational model for natural image segmentation [37]. This model is trained on a large and diverse dataset of natural images and can quickly create segmentations for various shapes and objects in images (see figure 1.2).

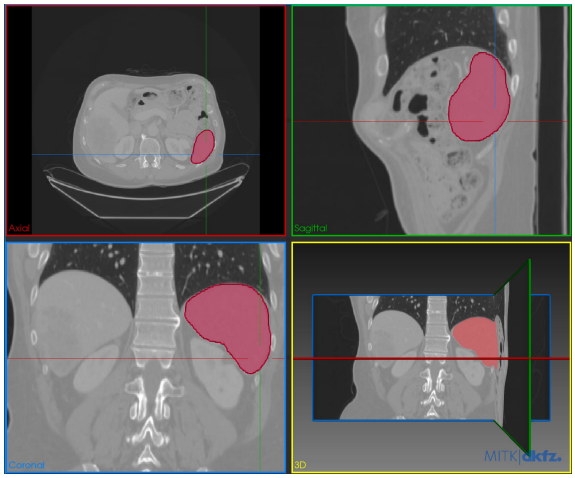


Figure 1.1: Example segmentation of the spleen in a Computed Tomography (CT) scan.

Figure 1.2: Input image and segmentations predicted by SAM [37] (photo by Annika Treial).

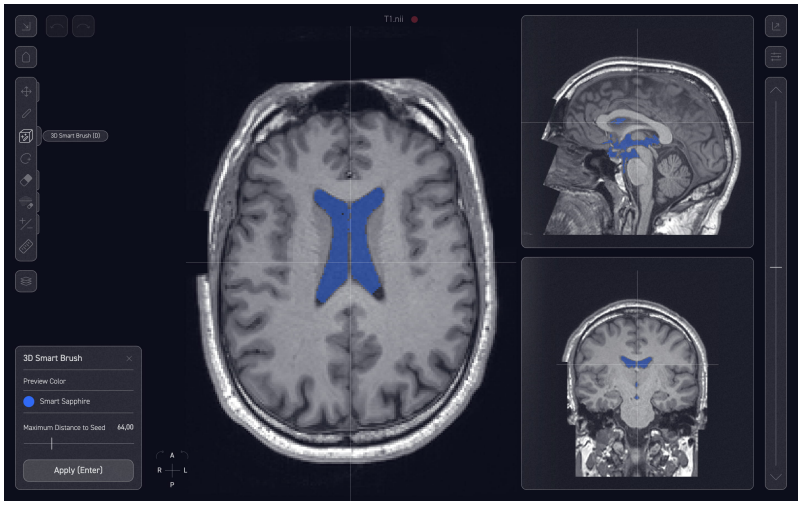


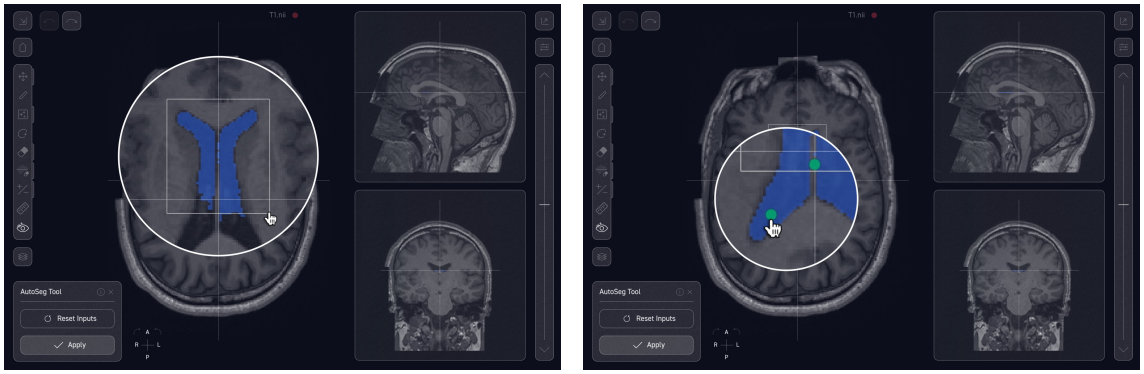
Figure 1.3: Viewing an MRI brain scan in the Visian editor.

During the bachelor project, we worked with the Visian editor (short "Visian"), which is a modern and user-friendly tool for viewing and editing medical image segmentations (see figure 1.3). It offers a variety of algorithmic tools to assist the user in the segmentation process but usually requires manual configuration for good results.

Therefore, it is worthwhile to investigate how SAM can be integrated to assist the user in the segmentation process within Visian.

1.2 Problem Statement

Obtaining segmentations for medical images is crucial for patient diagnosis and treatment, but also to build labeled datasets that can be used in research to train DL models. Because segmenting medical images requires considerable manual effort by domain experts, it is desirable to accelerate this process and support the user in creating accurate segmentations. So far, SAM has only been trained on natural images. However, since it shows impressive results in this domain, it is interesting



(a) Providing a bounding box leads to a prediction. **(b)** Adding foreground points refines the prediction.

Figure 1.4: Using the AutoSeg tool to generate a segmentation (areas of interest enlarged).

to investigate whether its capabilities can be extended for medical images. Hence, we want to determine if and how SAM can be integrated into the Visian editor to assist the user in the medical image segmentation process.

1.3 Approach and Contributions

In this thesis, we design and implement a new "AutoSeg" tool within Visian that can quickly generate helpful segmentation predictions for a given medical image. This tool is based on SAM and allows the user to iteratively guide the segmentation process until a satisfactory result is achieved, leveraging SAM's capabilities obtained through the training on non-medical images. AutoSeg allows the user to draw a bounding box around areas of interest or click to place foreground and background points, guiding the tool to create an initial prediction. When a prediction is shown as a preview, the user can either accept it or refine the segmentation by adjusting their previous inputs (see figure 1.4).

We detail relevant requirements for the AutoSeg tool that are used to steer and evaluate our implementation. Based on that, we also explore different options for the system architecture underlying the new tool and find a suitable Graphical User Interface (GUI) design. We evaluate the new tool by comparing it to existing methods and show that it can outperform traditional approaches in terms of speed, accuracy, and usability.

1.4 Thesis Structure

We first provide relevant background information in chapter 2 concerning different categories of medical image segmentation and existing tools for this task. Next, we provide a conceptual overview in chapter 3 about the Visian editor, SAM and its application to medical images. In chapter 4, we detail the requirements for the AutoSeg tool and discuss what options we consider for the system architecture and GUI design, including the eventually chosen approach. Chapter 5 describes our evaluation of the new tool, including a qualitative evaluation comparing it to existing methods, a

user study focusing on usability aspects, and a run-time performance evaluation. In chapter 6, we discuss the results and limitations of the implementation. Finally, chapter 7 provides an outlook on potential future work, and chapter 8 concludes the thesis by summarizing our contributions.

2 Background

Medical images are invaluable sources of information that allow performing patient-centered tasks, such as treatment planning and diagnosis, as well as broader research endeavors, such as training machine learning models. Segmentation labels must be created to utilize this information and delineate specific structures within these images. This chapter explores various methods and tools relevant to this segmentation process, allowing us to understand how the new AutoSeg tool can be helpful.

Segmentation in medical imaging can be broadly classified into three categories: manual, fully-automatic, and semi-automatic segmentation [63]. These methodologies are not mutually exclusive but form a spectrum ranging from complete user control to full automation. We will examine each approach in more detail to determine how an assistive tool for image segmentation, such as the AutoSeg tool, fits within this spectrum and how it can enhance the segmentation process.

2.1 Manual Segmentation

Manual segmentation represents the most fundamental approach, where the user, often a clinician, must manually classify the pixels or voxels of the target structure within the image [5, 63]. Utilizing dedicated software, users may employ pen, selection, or flood fill tools to create segmentations, similar to techniques found in typical photo editing software. For 3D medical images, this is often conducted slice by slice [20, 60, 75, 77].

Thanks to the specific domain knowledge of medical experts, manual methods can be executed with high precision, providing good controllability over the segmentation process. However, this approach faces two main challenges.

First, manual segmentation is inherently time-consuming. The time required for an expert to segment a complete dataset manually cannot be shortened, as each image must be segmented individually [58]. For instance, the accurate segmentation of a single MRI scan of a polycystic kidney might require up to 90 minutes [38]. Moreover, the time needed for segmentation scales with the number and complexity of images, which becomes problematic as medical imaging gets more accessible and the resulting data more complex. As a result, manual segmentation of large datasets is often unfeasible [58].

Second, the intra- and inter-observer variability of segmentations poses a significant problem. Intra-observer variability describes the variability of segmentations of the same image by the same user. This might occur when the user cannot reproduce the same segmentation multiple times, for example, due to fatigue or lack of concentration. On the other side, inter-observer variability

describes the variability of segmentations of the same image by different users. This can result from different users having disparities in domain knowledge or conflicting opinions on segmenting a specific target region [14].

Such variability can undermine the reliability of segmentations [48] and necessitate concerted efforts to establish a consensus on what constitutes the "correct" segmentation [31]. This is especially significant when manual segmentations are employed as ground truth for training ML models.

These obstacles have limited the widespread adoption of manual segmentation in clinical practice [48, 58, 63]. Nevertheless, users of automatic tools might resort to manual corrections to refine automatically generated segmentations [21].

In the context of the new AutoSeg tool, the aim is to minimize the need for manual correction of predicted segmentations. This goal aligns with requirement R5 introduced in section 4.1, which states that the AutoSeg tool should reduce the user interactions required.

2.2 Fully-Automatic Segmentation

Fully-automatic segmentation represents the other side of the spectrum of segmentation methods. In this approach, the user simply provides the image to be segmented, and the tool generates the segmentation without any further input or guidance from the user.

In recent years, DL based methods have strongly influenced the field of medical image segmentation, already being able to or getting close to generating segmentations indistinguishable from manual segmentations [38, 43, 56]. For example, specialized state-of-the-art ML models have shown exceptional performance for segmenting MRI scans of the kidney [38], the prostate [26], and brain tumors [27]. These methods are well-suited for applications that do not involve user interactions and do not require clinical precision, such as tracking organs in real-time for visualization purposes [29].

DL training approaches can broadly be classified as unsupervised and supervised methods. Unsupervised techniques are attractive as they do not require labeled training datasets and learn patterns from raw data. While they are less suited to directly produce image segmentations due to the absence of domain expert-provided labels, they can be used in tasks like image classification or clustering [19]. Applications utilize unsupervised learning either with traditional statistical methods [11] or self-supervised learning, wherein the model generates labels on its own, for instance, by predicting obscured parts of an image. Having learned general features of the data, such models can then serve as foundational models for supervised learning strategies [28].

In contrast, supervised approaches rely on labeled images, providing more accurate results and thus enjoying widespread use in the medical domain [19, 71]. However, they require substantial amounts of training data. A typical example of a supervised method utilized for fully-automatic segmentation is the U-Net architecture [59]. U-Nets are based on fully convolutional neural networks [45], where the input image is passed through a series of contracting convolutional layers, resulting in an encoded representation of the raw data. This encoding can then be upsampled to generate a pixel-wise segmentation of the original image [45]. U-Nets further modify this architecture by

introducing multiple upsampling layers and corresponding connections to previous contracting layers. This addition allows the network to use higher resolution information to generate more precise segmentations in expanding layers [59].

Fully-automatic segmentation approaches are usually fast and provide reproducible results [17]. For example, models like U-Net can segment a standard 512×512 MRI scan in under a second [59]. Since they do not involve user input when generating segmentations, they are immune to the intra- and inter-observer variability that manual methods inherit [48].

However, supervised models require a large amount of task-specific labeled training data to achieve good results. Since creating labeled data for the medical domain is time-consuming, this can be a significant challenge [17]. For example, training a model to segment polycystic kidneys required a dataset of 2000 manually segmented cases [38]. Secondly, the diversity in medical imaging data, resulting from varied scanner settings, patient demographics, and imaging protocols, often hinders the generalization capabilities of these models [47, 62, 76]. This requires the creation of custom pre-processing pipelines and architecture tweaks for optimal performance [51]. Since the clinician cannot interact with the segmentation process, models cannot simply be adapted to a new patient or population [62].

Further complicating their adoption is the inherent lack of transparency of DL models, which is especially important in the medical domain, where users need to be able to trust the results of the model [10]. By definition, fully-automatic approaches do not allow users to adjust or refine a generated segmentation. This prevents users from building intuition on how the model works and does not allow correcting potential mistakes [62].

Lastly, although some fully-automatic segmentation models already provide state-of-the-art results, they cannot be easily deployed for diagnosis in clinical practice [48, 62]. For example, EU law considers software that can be used for medical diagnosis or therapy as a medical product [25] and hence requires regulatory approval [9].

Nevertheless, the methods designed for fully-automatic segmentation have found utility as foundational components for semi-automatic segmentation systems. For example, the previously described approach of self-supervised learning is part of the system architecture of SAM [37], which is the foundation for our new AutoSeg tool.

2.3 Semi-Automatic Segmentation

Semi-automatic segmentation, also known as “interactive segmentation”, is positioned between manual and fully-automatic segmentation techniques [5]. This approach involves users supplying manual input, such as adjusting algorithm parameters, adding coordinate points of interest, drawing bounding boxes around relevant areas, or freehand lines on the image, upon which the tool generates a preliminary segmentation. Users can then further refine or accept the generated segmentation [29, 62].

Interactive segmentation tools can be broadly divided into algorithmic solutions and DL based solutions. Algorithmic solutions include methods such as Thresholding, Graph Cuts, or Region-growing. For Thresholding, users provide upper and lower bounds for intensity values to create a binary mask. This mask often necessitates further refinement to remove unwanted parts from

the segmentation [66]. In Graph Cut algorithms, the image is represented as a graph, nodes being pixels and edges the similarity between two pixels, for example, based on brightness values. Users can then provide a seed point for the foreground and background, for which the algorithm finds the optimal cut through the graph that separates the foreground from the background while keeping similar pixels together [6]. Region-growing techniques allow the user to provide a threshold value and seed points that should be part of the foreground or background. The tool then grows the provided seeds iteratively by adding pixels similar to neighboring pixels. This process is repeated until the threshold value is reached and no change is made to the resulting segmentation [73].

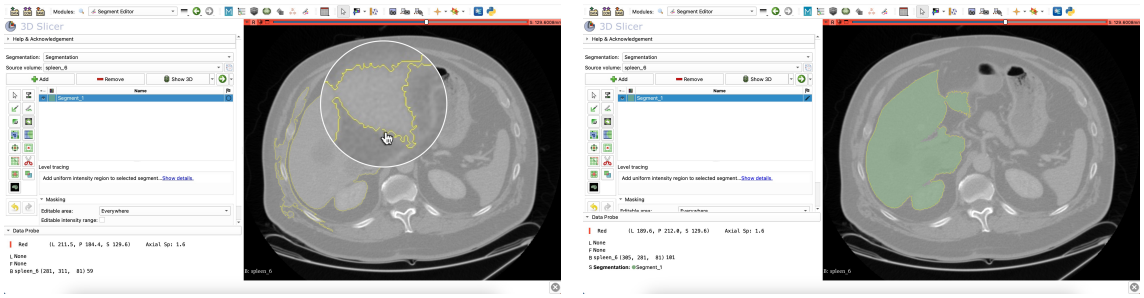
There are also a variety of DL-based solutions, including "Scribble CNNs" or click-based interactive approaches. In Scribble CNNs, a Convolutional Neural Network (CNN) model is initially trained to generate a preliminary segmentation based on a medical image cropped by a bounding box. To enhance the generated segmentation, users can then provide pixel strokes ("scribbles") indicating foreground and background regions. These scribbles, along with the initial segmentation, are used to create a weight map, which is used for the training of a secondary CNN model. This latter model produces the final and "corrected" segmentation [70].

Click-based approaches like the one introduced by Sakinis et al. [62] employ fully convolutional neural networks to generate segmentations based on input images and user-provided foreground and background clicks. To effectively train and assess these models, human interaction is mimicked by iteratively placing clicks in areas where initial segmentations yield false positives or false negatives. This repeated process enables the model to effectively learn and adapt to user preferences, improving its predictions when the initial segmentation is unsatisfactory.

Semi-automatic segmentation methods offer a balanced approach, combining the advantages of manual and fully-automatic techniques. Unlike manual segmentation, where the user has to delineate every region of interest from scratch, and fully-automatic segmentation, which suffers from a lack of user guidance, semi-automatic methods strike a productive middle ground. Since users only have to provide partial input and the tool generates a substantial portion of the segmentation itself, the time required for segmentation is drastically reduced [62, 70, 73]. Furthermore, semi-automatic approaches are not intended to generate clinically accurate segmentations, so their results do not have to be perfect and can instead be refined by the user.

Compared to fully-automatic approaches, semi-automatic systems are built actively considering users in the workflow [29], making it much easier for users to recover from segmentation failures of the tools. Users can simply redo or correct the segmentation, which is often more difficult with fully-automatic segmentation. This also helps with the previously discussed transparency issue of DL models since users can actively experiment and develop an understanding of how the model decides what to segment [62].

For these reasons, semi-automatic segmentation systems are widely deployed in clinical practice and corresponding tools [29, 62]. The new AutoSeg tool fits well into the category of semi-automatic approaches and aims to provide the benefits mentioned above to the user (for example, in the form of requirement R3).



(a) Hovering over pixels highlights possible contours. **(b)** Clicking applies a contour as segmentation.

Figure 2.1: Using the "Level Tracing" tool in 3D Slicer (areas of interest enlarged).

2.4 Tools for Semi-Automatic Segmentation

Several tools have been developed to view and edit medical images, including functionality for semi-automatic image segmentation. Here, we present three prominent tools, specifically 3D Slicer[1], ITK Snap[2], and the Medical Imaging Interaction Toolkit ("MITK")[3]. For each tool, we detail one mechanism that allows for semi-automatic segmentation. This allows us to identify the strengths and weaknesses of different implementations and structure our requirements (see section 4.1) accordingly.

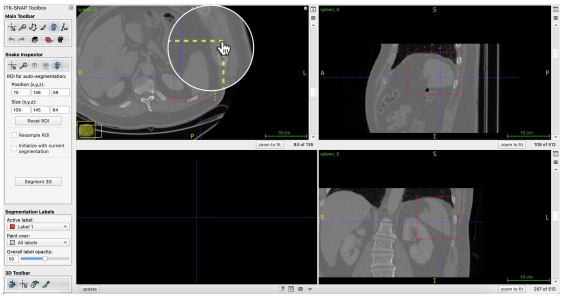
2.4.1 3D Slicer

3D Slicer is a free, open-source software for visualizing and segmenting biomedical images and meshes. Developed by the Surgical Planning Laboratory at the Brigham and Women's Hospital in collaboration with the MIT Artificial Intelligence Laboratory, 3D Slicer has established itself as an essential tool in medical imaging [20, 55].

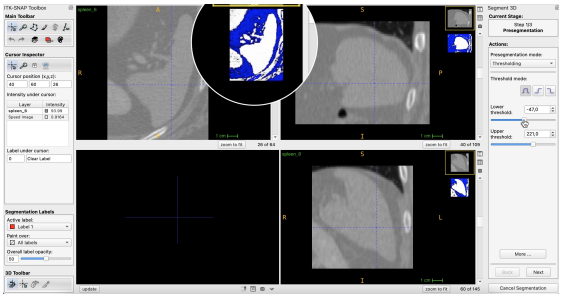
One of its semi-automatic segmentation tools is the "Level Tracing" tool. This basic algorithmic tool proposes a segmentation contour based on a trace of similar-intensity pixels starting at a user-selected point. It first generates a binary boundary map by thresholding the currently hovered pixel brightness. After that, the tool traces the boundary of this binary map, iteratively checking neighboring pixels and adding them to the contour if they align with the boundary shape.

The user can first hover over arbitrary points in the image when using the Level Tracing tool. The tool then automatically executes the previously described algorithm and displays the resulting contour as a yellow line as depicted in figure 2.1a. The user can then click to accept the proposed outline or move the cursor to receive a contour based on the new mouse position (see figure 2.1b).

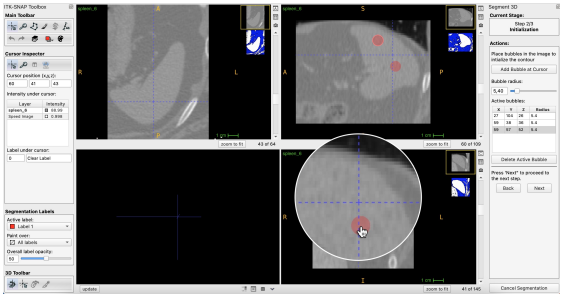
[1] https://www.slicer.org
[2] http://www.itksnap.org
[3] https://www.mitk.org



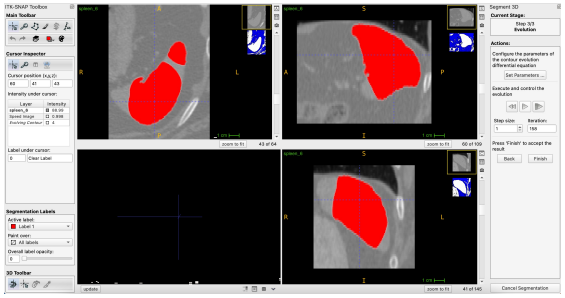
(a) Defining the target region with a bounding box.



(b) Creating a "feature image" by thresholding.



(c) Placing bubbles that will be used as seed points.



(d) Observing the expanding seed points during the evolution step.

Figure 2.2: Using the "Snake" tool in ITK Snap (areas of interest enlarged).

The primary advantages of this tool are its intuitive design and fast computation speed. However, the tool requires good contrast between the object and the background to achieve satisfactory results. Furthermore, it struggles with more intricate shapes without clearly defined edges. When applying the tool to such objects, the resulting outline is usually not smooth, and the segmentation requires manual correction.

2.4.2 ITK-Snap

Another widely used software in the domain of medical image segmentation is ITK-Snap. Developed at the University of North Carolina, it only provides a limited set of tools but therefore offers an intuitive user experience, focusing on simplicity and efficiency [75].

Among its features is the semi-automatic segmentation tool known as "Snake", which is based on the "Active Contour" algorithm [36]. The approach introduces the concept of a "snake", a flexible spline that reacts to simulated forces that attract it to image features like edges. This requires the image features to be transformed into a value map indicating whether an area is likely part of the contoured object. The user then initializes the snake spline by drawing or clicking, providing a rough outline of the initial starting contour. The algorithm then iteratively inflates this spline based on the underlying values of the previously generated value map until a satisfactory segmentation is achieved.

The "Snake" tool in ITK-SNAP follows a four-stage process. In the first stage, the user selects a region of interest where the tool will operate, ignoring all other areas to save computation resources (see figure 2.2a). This is followed by the "pre-segmentation" stage, where the original image is

transformed into a value map in the range [-1,1], referred to as the "feature image". Here, values of -1 correspond to the background, and values of 1 correspond to the target area. The user can apply various techniques to create this feature image based on what is most suitable for the current image (see figure 2.2b). For example, they can use a thresholding approach by adjusting sliders to mark the target area with positive values and the background with negative values. Other options include classification, where an ML model is used to label pixels according to the requirements of the feature image, or clustering, where voxels with similar intensity are assigned the same feature image value. Lastly, users can also choose an edge-based approach, where the feature image is generated by detecting edges in the image. In this case, edge areas are assigned values of 0, and the non-edge regions are designated as 1, thereby preventing the snake spline from crossing edges. The third "initialization" stage involves the user placing one or more "bubbles" of configurable size within the 3D space (see figure 2.2c). The outlines of these bubbles are then used to initialize the snake contour. In the fourth and final "evolution" stage, the bubbles are iteratively expanded based on the previously defined feature map, the speed of which is controlled by the underlying feature image. Consequently, areas with high values are quickly filled, while regions with low values are either ignored or slowly filled (see figure 2.2d). This stage also allows the user to pause or reset the evolution process.

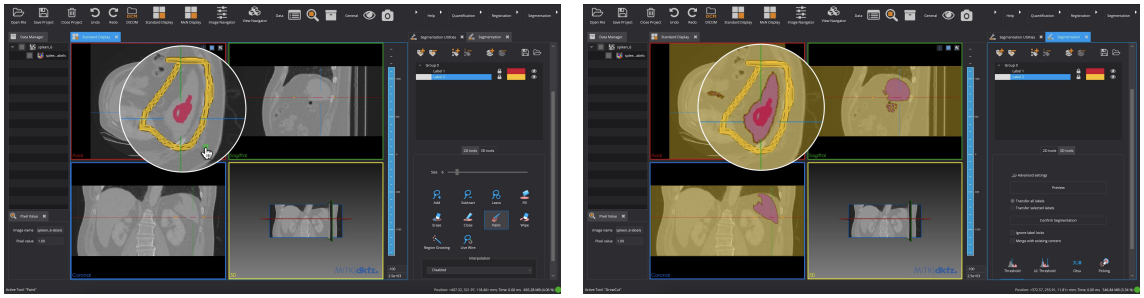
The tool can adapt well to different scenarios because of its high configurability: the user can control various aspects, such as how the feature map is created or where bubbles are placed. However, the segmentation process can be time-consuming, given the multiple steps involved. Usability issues also emerge in certain situations, for example, during the evolution step, where the user can only reset the entire process, not revert step by step.

2.4.3 Medical Imaging Interaction Toolkit

The Medical Imaging Interaction Toolkit (MITK) is another free and open-source software designed for interactive medical image processing, developed by the German Cancer Research Center [72]. MITK offers a collection of semi-automatic segmentation tools to suit various segmentation needs. Followingly, we focus on the functionality and implementation of the "Grow Cut" tool provided by MITK.

The Grow Cut tool is based on the identically named "GrowCut" algorithm, described by Vezhnevets et al. [69]. It considers the image as a cellular automaton, where every pixel functions as a cell. The user starts by providing initial seeds, typically by marking particular pixels, which are then used to initialize the automaton cells. Once initiated, the automaton iteratively updates the cells based on the values of neighboring cells, a process resembling a simulated "attack" of cells against each other. Each cell's attack "strength" is a factor of the value disparity between the cells, such as color or brightness. The iterative process leads to a convergence of cells into areas of similar values based on the initial seeds. The algorithm completes when the cells stabilize without further change.

Within MITK, employing the Grow Cut tool requires the user to create two segmentation layers corresponding to the foreground and background. Using standard Brush tools, the user can then approximately mark areas of the image as either foreground or background, serving as seed regions for the algorithm (see figure 2.3a). Subsequently, the GrowCut tool can be selected, and by clicking "Preview", the user initiates the segmentation generation. When tested, the computation took



(a) Marking foreground and background areas on two layers.

(b) Previewing the resulting segmentation.

Figure 2.3: Using the "Grow Cut" tool in MITK (areas of interest enlarged).

approximately one minute, after which a preview of the created segmentation was displayed (see figure 2.3b). The user can then confirm the preview or revert to the Brush and Eraser tool to correct the base segmentation if undesired areas were segmented.

The Grow Cut tool within MITK delivers impressive results with little input and even provides 3D segmentation capability. However, using the tool requires additional setup, such as creating initial segmentation layers. The tool may not be fast enough for real-time segmentation without adequate hardware. We also observe minor usability issues, such as the preview vanishing when returning to the Brush tool, making it challenging to determine which exact voxels need correction.

3 Conceptual Overview

Having looked at different approaches to medical image segmentation and compared traditional tools in the domain, we will now provide a conceptual overview of the elements that will form the basis for the new AutoSeg tool, namely the Visian editor and SAM.

3.1 Medical Image Editor "Visian"

The Visian editor[1] is a web-based application designed to facilitate the viewing and editing of medical images. It was developed as a modern alternative to existing medical image editors like ITK-Snap and 3D Slicer (discussed in section 2.4). Particular focus was put on its intuitive user interface, making it accessible to a wide range of users, including non-technical individuals like clinicians.

Visian handles both simple 2D images and more specialized 3D medical images, supporting the import of `.nifti` and `.dicom` files[2]. The editor allows users to navigate through the slices of 3D images, view them in different planes, or display them as 3D models. It can also handle layers to accommodate the various image modalities and provides multiple tools to create and edit segmentations.

In terms of architecture, Visian operates as a standalone browser application, requiring no additional installation or backend infrastructure. This means users can simply open the application through a public web address and start using it. Since all data is stored directly in the browser, sensitive medical data never has to leave the user's computer.

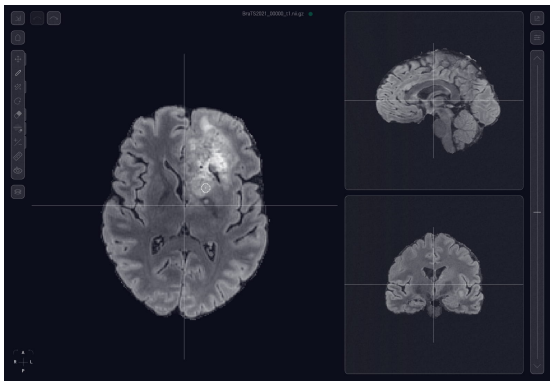
Next, we will detail the user interface, available tools, and the typical user workflow within Visian. This will inform the requirements and design of the new AutoSeg tool.

3.1.1 User Interface

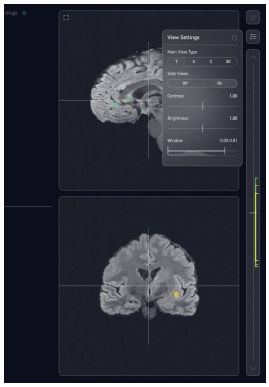
Visian presents a one-page user interface with a primary view in the center. When dealing with three-dimensional data in medical imaging, one often relies on three distinct view planes to locate anatomical structures. These view types are called "Transverse", "Sagittal", and "Coronal" planes [53]. The primary view of Visian displays the currently selected image slice, defaulting to the "Transverse" plane with a crosshair indicating the current position in the image (see figure 3.1a).

[1] https://visian.org

[2] The Neuroimaging Informatics Technology Initiative (NIfTI) standard and Digital Imaging and Communications in Medicine (DICOM) standard are both commonly used for storing medical images. While NIfTI is a file format specifically designed for neuroimaging data, DICOM is a more general standard that is used for a variety of medical images [40].



(a) Editor showing transverse view on the left, sagittal and coronal views on the right.



(b) Having opened the view settings.

(c) Displaying the layer menu help text.

Figure 3.1: The user interface of Visian.

The right side of the screen holds the side views, showing the "Sagittal" and "Coronal" planes, which are updated based on the crosshair's position in the main view. On the top-right, an export button allows downloading a ZIP archive of `.nifti` files of the created segmentation layers. The right side also features a button to adjust parameters like view type, contrast, brightness, opacity, various 3D view settings, and a slider for scrolling through slices of a scan. This slider also indicates slices that contain segmentation data (see figure 3.1b).

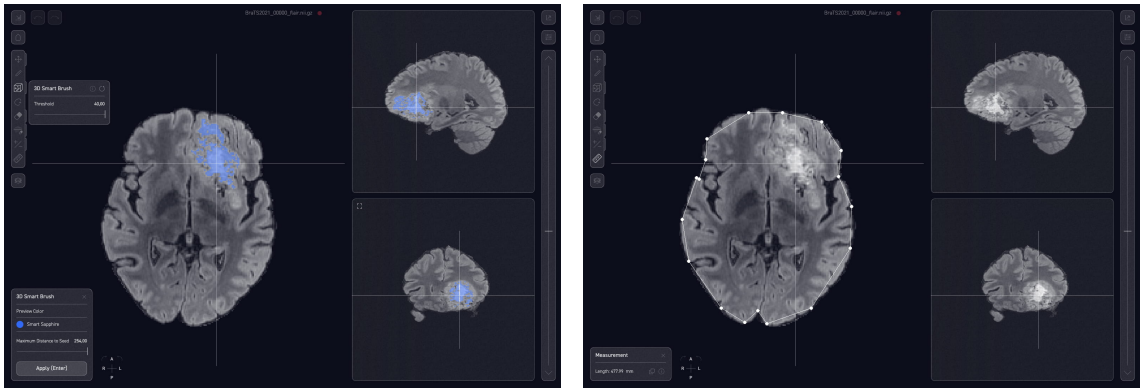
On the left side, users find buttons to import files, open the editor settings, activate various available tools, and toggle the layers menu. The layers button opens up a context menu showing the layer stack, allowing users to re-order layers or change their visibility. Image and segmentation layers are labeled with an image icon and a color dot, respectively. A right-click on a layer opens a context menu for various actions like renaming or deleting the layer. On the top right of the menu, the user can click a help icon to receive additional information or click the cross icon to close the menu (see figure 3.1c).

3.1.2 Available Tools

Visian provides various tools tailored for medical image segmentation and interpretation. Each tool can be activated by clicking on its respective button on the left side of the screen. In the following, we will discuss relevant tools for the thesis.

Brush and Eraser Tools

Visian is equipped with multiple brush and eraser tools, which allow the user to add or remove segmented image areas based on the current cursor position. Users can adjust the brush radius through the tool menu and lock it to the current zoom level. These tools also have enhanced versions, namely the "Smart Brush", "Bounded Smart Brush", and their eraser counterparts. These "smart" brushes extend or erase the segmentation until a specified threshold or brush radius is reached, using the current cursor position as a seed point.



(a) Using the 3D Smart Brush tool. **(b)** Tracing with the Measurement tool.

Figure 3.2: Examples for tools available in Visian.

The "3D Smart Brush Tool" deserves particulars attention: It creates a 3D segmentation for the current image based on a seed point and threshold provided by the user (see figure 3.2a). This seed is set by clicking on a pixel in the main view, which brings up a separate context menu on the bottom left of the screen and shows a blue overlay previewing the result of the operation. The user can adjust the maximum distance from the current seed in the menu. Based on the configuration, the tool then expands the segmentation in 3D until the specified threshold or maximum distance is reached and updates the preview accordingly. When the user is satisfied with the result, they can apply the segmentation to the current layer by clicking the "Apply" button in the context menu. This tool differs from the others in providing an intermediate step of previewing the result before applying it.

The various smart brush tools are currently the most helpful semi-automatic methods available in Visian. However, they still require a lot of user interaction, like drawing or adjusting seed points, since they cannot leverage previously learned knowledge about the image content. This challenge can be addressed using a DL based method like SAM, which we describe in section 3.2.

Measurement Tool

When using the "Measurement Tool", the user creates a path line by repeatedly clicking on the main view, each time creating a node point that will automatically be connected to the previous node on the path. A popup on the bottom left of the screen shows the total length of the created path. The node points are rendered as white dots, connected through a white line as an overlay on the main view (see figure 3.2b). We will orient ourselves on this tool when designing the interface of the new AutoSeg tool (see section 4.3.1).

3.1.3 User Workflow

The workflow in Visian is designed to be intuitive and straightforward, guiding users through a smooth process from import to export.

When the editor is opened for the first time, the user is prompted to import an image. This can be done by clicking the import button and selecting a file or simply by drag-and-drop. Upon loading the file, Visian will automatically add the scan as the main image layer and generate a new segmentation layer. This layer is selected by default, thereby allowing users to directly start segmenting the image with the various tools available. In the process, they can switch views, adjust options such as brightness and contrast, go through slices, and create additional segmentation layers if needed. Once the user completes the editing process, they can export their work. By clicking the export button, they trigger a download of the created segmentation layers as a ZIP archive of `.nifti` files. They can then use the result for further analysis or other purposes.

3.2 Segment Anything Model for Image Segmentation

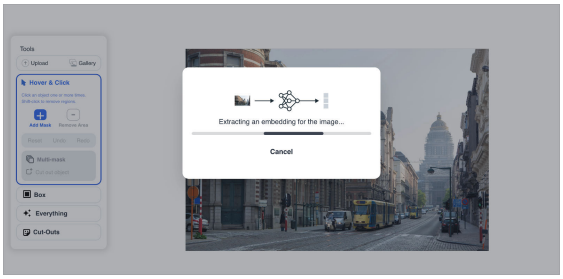
The Segment Anything Model (short "SAM") is an open-source state-of-the-art DL model for generating segmentation masks for 2D natural images, developed by Kirillov et al. [37] at Meta AI Research. As the name suggests, its novelty lies in its ability to segment "anything", i.e. a wide variety of objects and shapes in natural images.

Trained on 11 million images and 1 billion masks, the model not only performs exceptionally on natural images but also transfers well to previously unseen tasks. For example, SAM can be used in systems for edge detection and text-to-mask prediction (generating a mask based on free-form text) without the need for re-training. After obtaining an initial image embedding, generating a mask based on a given input (e.g. a bounding box) can take as little as 50 milliseconds on a CPU in a web browser, making SAM suitable for interactive use cases.

The development of SAM was motivated by the success of large language models (LLMs) in the Natural Language Processing (NLP) space. NLP models like GPT-3 [8] or BERT [30] are trained on a vast corpus of text, which allows them to learn general concepts that can later be applied to more specific tasks. For this reason, these LLMs are also called "foundational models". Instead of having to fine-tune them for specific problem domains, they can be used as is and instead be "prompted" with various instructions at inference time (also called zero-shot learning) [8]. The motivation behind SAM was to create a foundational model for image segmentation that generalizes well and can solve downstream segmentation problems on previously unseen data using a similar prompting approach as LLMs.

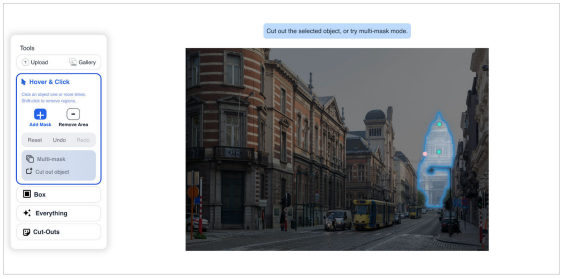
Unlike previous general interactive segmentation models that were only based on foreground and background point clicks [39], SAM can be prompted with a variety of inputs like bounding boxes, points, text, or existing masks. It outperforms previous methods on 23 tested datasets of previously unseen images on single-point mask generation (given the most relevant of 3 generated masks is selected[3]) [37]. Human annotators also rate the quality of generated masks consistently higher than the ones created with previous models [37].

[3] SAM is ambiguity-aware and capable of generating multiple possible masks for one point. Most datasets only provide one mask per point as ground truth, which SAM does not always consider as the most relevant when rating its generated masks.

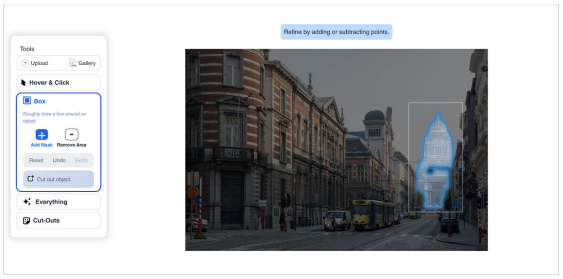


(1) Loading an image and generating an embedding.

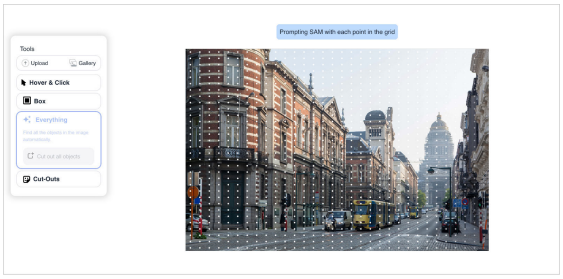
(2) Prompting SAM with a single foreground point.



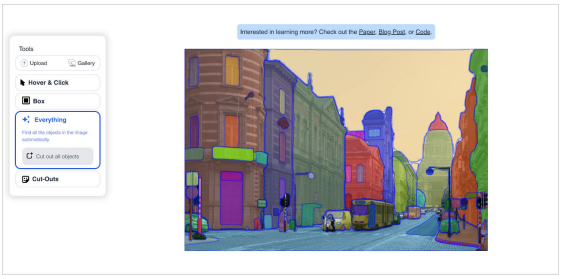
(3) Prompting SAM with two foreground and one background point.



(4) Prompting SAM with a bounding box.



(5) Prompting SAM to generate all masks using a point grid.



(6) Display of the masks found using the point grid.

Figure 3.3: Using SAM in the web demo created by Meta.

Meta has created a web demo[4] that allows exploring the capabilities of SAM in a standard web browser. An example workflow to generate segmentations for an image using this web demo is as follows (see figure 3.3):

1. The user obtains an embedding for the image (see section 3.2.1) by uploading it to a web service. The processing can take up to a few seconds but could also happen locally if the user has sufficient hardware.

2. Next, the user can either draw a bounding box around the object they want to segment, add foreground points that indicate areas that should be part of the mask, or add background points that indicate areas that should not be part of the mask. The user can also prompt for "automatic mask generation", which will instruct SAM to generate all possible masks for the given image (based on a configurable grid of points).

[4] https://segment-anything.com/demo

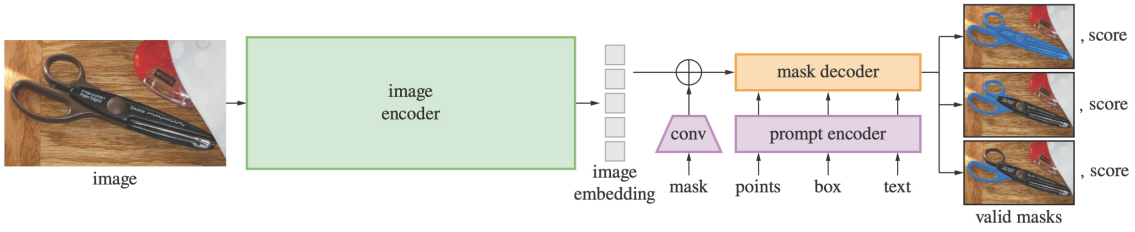


Figure 3.4: Overview of SAM's architecture [37].

3. The model will then instantaneously generate one or multiple masks for the given input. It can optionally output a predicted score and a heuristic measure of the mask quality, the covered pixel area, and the bounding box surrounding the mask.

Meta has released three different pre-trained model versions as checkpoints for SAM: ViT-B with 91 million parameters, ViT-L with 308 million parameters, and ViT-H with 636 million parameters.

3.2.1 Architecture

SAM consists of three main components: an image encoder, a prompt encoder, and a mask decoder (see figure 3.4).

The image encoder generates a vector representation for a given image (also called "embedding"), which only has to happen once. This encoder is a Masked Auto Encoder [35] based on a pre-trained and slightly adjusted Vision Transformer [16] for high-resolution images. It takes a 1024×1024 image as input, requiring larger images to be resized and their shorter side padded to 1024. Output is a $64 \times 64 \times 256$ image embedding. Because of its Vision Transformer architecture, the image encoder is quite computationally intensive.

The prompt encoder generates a 256 embedding for a given prompt, differentiating between "sparse" (bounding box, points, and text) and "dense" (mask) prompts. Sparse prompts are directly encoded (e.g. by summing the point position and a learned encoding indicating if it is a foreground or background point), while for dense prompts, an encoding is generated using a CNN [52] and then summed with the previously obtained image encoding to retain spatial information.

The mask decoder is based on a transformer decoder [2]. It takes the image and prompt embedding as input and generates a 1024×1024 mask, which then needs to be resized to the original image size. It also predicts a score for the mask quality, which can be used to select the best result if multiple masks are generated for one prompt.

Both prompt and image encoder are optimized for inference speed, which allows them to run in amortized real-time on consumer hardware (also see our run-time performance evaluation in section 5.3).

3.2.2 Suitability for Medical Image Segmentation

SAM has demonstrated exceptional performance on natural images, with the authors emphasizing its ability to solve previously unseen downstream tasks. However, the question of whether SAM can also be applied to medical image segmentation remains partly unanswered.

Medical images possess specific characteristics that distinguish them from natural images, including:

- **Dimensionality:** While natural images are only two-dimensional, medical images such as CT or MRI scans are often 3D representations of internal structures of the human body. The third dimension can provide additional clues about regions of interest, which are difficult to capture in 2D.
- **Value range:** Natural images are typically encoded as 8-bit RGB images (values ranging from 0 to 255), whereas medical images have a different value range depending on what they represent. For instance, the DICOM format stores 16-bit integers (ranging from -32768 to 32768) to easily map to the Hounsfield scale used in CT scans.
- **Multiple modalities:** Unlike natural images, which are usually in plain RGB format, medical images can be of different modalities, such as "T1" or "T2". While T1 images highlight fat and protein-rich fluid, T2 images bring out regions with water content more strongly [32]. These modalities can reveal additional information when viewed together, therefore being often bundled in one file.
- **Blurry or unclear edges:** Objects to be masked in natural images (e.g. an apple on a table) usually have sharp edges. In contrast, areas of interest in medical images (e.g. tissue of a brain tumor) often lack clear borders and can be difficult to distinguish from surrounding regions – even if the image resolution is sufficient. This is because changes in organic tissue are often only visible through gradual changes in the image's brightness.

Several studies have evaluated SAM's performance on different medical image datasets. This involves transforming medical images into a format that SAM can process and addresses the "dimensionality" and "value range" characteristics mentioned above. For SAM to process images, they are usually first normalized and then converted to an 8-bit RGB image (value range 0 to 255) [33]. 3D segmentations are typically generated by applying SAM to the 2D slices along the z-axis of an image and then combining the results back into a 3D segmentation [64].

SAM's performance was found to be inferior to segmentation models trained on specific datasets such as 3D liver segmentation using a U-Net [13] or 2D segmentation of polyps in colonoscopy images [67]. Specialized models such as U-Net, U-Net++, Attention U-Net and Trans U-Net showed Dice[5] overlaps that were up to 0.7 higher than Dice overlaps of SAM [64]. However, it should be noted that these models were specifically trained on the respective datasets, while SAM performed zero-shot inference without any fine-tuning.

[5] The Dice coefficient is a normalized measure indicating the similarity between two samples and is commonly used to compare how close an image segmentation matches ground truth. A score of 0 means no overlap between two masks, while a score of 1 means perfect overlap [68].

The quality of the generated segmentations correlates with relatively larger target regions (such as when segmenting the liver), higher contrast and 2D over 3D images [33, 46, 61, 64, 74]. Hence, the performance of SAM depends strongly on the dataset [74]. SAM also tends to produce over-segmentations (i.e. the segmentation is larger than the actual target region) or segmentations of an unintended area within a given bounding box [33]. Nevertheless, SAM outperforms other general interactive segmentation models such as SimpleClick [44] by a vast majority on single-point prompts [39, 46].

Different prompting methods have been evaluated for SAM. The "automatic mask generation"-mode of SAM that tries to find all possible masks is not considered a suitable prompting method since it does not provide semantic labels and generates non-meaningful masks for medical applications. For example, when computing all masks, SAM includes a segmentation of the whole skull in an MRI brain scan because it has a high contrast compared to the black background [33]. Prompts using a bounding box yield the best results [15, 46], to the extent that a single bounding box leads to better results than a prompt with ten individual foreground and background points [61]. The tighter the bounding box encloses the targeted region, the better the segmentation. Added jitter can affect results quite negatively [15].

In conclusion, while SAM shows promise for specialized segmentation tasks, its performance is highly dependent on the dataset and not yet on par with specialized models. However, it performs exceptionally well compared to other general interactive segmentation models, especially when only relying on a few inputs.

4 Implementation

While SAM has shown promising potential to generate high-quality segmentations, its performance is not comparable to models specialized for medical image segmentation. However, its impressive speed and robust understanding of shapes and structures in images make it an ideal candidate for supporting semi-automatic segmentation processes. Even if the initial segmentation it generates is not flawless, its output can easily be corrected with traditional tools, accelerating the segmentation process. Furthermore, Visian is a modern web-based editor for medical image segmentation and, therefore, a good candidate for further enhancement. However, it only supports algorithmic segmentation tools and offers no support for continuously user-guided methods. Hence, we have chosen to integrate SAM into Visian, introducing the new tool called "AutoSeg".

We will now detail the requirements for this integration, discuss the evaluated and selected options for the system architecture, and provide a comprehensive description of the GUI implementation.

4.1 Integration Requirements

We identify the following principal requirements for implementing the AutoSeg tool into Visian.

4.1.1 Functional Requirements

R1: The user must be able to guide the proposed segmentations. This entails the user being able to provide inputs such as bounding boxes or points to guide the tool in predicting segmentations. This feature is critical for semi-automatic segmentation as it permits correcting potential inaccuracies and fosters an understanding of the model's operation (see section 2.3).

R2: The user must have access to a preview of the proposed segmentation. For practical tool guidance, the user needs to preview the initial segmentation prediction. Furthermore, this preview should be non-destructive, i.e., not affect the currently selected segmentation layer.

4.1.2 Non-Functional Requirements

R3: The user must receive immediate feedback upon providing input. Specifically, the predicted segmentation should be visible as soon as the user provides input. Therefore, the tool must be sufficiently fast to deliver real-time feedback to the user.

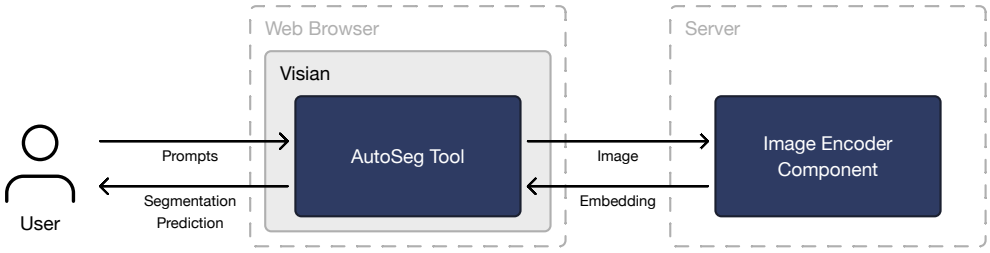


Figure 4.1: General architecture of the AutoSeg tool.

R4: The tool must be easy to understand and learn. The tool interface should be user-friendly and intuitive, creating a pleasant experience for new users. To make learning the new tool easier, AutoSeg should provide helpful operation instructions and seamlessly integrate into the existing workflows of Visian.

R5: The tool must be efficient to operate. Experienced users should be able to use the tool with as few interactions as possible, achieving their desired outcomes in the shortest time possible. This requirement differs from R4, focusing on the needs of users already familiar with the tool instead of first-time users.

4.2 System Architecture

The architecture of the tool integration consists of two integral parts: the image encoder component and the frontend component (see figure 4.1). The image encoder component, operating as a distinct backend service, generates embeddings from a provided image. The counterpart to this service, the frontend component, is contained within the implementation of the AutoSeg tool in Visian. It retrieves the image embedding from the image encoder component and generates segmentation predictions based on prompts provided by the user.

We will first examine the rationale behind selecting this architectural design, followed by a detailed description of the different implementation aspects of the two components.

4.2.1 General Considerations

The architecture of SAM entails a computationally intensive image encoder, a lightweight prompt encoder, and a small mask decoder (see section 3.2.1). The system must first create an image embedding using the image encoder, then encode the user-provided inputs to get a prompt embedding, and subsequently generate a segmentation prediction based on the obtained image and prompt embedding. The critical question is how to integrate these components into Visian efficiently, for which we consider two options.

Frontend-Only Integration The first option is to integrate SAM into Visian so that the entire system can still be run as a frontend-only application. This approach does not require additional infrastructure, such as a backend service that must be hosted and deployed separately. Hence, the user does not need to perform any extra setup to use the tool, and Visian can still be run directly from the browser. On the downside, this environment is restricted by the user's available hardware, such as RAM and CPU. This would result in much longer generation times for the image embeddings, as demonstrated when running the image encoder on a MacBook Pro vs. running the encoder on a high-performance GPU (see performance evaluation in section 5.3). Furthermore, the model checkpoint required to generate image embeddings has a size of 2.4 GB and would need to be downloaded and stored in the user's browser. This option also does not allow setting up the image encoder only once within an organization (for example, on a centralized server of a research institution or company) to provide access to multiple users.

Separation into Frontend and Backend The alternative option splits the system into a frontend and backend, where the image encoder can run remotely on a server while the prompt encoder and mask decoder are integrated into Visian. This option mirrors the setup of Meta's web demo of SAM[1], where the image encoder runs on a server with more resources while processing user input and generating masks is done in the browser. This way, the image encoder can be hosted on dedicated machines with more resources, such as high-performance GPUs in a compute cluster. At the same time, the architecture also allows the image encoder to run as a local server if the user possesses sufficient resources. In both cases, the image encoder model checkpoint must only be downloaded and stored once within the backend component. The frontend component in Visian can then communicate with the image encoder component using `HTTP` requests to obtain the image embedding, which also allows the system to scale more efficiently as multiple instances of the backend component can be run. The primary downside is the additional setup users must do, including selecting a suitable deployment platform, installing the backend component, and configuring Visian to utilize the backend.

Upon considering these options, we opt for the second approach. Because of requirements R3 and R5, we value the improved performance of the image encoder component on dedicated hardware and not having to store the large model checkpoint on the user's system. We also believe the increased setup complexity is acceptable, as the setup only needs to be done once, allowing multiple users to access the system afterward.

The complete architecture of the AutoSeg tool, including subcomponents, can be seen in figure 4.2. First, when activating the AutoSeg tool, the current image data is sent to the image encoder component, which returns the image embedding. We detail this process in section 4.2.2, section 4.2.3 and section 4.2.5. The obtained embedding is then cached in the embedding store, which we describe in section 4.2.4. Simultaneously, the user can specify a bounding box or point prompts, which the prompt encoder will use to generate a prompt embedding. Whenever the prompt inputs change, the tool generates a segmentation prediction through the mask decoder, using the image and prompt embedding as input. This segmentation is then displayed to the user. We will cover the segmentation generation thoroughly in section 4.2.6.

[1] https://segment-anything.com/demo

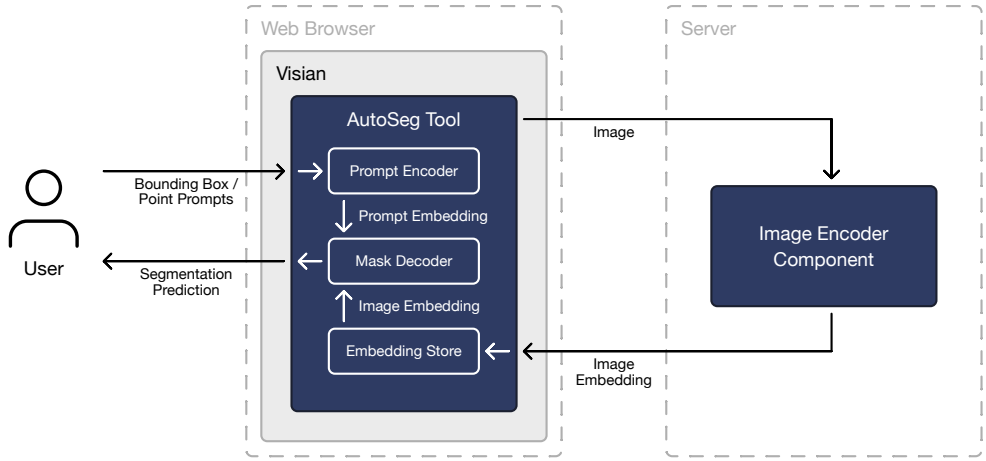


Figure 4.2: Subcomponents of the AutoSeg tool.

4.2.2 Image Encoder Component

The browser frontend needs an effective communication channel to request image embeddings from the backend component. We choose the `HTTP` protocol to facilitate this exchange for its simplicity and broad browser support. This decision necessitates that the backend of the image encoder component should be a web server that presents an accessible Application Programming Interface (API) for the frontend component to utilize.

Since SAM is published as an open-source Python package[2], it is convenient to also use a Python framework for the backend web server. Potential web frameworks suitable for building an API service include Django, Flask, and FastAPI. We select FastAPI[3] for our integration as it is not only lightweight and fast compared to other Python frameworks [7] but has also proven robust in production environments by companies such as Netflix and Uber [23, 50].

The backend component's API exposes a single `POST` endpoint, namely `/embedding`, for generating image embeddings. This endpoint anticipates an `HTTP` request with content type `multipart/form-data` containing the following fields:

- `image`: Raw bytes representing the 2D image (encoded as 8-bit integer values) for which an embedding has to be generated.
- `width`: An integer representing the image width in pixels.
- `height`: An integer specifying the image height in pixels.

Upon successful embedding generation, the endpoint returns the embedding as raw bytes in the response body, using the `application/octet-stream` content type.

[2] https://github.com/facebookresearch/segment-anything
[3] https://fastapi.tiangolo.com

Notably, the SAM package exclusively accommodates 8-bit RGB images as inputs. Since we only operate on one modality of the medical scan, displayed to the user as a greyscale image in Visian, we can only transmit the bytes of this single channel to the backend to reduce the amount of data sent over the network. Hence, we convert the single-channel image to RGB before passing it to SAM. The width and height fields are required to enable the correct reconstruction of the image from the raw bytes, preserving its original dimensions.

As mentioned before, our decision to introduce a separate backend component increases the setup effort for the user. To make this process easier, we also provide configuration files that can be used to create a Conda[4] environment with all required dependencies installed and scripts to start the web server directly on a pre-configured port.

4.2.3 Obtaining Input for the Image Encoder Component

Given that SAM currently only supports 2D images, the AutoSeg tool should exclusively be active in the 2D view instead of the 3D environment. Moreover, our image encoder component requires a 2D array of 8-bit integer values to generate a segmentation using SAM, requiring a process to obtain this 2D input from the 3D medical scan.

Keeping the model input as close as possible to what the user sees in the editor fosters confidence in the system (also see section 2.2). This way, the user can better understand and experiment with why the model is making specific segmentation predictions. The following editor parameters affect what the user sees in the editor:

- **Current Slice** The user typically operates on 3D medical scans of which they only view a specific 2D slice at a given time.
- **Visible Layers** The user can concurrently view multiple channels of a medical scan (usually modalities as described in section 3.2.2), which Visian will overlay on top of each other as separate layers. However, the editor will only display the layers the user has not hidden.
- **View Type** The chosen view type (e.g., “Transverse”, “Sagittal”, or “Coronal”) impacts which image information is visible to the user. For example, the “Transverse” view will display the current slice along the z-axis, while the “Sagittal” view will display the slice along the x-axis.
- **View Settings** View parameters such as contrast or brightness influence the visible details and structures in the image.

Ideally, the input constructed for the image encoder would reflect all these parameters so that SAM receives nearly the same image as the user sees in the editor. However, accounting for all selected layers and view settings lies outside the timeframe of this thesis. Hence, we construct the image encoder input by extracting the raw image data of the current slice of the topmost visible image layer while considering the selected view type (see figure 4.3). This behavior is also detailed in the tool’s help text (see section 4.3.2), ensuring the user is aware of it.

As Visian stores image layer data as 32-bit floating point values, we must convert them into 8-bit integer values before making the `HTTP` request to the image encoder API endpoint.

[4] https://docs.conda.io

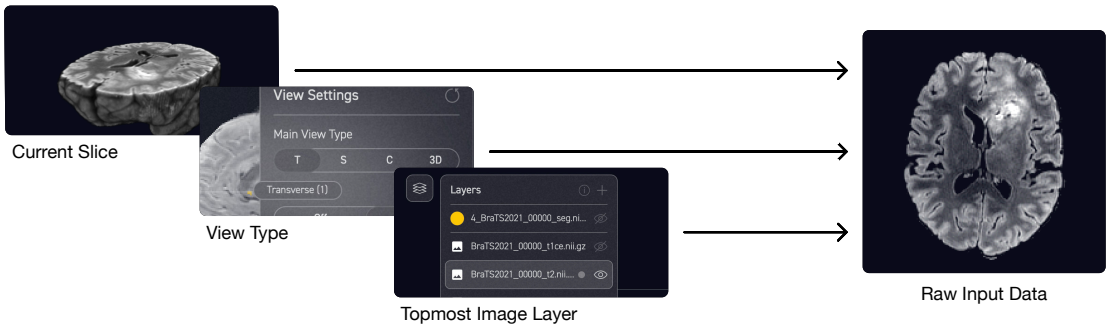


Figure 4.3: Obtaining image data based on current slice, view type, and topmost visible image layer.

4.2.4 Storing Embeddings

We have now established how to obtain an image embedding from the image encoder component. Hence, the next step is to determine how to store these embeddings. For that, we consider three possible options, which we will detail in the following.

Not Storing Embeddings In the first option, we do not store any embeddings at all and instead generate them before every segmentation creation. This approach removes the need for a storage system. However, it is slow, as an embedding must be fetched before each segmentation creation, adding an overhead of up to 32 seconds per interaction as described in section 5.3. This option is likely easy for users to understand since they only need to activate the tool, input their prompts, and receive the output after a specific wait time.

Storing Single Embeddings The second option entails the storage of the image embedding for the current editor parameters, allowing the reuse of the same embedding for multiple segmentation generations. However, the embedding must be regenerated if the user changes a parameter (e.g., the view type) and then reverts to the previous parameter value. This option introduces two concepts for the user: First, the generation of an image embedding, and second, the creation of a segmentation based on the embedding.

Storing Multiple Embeddings The third option proposes storing embeddings for arbitrary combinations of editor parameters, allowing multiple embeddings to be stored simultaneously. In contrast to the second option, when a user alters a parameter and resets it, the previous embedding will remain available. However, this approach presents a memory overhead, as simultaneously storing the embeddings for all slices, layers, and view types could require up to 7 GiB of RAM[5]. Nevertheless, a user is unlikely to exhaust all possible combinations in practice, as users typically

[5] A single embedding is of size $64 \times 64 \times 256$, resulting in 4 MiB per embedding (using 32-bit floating point numbers). Using the topmost image layer, current slice, and view type as parameters and assuming a scan with 155 slices and four layers across three view types, this results in 1860 potential embeddings, requiring a memory overhead of 7.27 GiB (see section 3.2.1 and section 4.2.3).

focus on a single view when systematically segmenting multiple slices of a scan. Furthermore, not all slices within a single scan will be relevant for segmentation; for instance, a brain tumor requiring segmentation might only appear in 62 out of 155 slices. Aside from the potential memory overhead, this option provides a superior user experience without re-fetching image embeddings.

Given the enhanced user experience in the context of requirement R5, we implement the last option. We also believe the memory overhead is acceptable for the reasons outlined above, and should memory usage in a production environment prove problematic, unused embeddings could automatically be removed (e.g., based on time since last use or distance to the current slice).

4.2.5 Triggering Embedding Generation

Having established how to store embeddings, we must determine when to generate these embeddings. Again, we consider different options.

Manual Generation The first option allows the user to initiate embedding generation by clicking an "Initialize" button. This triggers a request to the image encoder backend and notifies the user when it is ready. While this method only generates embeddings when needed, it introduces an extra step and potential waiting time before being able to create a segmentation, which may frustrate users unfamiliar with the process.

All-At-Once Generation The second option automatically generates all likely needed embeddings when loading an image into Visian, allowing to quickly switch editor parameters (i.e. layer, slice, or view type) without waiting time. However, this approach is time- and memory-consuming for unused embeddings (section 4.2.3). Handling the generation process in the background minimizes user interaction but might lack transparency. Alternatively, a loading bar could show the progress but would add more complexity to the user interface.

Debounced Generation The third option blends the benefits of manual and all-at-once methods. It automatically creates an embedding for the current editor settings but controls the rate by using a "debounce" function, which delays the process until the user's interaction is paused for a specified time. This function is only executed if the user does not start interacting again during that delay (otherwise, the delay timer would be reset and start over). This approach requires no extra user interaction and reduces perceived wait time. It also restricts the generation of embeddings only to those needed, i.e. when a user stays at a selected slice for a certain period, thereby minimizing unnecessary time and memory consumption. In this option, the user must wait for the embedding to be generated to get a segmentation proposal. However, the delay is less noticeable as the embedding generation begins in the background once Visian is opened or the editor parameters change. Though a user could intentionally trigger the creation of all embeddings by slowly scrolling through slices, this scenario is unlikely in everyday use.

For our integration, we choose option three using a delay of 1 second for the debounce function. This option offers the best trade-off between user experience and resource consumption, especially since it requires less user input, making it favorable to fulfill requirement R3. The delay of 1 second is long enough to avoid unnecessary embedding generation and short enough not to cause a noticeable wait time. This approach could be extended to generate embeddings automatically in the background for slices the user will likely look at next.

4.2.6 Generating Segmentations

After obtaining an image embedding, the next step is to generate a segmentation based on this embedding. Since SAM shows the best results for medical image segmentation using bounding box and point prompts (see section 3.2.2), we limit the AutoSeg tool to these two types of prompts. The user can specify up to one bounding box and an arbitrary number of points, either of type foreground or background. These prompt information are stored directly in the AutoSeg tool state and can be modified by the user at any time.

To generate a segmentation prediction using SAM based on these prompts and the current image embedding, we first need to encode the prompts and then decode the resulting mask from the image and prompt embedding (see figure 4.2). However, as we aim to run the prompt encoder and mask decoder in the browser, it is not feasible to use the existing SAM Python package for this inference step. This is because browsers run web applications in a limited sandbox environment, preventing the execution of Python code and access to essential libraries such as PyTorch.

ONNX Framework We can use the Open Neural Network Exchange (ONNX) framework[6] to convert SAM to the ONNX format, which represents inference operations such as matrix multiplications as a serializable graph. This graph can then be optimized and executed on multiple platforms using a specialized ONNX runtime (short "ORT"), like, for example, the ORT Web runtime that is optimized to run in a web browser. ORT Web dynamically detects the user's hardware capabilities and selects an appropriate "execution provider" to process the model graph. For instance, if the user has a GPU, ORT Web utilizes WebGL as the execution provider for GPU inference. If the user lacks a GPU, ORT Web switches to WebAssembly (WASM) as the execution provider for CPU inference. Moreover, when initialized with the WASM backend, ORT Web examines whether the browser supports features such as multi-threading or SIMD[7] and takes advantage of them if available. Additionally, ORT Web supports quantized models with 8-bit integer values instead of 32-bit floating point values to represent weights and activations, reducing the model size and enabling faster inference. In section 5.3.2, we also compare inference performance based on the available features of ONNX.

Using ONNX with SAM To make use of the ORT web runtime for generating segmentations in the frontend component, we utilize a script provided by the authors of SAM to create a quantized ONNX model that contains both the prompt encoder and mask decoder. During this step, we also

[6] https://onnx.ai

[7] Single Instruction, Multiple Data (SIMD) refers to a parallel computing technique that allows to simultaneously perform the same operation on multiple data elements [22].

configure the model to return only one mask instead of multiple ranked masks for a given input. This avoids unnecessary computation since we only need one segmentation prediction according to requirement R2. This optimized model has a size of 8.74 MB, making it feasible to serve it as a static asset of the web application.

Once we have obtained an embedding from the image encoder component, we use the JavaScript library `onnxruntime-web`[8] to create an "inference session" based on the quantized model, which we can reuse throughout the tool's lifespan. Each time the user-provided prompts change (for example, by adding a bounding box or altering point prompts), we run the ONNX model by providing the current image embedding, the user prompts, and a scaling factor. We obtain this scaling factor by determining how much the image needs to be resized so that its longest side is 1024 pixels long, matching the required input size of the image encoder component (see section 3.2.1). The factor is required by the prompt encoder and mask decoder model to automatically scale the prompt inputs to the correct size and return a mask matching the original image size. It could not be determined otherwise, as the provided image embedding no longer contains information about the original dimensions.

After running the model and receiving a prediction, we can directly use the resulting segmentation since we configured the model only to return one mask. The only post-processing required is to convert the resulting array of floating-point values to a binary mask by thresholding it at 0.

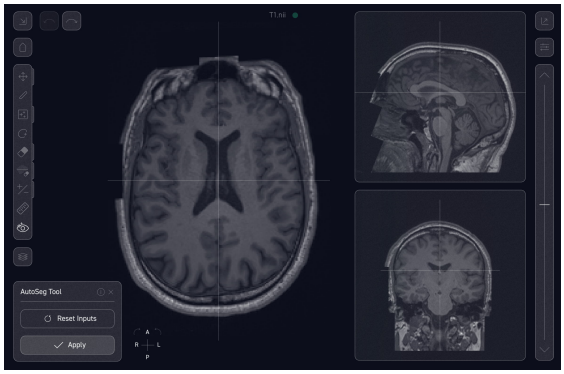
Debouncing Segmentations While the authors of SAM claim an inference speed as low as 50 milliseconds (see section 3.2), when we integrate the model as described above, generating a prediction in Microsoft Edge takes on average 56 milliseconds on a MacBook Pro with M1 Pro Chip (see evaluation in section 5.3). While this allows for immediate segmentation after the user has specified some prompts, it is not fast enough to generate a segmentation prediction for every prompt change. For example, attempting to generate a segmentation each time the bounding box changes would result in system lag, as the box coordinates change at an interval shorter than 56 milliseconds when the user starts dragging, leading to a delayed GUI update and consequently a suboptimal user experience. To address this, we implement a debounce mechanism similar to the one described in section 4.2.5, but with a shorter delay of just 30 milliseconds. This delay shows to be short enough to quickly provide the user with feedback after they pause their interaction with the tool while also preventing the system from lagging when they are still interacting. This way, we ensure to be in alignment with requirements R3 and R4.

4.3 Graphical User Interface

The interface of the AutoSeg tool should be intuitive to use and easy to learn, as stated in requirement R4. To achieve that, we follow the design principles of Visian and build upon existing concepts.

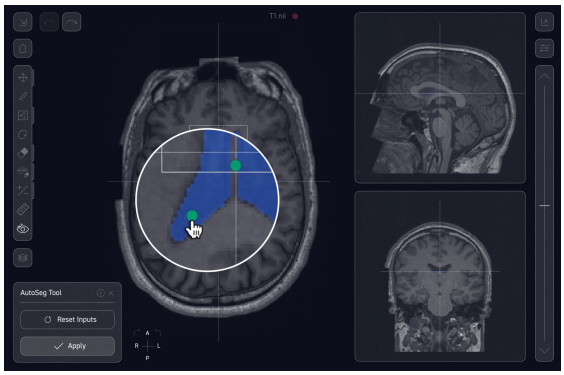
When the user activates the AutoSeg tool within Visian in the left toolbar, a popup appears in the bottom left of the screen, similar to that of the 3D Smart Brush tool (see figure 4.4a). The tool is now in an intermediate state, enabling the user to draw a bounding box or add foreground and

[8] https://www.npmjs.com/package/onnxruntime-web



(a) Activating the AutoSeg tool opens a popup on the bottom left.

(b) Specifying a bounding box as prompt and receiving the first segmentation prediction.



(c) Adding two foreground points and receiving an updated prediction.

(d) Accepting the prediction to apply it to the current layer.

Figure 4.4: Using the AutoSeg tool (areas of interest enlarged).

background points that guide the tool to generate a segmentation prediction (see figure 4.4b and figure 4.4c). As soon as the user provides such prompts, the prediction is generated and displayed as a blue overlay preview, similar to the preview of the 3D Smart Brush. Subsequently, the user can click a button to apply the preview, effectively persisting the prediction to the currently selected segmentation layer (see figure 4.4d).

In the following sections, we detail available options for specifying prompts through the GUI and how the complete workflow using the AutoSeg tool looks.

4.3.1 Specifying Prompts

As previously discussed in section 4.2.6, the AutoSeg tool allows for bounding box and point prompts to guide the segmentation generation. Consequently, various methods should be available for the user to specify these inputs. Following, we highlight multiple options and the rationale behind our final decision.

Prompting Sub-Tools In option one, we would introduce multiple "sub-tools" within the interface of the AutoSeg tool. For example, a bounding box tool would allow users to create and manipulate a bounding box through simple click-and-drag actions, enabling them to move the entire box or adjust its corners. Additionally, a foreground point tool would enable users to add or remove points with left and right clicks, following the approach used by the measurement tool for nodes (see section 3.1.2). A similar tool would be used for background points. These tools would be shown as buttons in the bottom left popup, with a highlight on the active tool, similar to Meta's web demo[9]. This design would clarify for the user which prompt mode is currently active and what a click will likely do. However, this option requires more user interaction since each sub-tool must be activated first. It also adds complexity to the GUI, requiring users to understand the sub-tools concept for effective use.

Prompt Mode Switch Option two involves implementing a "switch button" that changes the result of a user click, allowing users to easily toggle between "bounding box" mode and "points" mode. In bounding box mode, the mechanics would remain as in option one. In points mode, users could add foreground points with a left click, background points with a right click, and remove any existing points with another right click. This approach consolidates the separate point sub-tools from option one into one mode, reducing some of the GUI complexity. For instance, removing a point becomes independent of whether it is a foreground or background point. The option also allows associating one mode with a specific modifier key, like the "Ctrl" key, enabling automatic switching between modes when pressed. This feature caters to experienced users in the context of requirement R5, allowing quicker tool operation. However, this option's downside lies in its need for additional explanations, particularly for adding background points, as this may not be immediately clear from the GUI.

Mouse-Only Interaction A third option is to allow all actions to be performed with the mouse alone. Users could then create a bounding box by clicking and dragging. If a user drags again, the original box will be removed and replaced with a new one. This guarantees that only one box is present at any given time, as the model does not permit more than one bounding box as a prompt. Furthermore, users can independently add foreground points with a left click and background points with a right click. Clicking right on an existing point will still remove it. This option offers the cleanest user interface since no additional buttons are required. It also allows for fast interaction because no tool switching is necessary, and actions are not bound to additional modifier keys. However, it may require more explanation for the user to understand how to operate the tool.

After testing the different approaches in a prototypical implementation, we conclude that the last option best aligns with requirement R5. Especially when users segment many scans, providing an efficient way to guide segmentations is essential. Accordingly, we aim to optimize speed and simplicity rather than ease of use for first-time users. We therefore give more weight to R5 compared to R4 in this case. To accommodate first-time users, we implement a help guide explaining how to interact with the tool when activating it for the first time (see section 4.3.2).

[9] https://segment-anything.com/demo

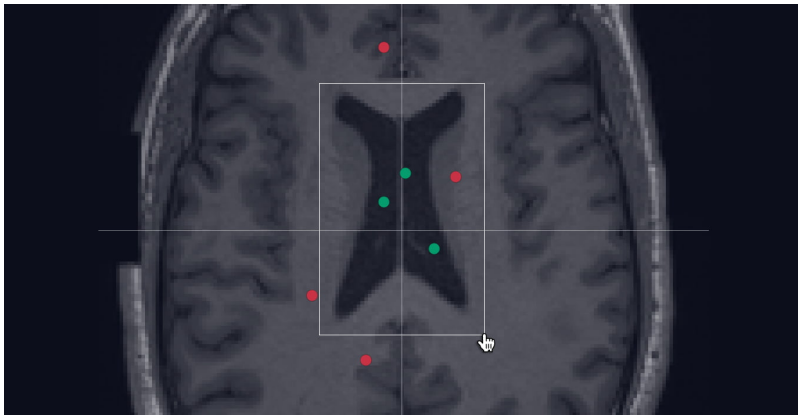


Figure 4.5: A bounding box, three foreground points, and four background points as prompt inputs for the tool.

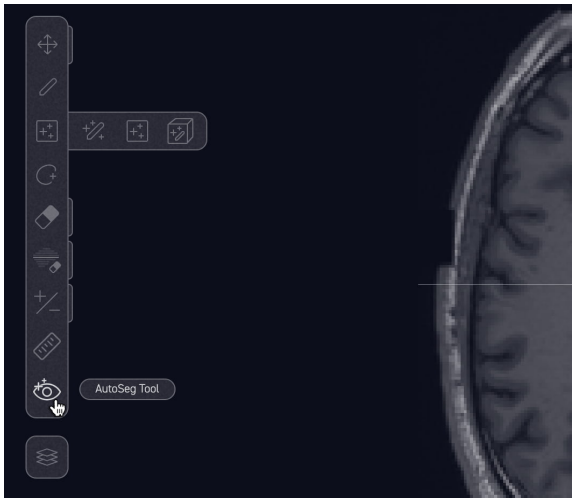


Figure 4.6: Hovering cursor over the AutoSeg tool icon, smart brush tools on the top.

To display the user-provided prompts, we follow the design of the measurement tool (see section 3.1.2) to achieve a consistent look and feel. Hence, we represent the bounding box with a simple white rectangle, the foreground points with green circles, and the background points with red circles (see figure 4.5). When hovering the mouse over a point, the cursor changes to indicate that the user can remove the point by clicking.

4.3.2 Tool Workflow

Following, we describe the entire workflow a user experiences when using the AutoSeg tool, including different details of the GUI.

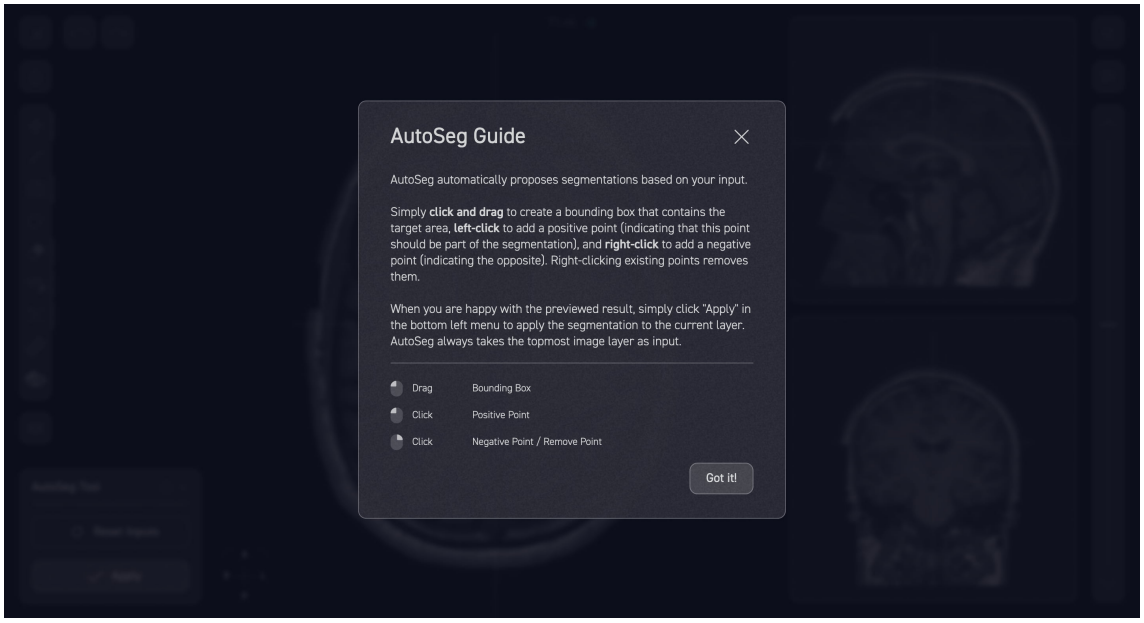


Figure 4.7: Help text of the tool.

We design the AutoSeg tool so users can easily activate it through a button in the Visian toolbar, similar to other tools. Symbolized by an eye icon embellished with stars (see figure 4.6), it visually aligns with "smart" tools such as the 3D Smart Brush, therefore complying with requirement R4. Once clicked, the tool becomes active, and a popup appears at the screen's bottom left corner (see figure 4.4a).

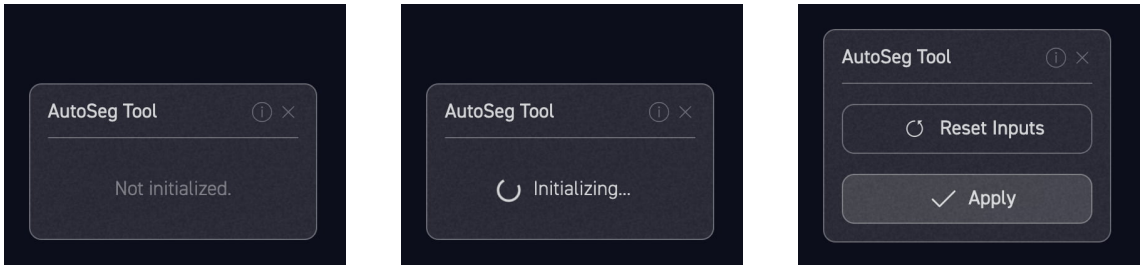
Popup and Help Text The AutoSeg popup consists of two primary sections: the header and the body. The header displays the tool's name, a help icon, and a cross icon. Clicking the cross icon deactivates the tool, while clicking the help icon shows a text popup in the center, explaining the tool's functionality and how the user can interact with it (see figure 4.7).

This text popup also appears automatically when the user activates the tool for the first time. After the user acknowledges the tips provided, we store a flag in the browser's `localstorage` database so that the popup is not shown again when activating the tool.

Tool States The body of the popup can exhibit three different states: "uninitialized", "loading", and "ready" as shown in figure 4.8.

The "uninitialized" state is displayed when no embedding is available yet for the current editor parameters, namely the visible slice, view type, and topmost image layer (also see section 4.2.3). Since the system automatically generates an embedding for the present editor parameters after a short delay (see section 4.2.5), this state is usually only visible to the user when they quickly change editor parameters, hence not triggering the embedding generation.

The "loading" state is visible while the embedding is generated, i.e., a request has been sent to the image encoder component, and the system is awaiting its response. Upon successfully retrieving the embedding, the tool transitions into the "ready" state.



(a) State “unitialized” **(b)** State “loading” **(c)** State “ready”

Figure 4.8: How the AutoSeg tool represents its current state.

Prompt inputs that the user created as described in section 4.3.1 are then used to automatically generate a segmentation preview, which is displayed as a blue overlay on top of the image (see figure 4.4b), similar to the one displayed when using the 3D Smart Brush tool (see section 3.1.2). This preview does not affect the selected layer and disappears when the tool is no longer active, fulfilling requirement R2. In the “ready” state, the tool popup on the bottom left shows a “Reset” and an “Apply” button. Clicking the “Reset” button (or pressing the “Esc” key) clears the current user prompt inputs, thereby removing a potentially visible bounding box or rendered point prompt circles. By clicking the “Apply” button (or pressing the “Enter” key), the current segmentation preview is applied to the selected layer, and the tool is reset in the same way as when clicking the “Reset” button.

Segmentation Process After applying the segmentation preview, the user can repeat the process by providing new prompt inputs and receiving a new prediction. This way, the user can also iteratively create a complex segmentation that the tool cannot generate in one step. The user can also switch to other tools, such as the simple Eraser or Brush tools, to correct minor segmentation errors manually. The AutoSeg tool will maintain already fetched embeddings as described in section 4.2.4.

Notably, the user is also allowed to create prompt inputs in all three states, i.e., when no embedding is available yet. This enables users to start guiding the tool while waiting for the embedding, complying with requirement R5. The corresponding segmentation preview will automatically be generated once the embedding is available.

As discussed in section 4.2.6, we debounce the segmentation generation to avoid unnecessary computation. This means the segmentation preview is not updated immediately when the user changes the prompt inputs but only after the user stops interacting with the tool for 30 milliseconds. Hence, the user can interact with the tool smoothly, and the segmentation preview is only updated when the user naturally pauses, aligning the process with requirement R3.

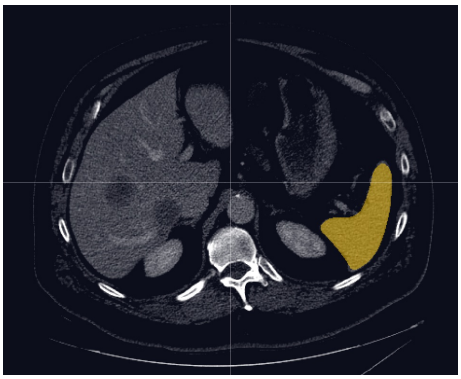
5 Results

We will now assess the new AutoSeg tool within Visian. For that, we perform a qualitative evaluation comparing the new tool with existing tools in Visian and methods in 3D Slicer (see section 2.4). Next, we describe and evaluate a user study we performed with volunteers, detailing the qualitative and quantitative feedback we gathered. Lastly, we measure and analyze the runtime performance of the components of the AutoSeg tool in different environments.

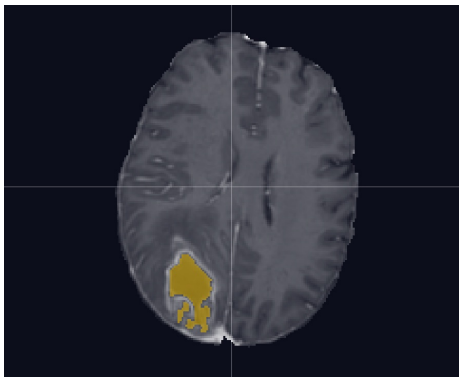
5.1 Qualitative Evaluation

In this section, we conduct an in-depth comparison to evaluate the capabilities of the new AutoSeg tool against existing segmentation methods in Visian and 3D Slicer. We create segmentations with the Level Tracing tool in 3D Slicer, the Bounded Smart Brush tool in Visian, and the new AutoSeg tool in Visian. The Level Tracing tool and Bounded Smart Brush tool have been chosen for comparison since they have shown to be the most efficient semi-automatic segmentation tools in their respective applications. We conduct this evaluation using two distinct segmentation tasks of varying difficulty levels to ensure a comprehensive understanding of the tool's capabilities.

The first task involves recreating a professional segmentation of the spleen using the three methods described above. For that, we utilize the `spleen_6.nii.gz` 3D CT scan of the abdomen, obtained from the dataset `Task06_Spleen` of the Medical Image Decathlon [1] and provided by the Memorial Sloan Kettering Cancer Center. Slice 86 of the scan can be seen in figure 5.1a, including a ground



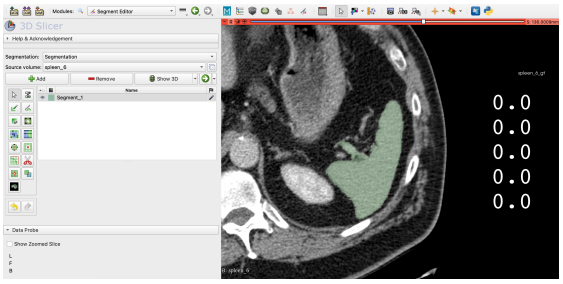
(a) CT abdomen scan including a segmentation of the spleen.



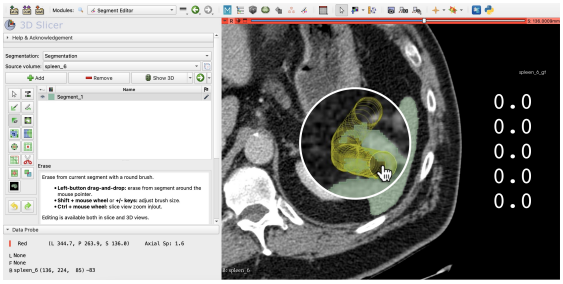
(b) MRI T1 brain scan including a segmentation of the tumor core.

Figure 5.1: Slices of medical images used for evaluation.

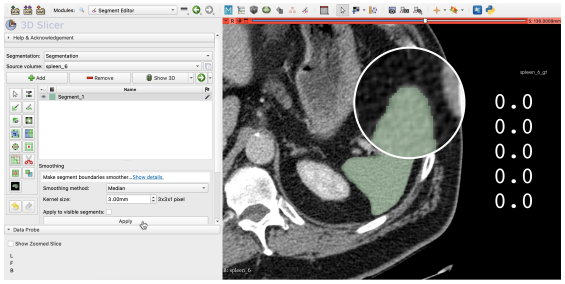
(a) Finding the contour with the Level Tracing tool.



(b) Accepting the contour leads to a segmentation.



(c) Removing unwanted parts with the Eraser tool.



(d) Using the Smooth tool to soften the segmentation.

Figure 5.2: Segmenting the spleen in a CT scan in 3D Slicer (areas of interest enlarged).

truth segmentation of the spleen created by a medical professional. We rate this task as "easy" given that the spleen is prominently visible and features a distinct brightness value compared to the surrounding tissue.

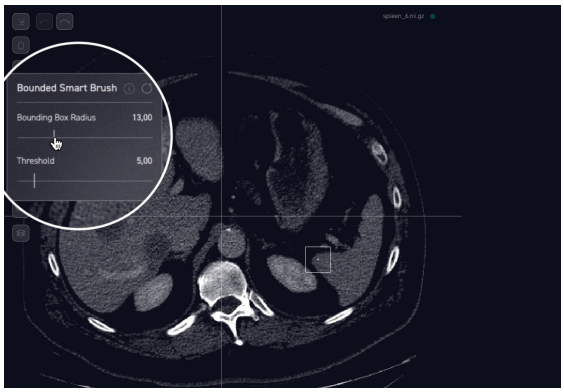
Our second task focuses on brain tumor segmentation. Here, we use the 3D brain MRI scan `brain_00495_t1ce.nii.gz` obtained from the BraTS dataset [3, 4, 49]. Notably, we employ the contrast-enhanced T1-weighted modality of the scan that highlights the presence of fatty tissue. We show slice 94 of the scan in figure 5.1b, including a ground truth segmentation of the tumor core created by a medical professional. We categorize this segmentation task as "difficult" due to the complex outline of the tumor core, its ill-defined contour in certain regions, and its lack of uniform brightness values.

In the following sections, we will compare how well the AutoSeg tool performs in these tasks against the existing methods and highlight general observations we made during the evaluation.

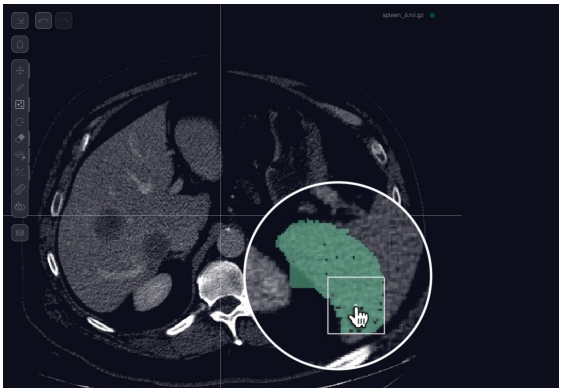
5.1.1 Easy Segmentation Task

First, we applied the Level Tracing tool, the Bounded Smart Brush tool, and the new AutoSeg tool to the easy spleen segmentation task.

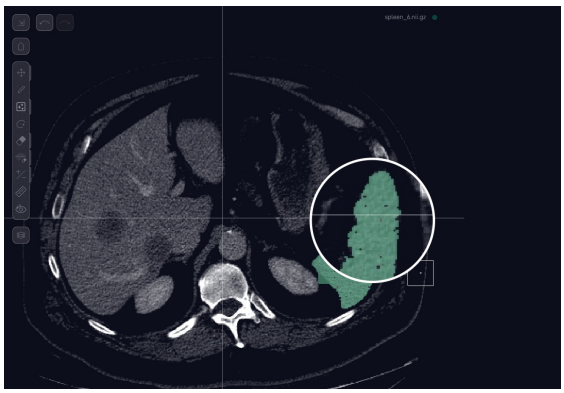
Level Tracing Tool In 3D Slicer, we started by loading the abdomen scan and creating a new "segmentation" (the corresponding concept to layers in Visian). We then selected the "Level Tracing" tool, which highlights a closed contour starting and ending at the current mouse position while maintaining a constant brightness value (see section 2.4.1 for a detailed explanation). By finding and selecting a contour that outlines the spleen (see figure 5.2a), we could apply the outlined



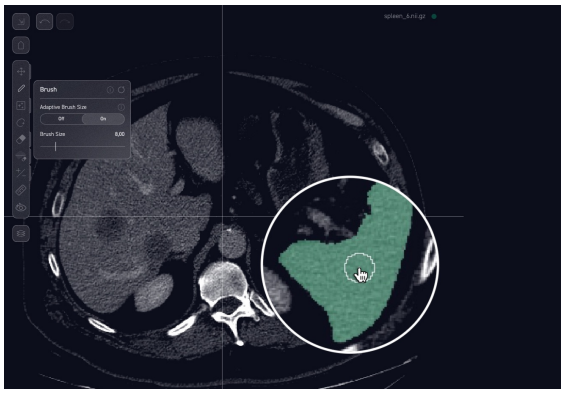
(a) Configuring the Bounded Smart Brush tool.



(b) Starting a segmentation.



(c) Using the Bounded Smart Brush leaves artifacts.

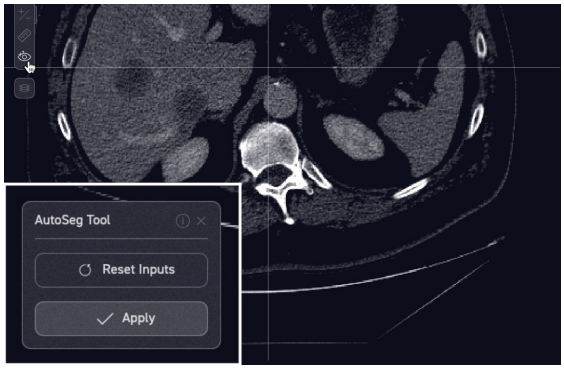


(d) Using Eraser and Brush tool to correct errors.

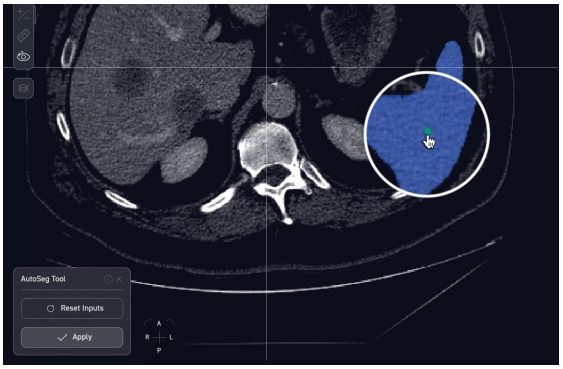
Figure 5.3: Segmenting the spleen using the Bounded Smart Brush tool in Visian (areas of interest enlarged).

area to the active segmentation with a single click (see figure 5.2b). However, the tool's brightness threshold resulted in selecting neighboring tissue, necessitating corrective measures. We removed unwanted parts using the Eraser tool with two click-and-drag operations (see figure 5.2c). Lastly, as the Level Tracing tool follows a pixel-wise path, the segmented area did not have a smooth outline. We corrected this by applying the "Smooth" tool and thereby achieved a satisfactory result (see figure 5.2d). We had to click eight times and needed 29 seconds to complete the task. Compared with the ground truth, the resulting segmentation achieved a Dice score of 96.31%.

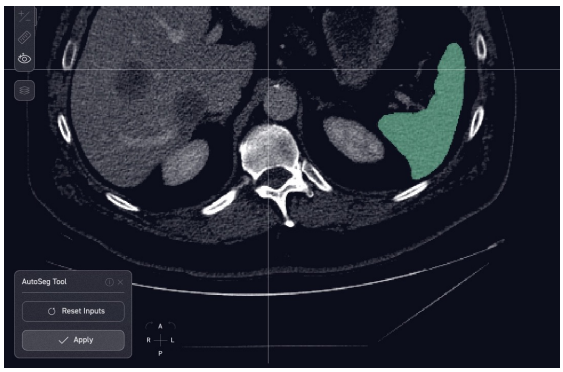
Bounded Smart Brush Tool In Visian, we also started by loading the scan and creating a new segmentation layer. We then activated the Bounded Smart Brush tool and slightly adjusted the brush radius (see figure 5.3a) before segmenting the target area by clicking and dragging. This process marked all pixels within the brush radius and a brightness difference below the default threshold value (see figure 5.3b). While filling the target area, the tool occasionally segmented outside parts when encountering pixels with similar brightness values to the background. In addition, it frequently left out pixels within the target area as they were not below the threshold (see figure 5.3c). These pixels could have been "captured" using a higher threshold; however, this would have prevented us from correctly segmenting the edges of the target area. After filling the target area as best as



(a) Activating the AutoSeg tool.



(b) Adding a prompt point and resulting prediction.

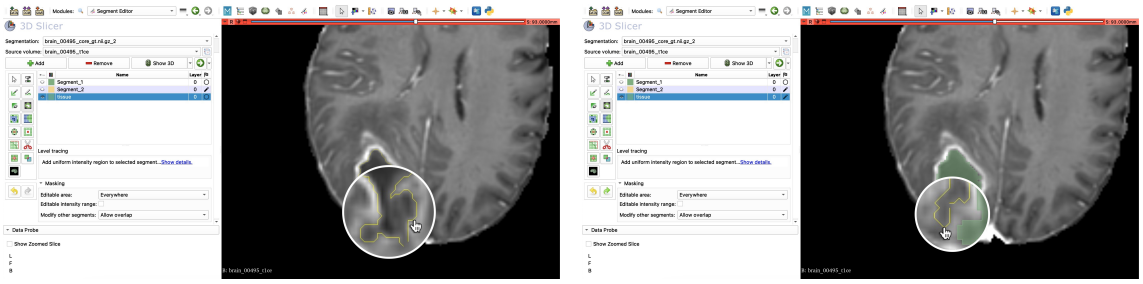


(c) Having applied the predicted segmentation.

Figure 5.4: Segmenting the spleen using the AutoSeg tool in Visian (areas of interest enlarged).

possible using the Bounded Smart Brush, we used the Eraser tool to remove unwanted parts and the Brush tool to fill in missing pixels and smooth the contour (see figure 5.3d). The total number of required clicks was nine, and the process took 59 seconds to complete. The Dice score, when compared to the ground truth, was 96.51%.

AutoSeg Tool Lastly, we started the segmentation process in Visian using the new AutoSeg tool, again loading the scan and creating a new segmentation layer. After activating the tool, the bottom left menu directly displayed the “Apply” and “Reset Inputs” buttons, indicating the tool had already loaded the embedding for the image (see figure 5.4a). We then created a single foreground prompt point in the center of the target area by clicking. Upon placing the point, the tool displayed a blue segmentation prediction overlayed on top of the image (see figure 5.4b). As the prediction was satisfactory, we accepted the proposal by clicking “Apply”, which made the blue overlay disappear and instead showed the segmentation applied to the current layer (see figure 5.4c). This entire process required a total of three clicks and took 7 seconds. The resulting segmentation yielded a Dice score of 97.15% compared to the ground truth.



(a) Finding the contour of the main target area.

(b) Finding the outline of the separate tumor part while having applied the main area.

Figure 5.5: Segmenting a brain tumor in an MRI scan in 3D Slicer (areas of interest enlarged).

We observe that using the AutoSeg tool, we can achieve a segmentation with higher accuracy in less time than using the Bounded Smart Brush tool or the Level Tracing tool in 3D Slicer. Besides the improved speed and accuracy, achieving the shown segmentation using AutoSeg requires fewer and different user interactions compared to the other methods. Instead of having to click and drag to create or correct the segmentation, single clicks are sufficient to guide the prediction.

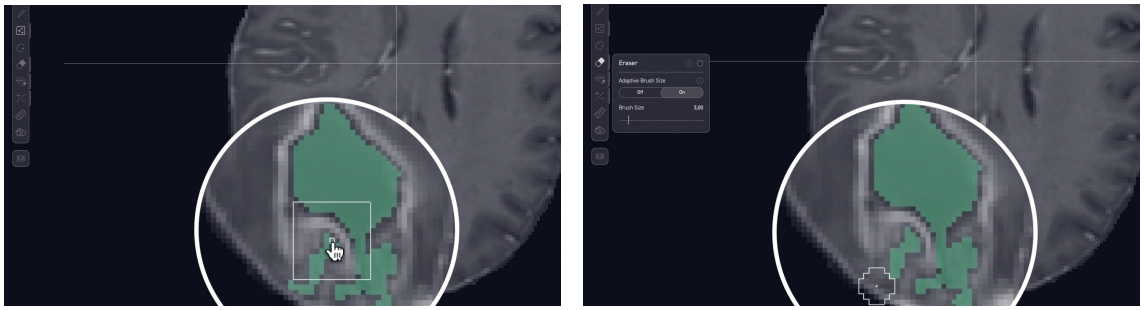
The GUI of our new tool was more straightforward compared to the GUI of 3D Slicer, which presents users with many text and interaction options such as buttons and dropdown menus (see figure 5.2). While there is only a moderate gap in GUI complexity to the Bounded Smart Brush due to the overall Visian design, the AutoSeg tool still provides a layout with fewer configuration options and more mouse-driven interactions.

5.1.2 Difficult Segmentation Task

We also carried out the segmentation process using the same tools but now addressed the more challenging task of segmenting the previously described brain tumor scan.

Level Tracing Tool We proceeded with the Level Tracing tool in 3D Slicer in the same way as for the easy segmentation task. This time, however, it proved more challenging to find a suitable outline since the target area consisted of separate parts that needed to be selected individually (see figure 5.5). Unlike in the easy task, no smoothing or additional retouching was required, as the segmentation quality appeared to be sufficient. The entire process involved three clicks: one to select the tool, one to apply the first outline, and another to apply the second outline. This segmentation took 34 seconds to complete and achieved a Dice score of 90.86% compared with the ground truth.

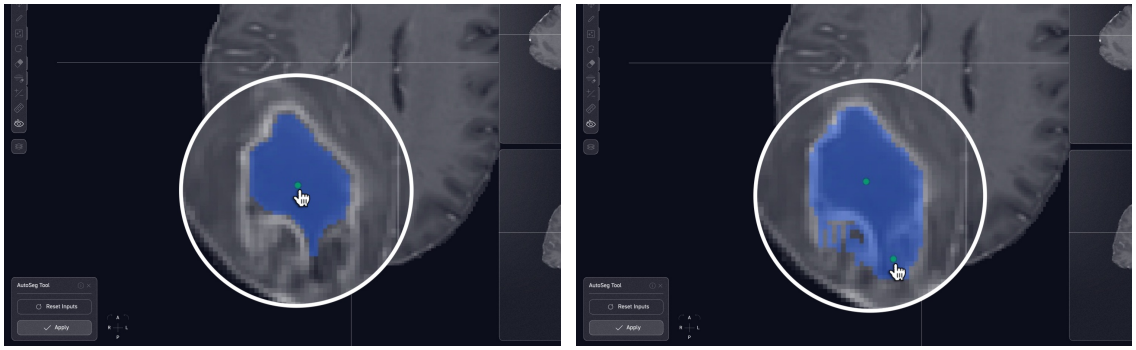
Bounded Smart Brush Tool With Visian's Bounded Smart Brush tool, we also proceeded similarly to the easy task, but this time, we did not adjust the brush configuration as the defaults were suitable. Despite the complexity of the segmentation, the process was relatively easy, and only one correction using the Eraser tool was necessary (see figure 5.6). The total process involved six



(a) Using the Bounded Smart Brush tool.

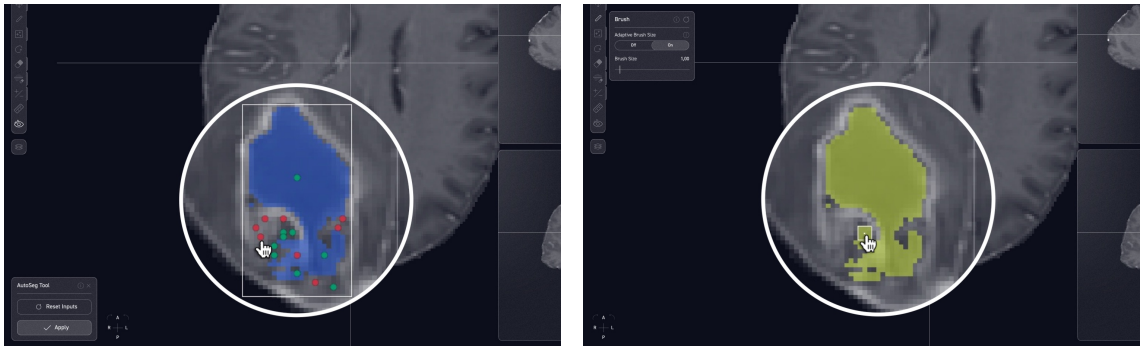
(b) Using Eraser and Brush tool to correct errors.

Figure 5.6: Segmenting a brain tumor using the Bounded Smart Brush tool in Visian (areas of interest enlarged).



(a) Adding a single foreground point leads to the correct prediction of the inner tumor area.

(b) Adding a second point to include the remaining area leads to overfitting on other parts.



(c) Adding more point and bounding box prompts does not improve the prediction.

(d) Correct mistakes in the applied segmentation using traditional Eraser and Brush tools.

Figure 5.7: Segmenting a brain tumor using the AutoSeg tool in Visian (areas of interest enlarged).

clicks: one to select the tool, three click-and-drags to fill the target area, one to select the Eraser tool, and one click-and-drag to erase unwanted parts. The process took 29 seconds, resulting in a Dice score of 92.02% compared to the ground truth.

AutoSeg Tool When employing the AutoSeg tool for the difficult task, the initial placement of the first prompt point led the tool to accurately predict the main tumor area (see figure 5.7a). However, adding a second point to include the remaining area resulted in a larger prediction than desired (see figure 5.7b). We created additional prompt points and a bounding box but failed to improve the segmentation proposal. Instead, providing more prompt inputs appeared to decrease the quality of the prediction, as the tool often suggested a larger area than intended or left out parts that we previously marked with foreground point prompts (see figure 5.7c). We eventually resorted to the Eraser and Brush tools to manually correct the prediction (see figure 5.7d). The process involved 28 clicks, including various interactions such as point placements, bounding box drawing, and manual corrections. It took 50 seconds to complete and yielded a Dice score of 94.60% compared to the ground truth.

In the difficult task, the AutoSeg tool still provides a more straightforward interface than 3D Slicer and the Bounded Smart Brush tool in Visian. However, we could not produce a satisfying segmentation without manual correction of errors in the prediction. Placing multiple prompt points in the image and correcting the segmentation with Brush and Eraser tools took longer than using the Bounded Smart Brush or the Levels Tool in 3D Slicer.

5.2 Usability Evaluation

To understand how well the new AutoSeg tool meets the requirements specified in section 4.1, we have conducted a user study to answer the following two questions.

Question 1 "How easy is it to understand and learn the new AutoSeg tool?" This question targets requirement R4 to assess the tool's usability for new users, especially since we could not evaluate this aspect when using the tool ourselves in section 5.1.

Question 2 "How does the AutoSeg tool compare to a traditional segmentation tool concerning speed, accuracy, and user experience?" This question is targeted at the requirements R3 and R5. We also want to answer this question for a scan that is easy to segment as well as one that is hard to segment, as described in section 5.1.

5.2.1 Participants and Setup

We recruited 12 participants (ages 21 to 59, 4 female, 8 male) to conduct the study. While all participants were familiar with medical image segmentation, only three had previous hands-on experience in the domain. These three individuals occasionally segmented images as part of their professional duties or personal interests.

The study was conducted in both remote and in-person environments, allowing participants to use their own system or a system provided by us. We first gave participants a brief introduction to medical image segmentation and the Visian editor, including a demonstration of core GUI elements, navigation tools, and the usage of the Bounded Smart Brush tool. We also shared with them the

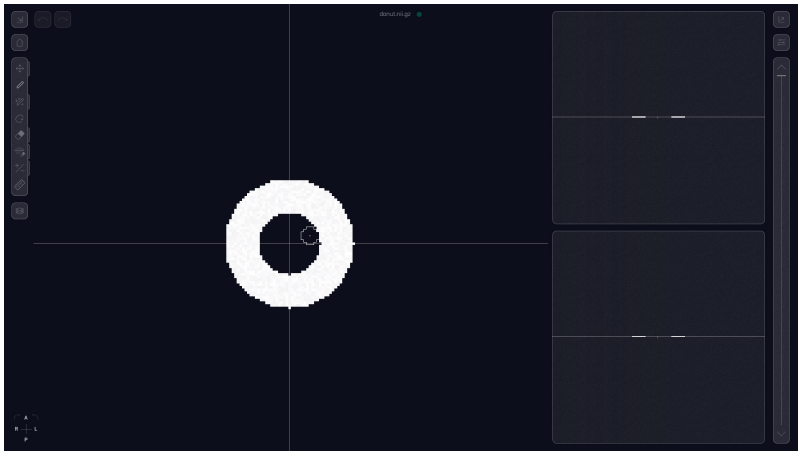


Figure 5.8: Custom created “donut” scan for task 1 of the user study.

background of developing the AutoSeg tool within Visian, driven by recent advances in natural image segmentation. Furthermore, participants were told they could decide how precise they wanted their segmentations to be as long as they maintained their accuracy consistently across the study.

The participants then performed three tasks within the Visian editor, outlined in subsequent sections. We measured the time the participants took to complete each task and collected the created segmentation layers when applicable. This allowed us to evaluate the accuracy of the created segmentations compared to the ground truth segmentation.

In addition to the tasks, we requested participants to fill out a two-part questionnaire that combined elements from the Questionnaire for User Interface Satisfaction (QUIS) [12] and the Computer System Usability Questionnaire (CSUQ) [41]. Since the questionnaire contained identical questions for tasks 2 and 3, we instructed participants to complete the first questionnaire part directly after finishing the second task. This part collected demographic information and included the question set for task 2. After performing the third task, we asked participants to complete the remaining portion containing the question set for task 3.

5.2.2 Tasks

Every task the participants performed was structured similarly, commencing with loading a scan into Visian. Next, we instructed each participant to create two new segmentation layers. In each task, the participant used the previously introduced Bounded Smart Brush tool and the new AutoSeg tool.

Task 1: Learning to Operate the Tool

The first task started with the user loading a custom-created single slice “donut” image, as depicted in figure 5.8. We then instructed participants to perform subtask 1.1 on the first layer as follows: “Use the Bounded Smart Brush tool to segment only the white donut. When you are satisfied with the result, declare that you are done.” Subsequently, the user hid the current layer, switched to the second layer, and was directed to perform task 1.2: “Find the new AutoSeg tool in the interface.

Then, use it to segment the white donut. When you are satisfied with the result, declare that you are done." This task was intended primarily for gathering data on question Q1. Moreover, we designed it to require the participant to add at least two prompt inputs to remove the donut hole from the prediction. For example, one bounding box around the donut leads to the tool predicting the whole donut area as segmentation. Only after placing a background point in the middle of the donut will the tool correctly predict the outer ring. However, the same could be achieved by placing a foreground point on the outer ring and then a background point in the middle of the donut. After finishing the task, we allowed the participant to explore the AutoSeg tool further and ask questions.

Task 2: Easy Segmentation

The second task required the participant to load the same abdomen CT scan and ground truth segmentation used for the qualitative evaluation (see figure 5.1a). We then directed the user to change to slice 86 and adjust the contrast window to 0 – 0.3, so that the spleen was visible and the low contrast of the CT scan was mitigated. Furthermore, we informed the participants that the subsequent tasks would focus on recreating the ground truth segmentation of the spleen and that they could hide or show the ground truth segmentation layer at any time. They were then instructed to perform task 2.1 on the first layer: "Recreate the ground truth segmentation using the Bounded Smart Brush tool. When you are satisfied with the result, declare that you are done." Afterward, the participant hid the current layer, switched to the second layer, and was instructed to perform task 2.2: "Recreate the ground truth segmentation using the new AutoSeg tool. When you are satisfied with the result, declare that you are done."

Task 3: Difficult Segmentation

The third task involved the participant loading the same brain MRI scan and ground truth segmentation we use for the qualitative evaluation (see figure 5.1b). We guided the user to change to slice 94 to make the tumor core visible; no contrast adjustment was necessary for this scan. Afterward, the participant was instructed to perform task 3.1 on the first layer: "Recreate the ground truth segmentation using the Bounded Smart Brush tool. When you are satisfied with the result, declare that you are done." Afterward, the participant hid the current layer, switched to the second layer, and was directed to perform task 3.2: "Recreate the ground truth segmentation using the new AutoSeg tool. When you are satisfied with the result, declare that you are done."

5.2.3 Results and Analysis

Following, we will present the results of the user study. We start by detailing findings concerning the learnability of the tool based on task 1, then analyze the results of tasks 2 and 3, and conclude with general findings concerning the participants' reaction to the AutoSeg tool.

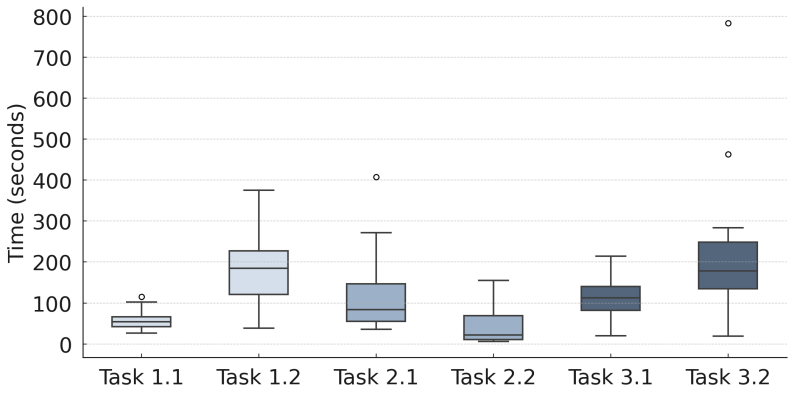


Figure 5.9: Measured completion time of all tasks.

Task 1: Learning to Operate the Tool

In task 1.1, the participants were generally able to successfully use the Bounded Smart Brush tool to segment the donut shape. However, there was a noticeable variation in their understanding of how the tool could be adjusted to their specific needs. For instance, the optimal approach to complete this task was to increase the brush radius so it covers the entire donut shape and set the threshold to a value encompassing all bright pixels of the scan. Nevertheless, some participants instead employed a small brush radius and low threshold, requiring more interactions than necessary for the task. The average time to complete task 1.1 was 59 seconds, as shown in figure 5.9.

In task 1.2, all but two participants could independently locate and activate the AutoSeg tool. Initial reactions to the help text were mixed; two participants disliked the volume of text displayed, questioning whether they were required to read it. Once users had acknowledged the help text, we observed a generally good understanding that the tool works by providing prompt inputs and receiving a segmentation in return.

Some challenges emerged during the operation of the AutoSeg tool. For example, participants occasionally struggled with how to proceed after the initial prediction, often resulting in the prediction of the entire donut shape, including the center part. While five participants wished to revisit the help text, none of them could locate the help icon or associate it with the initially shown help text. Nevertheless, after continued experimentation, all participants generated segmentation predictions leading to the intended segmentation of the donut ring. We also observed that participants commonly created a bounding box with a significant margin around the donut shape, making the AutoSeg tool predict the outer black part of the image instead of the donut shape itself. Two participants expressed dissatisfaction with the tool's inability to predict some pixels of the shape, leading to additional effort in correction.

The concept of first receiving a prediction and then applying it to the selected layer was not immediately clear to all participants, as some neglected to apply the generated prediction before declaring that they were done. However, other participants even applied the prediction multiple times, segmenting the donut shape step by step. These participants tended to only use the bounding box tool and expressed their thoughts that only this way could they prevent the tool from predicting

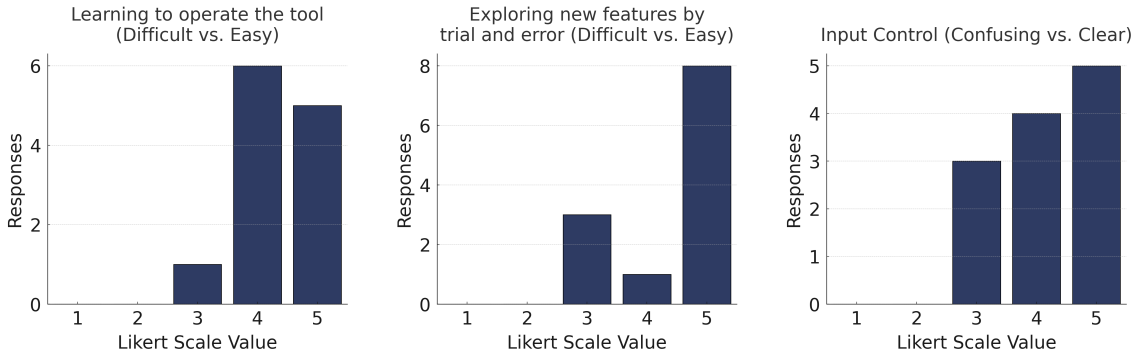


Figure 5.10: Questionaire results concerning learnability of the AutoSeg tool.

the center part of the donut. This revealed potential ambiguities in the help text and demonstrated that the ability to combine different prompt methods or add negative prompt points was not readily understood.

The average time taken to complete task 1.2 was 3 minutes and 6 seconds, as depicted in figure 5.9.

Afterward, five participants expressed the wish for a better explanation of the tool. This was also reflected in suggestions for improvement obtained through the questionnaire, such as incorporating explanatory gif animations instead of the help text. The questionnaire responses shown in figure 5.10 also reveal that 41.7% of participants rated the ease of learning to operate the AutoSeg tool with 5/5 on the Likert scale [42], 50.0% rated it with 4/5, and 8.3% with 3/5, indicating in context of question Q1 that the tool was overall easy to learn. Interestingly, 66.7% rated how easy it was to explore features by trial and error with 5/5, 8.3% rated it with 4/5, and 25.0% with 3/5. This discrepancy between rating 5/5 and 3/5 indicates that while some participants found it intuitive to get to know the tool by trial and error, others might have preferred a more guided introduction. The average rating of 4.2/5 for clarity in input control aligns with the qualitative observations during the study, indicating that the tool was more intuitive for some participants than others.

Task 2 and 3: Easy and Difficult Segmentation Task

In both task 2 and task 3, participants were tasked with recreating the presented ground truth segmentation using the Bounded Smart Brush and AutoSeg tool. All participants could complete this process for the easy and difficult segmentation task.

Easy Segmentation Participants expressed their satisfaction when using the AutoSeg tool for the easy segmentation task 2.2, highlighting the ease of use compared to the previously used Bounded Smart Brush tool for the same segmentation. While the majority of participants placed either a single foreground point prompt in the spleen's center or drew a bounding box around it, some opted for a step-by-step approach, using multiple bounding boxes and applying after each prediction. Only two participants tried to tweak the initially generated prediction further by placing additional foreground and background point prompts at the edge of the prediction. However, this attempt did not enhance the segmentation. Instead, the tool modified the prediction in a different area, similar

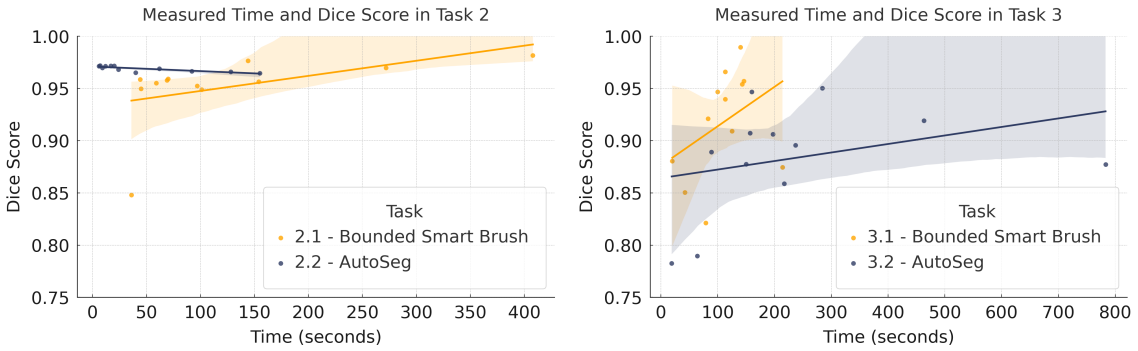


Figure 5.11: Comparing measured time and accuracy for the easy and difficult segmentation task.

to what we observe in section 5.1.2, and led the participants to place even more point prompts there. The two participants complained about not understanding how to place the points to get their desired prediction.

Comparing the times needed to complete the spleen segmentation using both tools reveals that the AutoSeg tool is significantly faster than the Bounded Smart Brush tool ($t_{11} = 3.16$, $p = 0.0091$). On average, participants accomplished the spleen segmentation in just 48 seconds using the AutoSeg tool, while the Bounded Smart Brush tool required 2 minutes and 5 seconds (see figure 5.9). Furthermore, the segmentations achieved with the AutoSeg tool were more accurate (with an average Dice score of 96.9%) compared to those produced with the Bounded Smart Brush tool (with an average Dice score 95.13%). However, we did not find this difference to be significant ($t_{11} = -1.73$, $p = 0.111$).

Difficult Segmentation In task 3, participants were less enthusiastic about the AutoSeg tool than the Bounded Smart Brush tool. Four participants explicitly found task 3.2 to be "more difficult" than task 2.2 and expressed frustration with the tool's failure to predict the desired segmentation in response to their inputs. Another four participants attempted to generate the complete segmentation without applying partial predictions, leading to unsatisfactory results and more time spent correcting segmentation. The remaining participants segmented the tumor core incrementally using the AutoSeg tool, with those aiming for higher accuracy often drawing small bounding boxes to include small areas neglected by the tool's previous predictions.

In this task, we found the AutoSeg tool to be significantly slower than the Bounded Smart Brush tool ($t_{11} = -2.59$, $p = 0.0252$). Participants completed the brain tumor segmentation on average in 1 minute and 50 seconds using the Bounded Smart Brush tool, compared to 3 minutes and 55 seconds with the AutoSeg tool (see figure 5.9). We also found the resulting segmentations to be significantly less accurate when using the AutoSeg tool compared to the Bounded Smart Brush ($t_{11} = 3.5724$, $p = 0.0044$). While participants achieved an average Dice score of 91.76% using the Bounded Smart Brush tool, they only achieved a score of 88.34% using the AutoSeg tool.

Correlation of Completion Time and Accuracy To add to the results for question Q2, we can analyze the relationship between the time needed to complete a segmentation and the accuracy achieved. As shown in figure 5.11, these relationships vary for task 2 and 3.

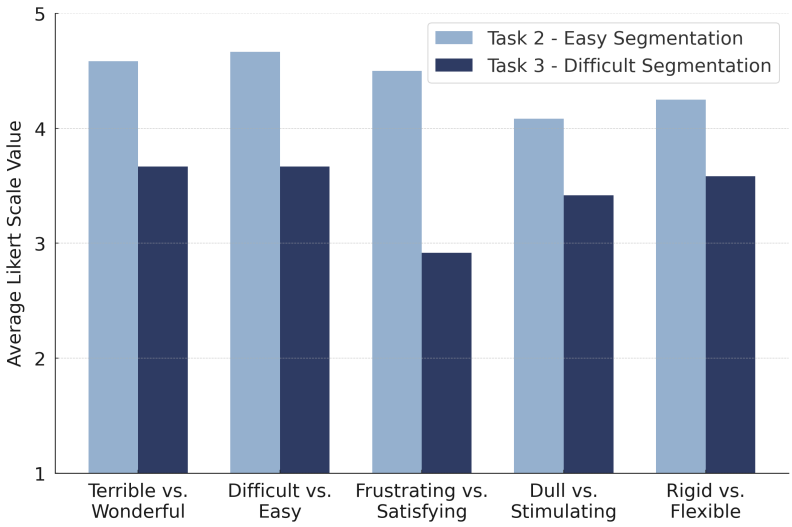


Figure 5.12: Average reaction ratings of participants for the AutoSeg tool in task 2 and task 3.

For task 2.1 and 3.1 we find a moderate positive correlation, though not significant, with $r_{10} = 0.47$, $p = 0.1224$ for task 2.1 and $r_{10} = 0.38$, $p = 0.2260$ for task 3.1. This indicates that spending more time using the Bounded Smart Brush tool tends to improve the segmentation.

Interestingly, we observe a strong negative correlation in task 2.2 that is also significant ($r_{10} = -0.81$, $p = 0.0014$), implying that spending more time with the AutoSeg tool leads to worse segmentations in the second task. This result further supports the qualitative observations in section 5.1.2 that more prompt inputs lead to worse segmentation predictions.

The results of task 3.2 show a moderate positive correlation that is not significant, with $r_{10} = 0.32$, $p = 0.3100$. In this task, participants tended to create multiple segmentations with partial predictions, indicating that spending more time and using smaller target areas improves segmentations when using the AutoSeg tool.

General Reaction to the AutoSeg Tool

Besides measuring time and accuracy for the two tasks, we also asked participants to rate their reaction to the AutoSeg tool on different perception aspects for both task 2 and task 3. The responses, visualized in figure 5.12, reveal that participants generally rated their reaction to the AutoSeg tool higher for task 2 than for task 3 on a five-point Likert scale [42]. Specifically, we observe the highest discrepancy in the reaction score for the prompt asking how satisfying the tool was to use. While participants gave an average score of 4.5 for using the AutoSeg tool in task 2, they only gave an average score of 2.9 for task 3. This difference further supports the qualitative observations of participants' dissatisfaction with the tool's prediction in task 3. It also aligns with the slower tool speed and lower accuracy in task 3 compared to task 2.

Furthermore, we asked participants to rate general aspects of the AutoSeg tool. Remarkably, 91.7% of participants rated the speed of the tool with 5/5 as "fast enough", and only 8.3% rated it with 4/5. Regarding correcting mistakes, the responses were more varied, with 25.0% giving a score of 2/5, 25.0% a score of 3/5, 8.3% a score of 4/5, and 41.7% a score of 5/5. This distribution supports the

observations that users often encountered issues with the tool modifying the prediction differently than they expected when adding a new prompt input. However, 66.7% of participants strongly agree that they became productive quickly using the AutoSeg tool, with 25.0% somewhat agreeing and only 8.3% taking a neutral position. Additionally, all participants either somewhat agree or strongly agree that the information provided for the AutoSeg tool is easy to understand. This is an intriguing observation as we noticed some participants having difficulties understanding how the tool works in task 1.2, indicating a potential gap between perceived simplicity and actual tool behavior. Lastly, all participants either strongly or somewhat agree that they enjoy using the interface of the AutoSeg tool, showing a generally positive reception of the tool's design.

5.3 Run-Time Performance Evaluation

Since the AutoSeg tool aims to offer immediate feedback to the user according to requirement R3, all system components must be optimized for fast run-time performance. In the following sections, we measure and analyze the speed of the AutoSeg components to understand how well they fulfill requirement R3.

5.3.1 Image Encoder Component

The modular design of the AutoSeg tool enables the image encoder component to operate independently from the frontend component. This independence provides flexibility regarding the computational environment for running the encoder. Given this flexibility, we evaluate the runtime performance of the image encoder across different hardware devices. Specifically, we test it both on consumer devices, namely on two MacBook Pros with Intel and M1 Max chip, as well as on a scientific compute cluster with access to different types of GPUs.

For our evaluation, we generate ten embeddings from randomly produced image data with the typical dimensions processed by the image encoder ($244 \times 170 \times 3$) and measure the time required to generate each of them. We report the average measured time as shown in table 5.1:

Processing Device	**Avg. Time (seconds)**
Intel Quad-Core i5 2 GHz	32.40
Apple M1 Max	3.04
Nvidia GeForce RTX 2080 Ti 11 GB	0.82
Nvidia A40 48 GB	0.61
Nvidia A100 40 GB	0.51

Table 5.1: Average time of generating ten embeddings using different hardware devices.

The results indicate that leveraging a GPU offers a considerable advantage in computational speed. Running the image encoder on systems equipped with GPUs consistently yields faster performance than a CPU-only configuration.

It should be noted that this benchmark purely evaluates the time it takes to generate the embedding within the image encoder component. The time the user has to wait for the AutoSeg tool while it is in "loading" state (and therefore waiting for a response from the image encoder component) might be longer due to individual network latency, which we have not evaluated in this case.

5.3.2 Frontend Component

The frontend component of the AutoSeg tool takes an essential role in shaping the user experience, specifically regarding the speed of segmentation prediction. Unlike when creating an embedding, we do not implement a "loading" state that indicates to the user that the system is working. Hence, we must ensure that users do not have to wait for a segmentation preview or experience any lag when adding a new prompt input.

Users generally perceive a system as instantaneous when it responds within 100 milliseconds. A response time between 100 milliseconds and 1 second may be noticeable but does not typically interrupt the user's flow of thought. However, response times above 1 second can become strongly noticeable, risking a shift in the user's attention [18]. These thresholds provide a benchmark against which to measure the AutoSeg tool's prediction speed.

We will first evaluate the general speed of the tool and then conduct a second evaluation with certain browser features disabled.

General Response Time

To evaluate the general response time of the tool, we conduct an assessment across different operating systems and browsers. Specifically, we choose four common browsers at the beginning of 2023[1], namely Google Chrome, Safari, Microsoft Edge, and Firefox. In each environment, we create 150 predictions with the same medical scan and editor setup. We measure the time it takes to generate a prediction when placing one foreground prompt point in the center of the image slice. We show the results of the benchmark in table 5.2:

	Avg. Prediction Time (milliseconds)			
	Chrome	**Safari**	**Edge**	**Firefox**
MacOS 13.4, Intel Core i5 2.0 Ghz	107.47	210.79	95.05	219.10
MacOS 13.4, Apple M1 Pro	70.52	132.67	56.45	141.37
Ubuntu 22.04, AMD Ryzen 7 Pro	115.37	—	113.16	188.34
Windows 10, Intel Core i5 2.8 Ghz	92.85	—	91.28	191.75
Browser Average	96.55	171.73	88.98	185.14

Table 5.2: Average measured prediction time in different environments and browsers.

[1] https://gs.statcounter.com/browser-market-share#monthly-202301-202301-bar

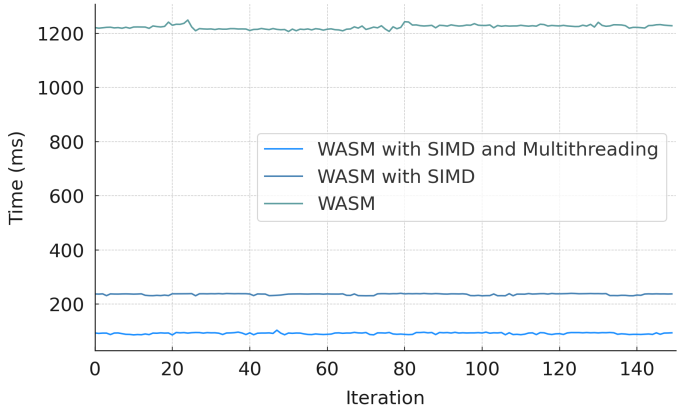


Figure 5.13: Measured time of the AutoSeg tool to generate multiple iterations of segmentation predictions utilizing different features of the ORT Web runtime.

We find that the prediction speed of the AutoSeg tool is acceptable across all tested environments and browsers since the response time never exceeds the threshold of 1 second, where the user's attention might shift. Remarkably, in five of the 14 tested combinations, the measured prediction time is even below the threshold of 100 milliseconds, implying that users will likely perceive the tool as reacting instantaneously.

Additionally, we observe that prediction times are consistently lowest on Microsoft Edge, with an average of 88.98 milliseconds. Comparing the tested systems, running the AutoSeg tool in browsers on MacOS 13.4 using an Apple M1 Pro chip yields the fastest prediction times, averaging 91.28 milliseconds.

Disabled Browser Features

To perform inference in the Browser using SAM, we utilize the ORT Web runtime. This aspect is significant because not all browser versions support features that the ORT Web runtime can utilize to optimize model inference. Because of this, we conduct measurements of the prediction speed with different features enabled using a 2022 13-inch MacBook Pro with an Apple M2 chip.

We evaluate the performance of the AutoSeg tool with (1) no specific features available that the ORT runtime can use, (2) with only SIMD (see section 4.2.6) enabled, and (3) with SIMD and multithreading enabled. The results of these measurements over 150 iterations of predicting a segmentation are depicted in figure 5.13.

Our observations reveal that the worst performance occurs when the ORT Web runtime cannot utilize the SIMD feature of WASM. In this scenario, we measure an average prediction time of 1223.21 milliseconds, leading to a noticeable lag before the tool can display a prediction after the user has stopped interacting. Because we debounce the segmentation generation (see section 4.2.6), the user will have to wait for over a second after providing a prompt input before receiving a prediction.

Fortunately, most browsers support the SIMD feature since 2022[2]. When SIMD is enabled, we see a substantial decrease in prediction time, averaging 236.20 milliseconds.

We can further improve the performance by enabling access to the WASM Threads feature, which allows multithreading for model inference. On average, this optimization reduces the prediction time to 91.94 milliseconds. Since this value is below the 100 milliseconds threshold, we can assume that users will perceive the AutoSeg tool as reacting instantaneously when both SIMD and multithreading features are available.

[2] https://webassembly.org/roadmap

6 Discussion

We created the AutoSeg tool to provide quick segmentation predictions for medical images in the Visian editor. Built on Meta's SAM, the tool leverages the model's abilities to generate segmentations based on user-provided prompts.

Our qualitative evaluation showed that AutoSeg meets the defined functional requirements. Users can guide segmentations with bounding box and point prompts, fulfilling requirement R1. When prompts change, the tool shows a blue segmentation preview, providing immediate feedback and meeting requirement R2. The displayed preview refreshes automatically when new prompt inputs are provided and does not affect the currently selected segmentation layer unless applied.

Next, we discuss achievements and limitations concerning non-functional requirements in more detail.

6.1 Run-Time Performance

Requirement R3 states that users should receive immediate feedback upon providing input. This goal was based on the fast speed of SAM claimed in the original paper and the provided web demo.

During the run-time evaluation described in section 5.3.2, we found that AutoSeg requires, on average, 136 milliseconds to produce a segmentation for most browsers running on commodity hardware. Although this duration prevents per-frame predictions while drawing a bounding box, we did not observe any system lag during such activities. This responsiveness can be attributed to debouncing the segmentation generation, as described in section 4.2.6, indicating that this was a good optimization choice.

The user study confirmed AutoSeg's satisfactory speed, with predictions often generated in under 100 milliseconds on various browsers and different hardware (also see run-time evaluation in section 5.3.2). This makes it appear almost instantaneous to users, and any delays are minimal and non-distracting [18]. The tool is noticeably faster than semi-automatic tools like MITK or ITK-Snap (discussed in section 2.4) for single-slice 2D segmentations. The potential to extend AutoSeg to 3D segmentations is discussed further in chapter 7.

During our evaluation of the AutoSeg tool, we also investigated how fast the embedding generation is. We did not experience any wait time concerning embedding generation, mainly since we worked on a single slice for which the tool could already fetch the embedding in the background. Because we did not evaluate the tool for segmenting multiple slices of a 3D image, we cannot make a statement about the user experience in this scenario. However, our run-time evaluation showed that embedding generation takes less than a second when using a state-of-the-art GPU and is unlikely

to pose an issue when segmenting multiple slices. The benchmark also showed that running the image encoder locally on a system with suitable hardware is a good option to reduce added network latency.

Limitations Embedding generation may take considerable time when using a CPU, and users can, therefore, benefit from running the image encoder in a dedicated ML computing environment. However, this indicates that our architecture decision of splitting the system into a frontend and backend component is suitable for different use cases. The model checkpoint size of the image encoder currently prohibits embedding generation directly in the browser but could be considered in the future if the model size decreases.

Overall we conclude that the AutoSeg tool can generate segmentation predictions instantaneously and is therefore suitable for interactive use, fulfilling requirement R3.

6.2 User Experience and Learnability

According to requirement R4, the AutoSeg tool should be easy to understand and learn, especially for first-time users. We assume that choosing certain GUI options, such as using a similar design as existing Visian tools or providing a help text, would improve learnability.

In the user study, most participants could learn to use the AutoSeg tool without previous knowledge, showing that our implementation effectively communicates the tool's primary usage. During our qualitative evaluation, we also found that the interface of the AutoSeg tool is cleaner and less complex than that of the Level Tracing tool of 3D Slicer. AutoSeg stands in contrast to methods available in ITK-Snap, where we needed to read online documentation before being able to operate the "Snake" tool described in section 2.4.

Limitations The user study showed that the initial explanation of the tool could be improved. Half of the participants rated the "learnability" of the tool only with 4/5, confirming our observation that users were not always sure how to operate the tool after reading the initially shown help text. For example, many participants did not apply the predicted segmentation or did not realize they could add negative prompt points. Based on that, our initial assumption that a help text is sufficient to learn the tool can be called into question. Instead, a step-by-step tutorial that guides the user through the AutoSeg process by employing gif animations in place of text explanations could be more effective. The user study also showed that participants could not locate the help icon, indicating that it is not prominent enough.

Furthermore, the varying responses to the question of how easy it was to explore the tool by trial and error revealed that some participants were more comfortable playing with the tool to understand how it works than others. For example, one participant compared AutoSeg to the image background removal tool of Microsoft PowerPoint that he was familiar with. Followingly, AutoSeg might be easier to learn for users with previous experience in image editing software and more challenging to grasp for users without this experience.

Overall, we conclude that the AutoSeg tool is easier to learn and understand than existing semi-automatic segmentation methods in tools like MITK or ITK-Snap. However, the initial explanation of the tool can be improved through the discussed measures.

6.3 Operation and Task Efficiency

Lastly, we consider requirement R5, which states that the tool should require few interactions so users can operate it efficiently. We assumed that we could use SAM to obtain helpful segmentations, using its learned knowledge of natural images and applying it to the medical image domain. We reviewed existing research on SAM's usefulness for medical images in section 3.2.2, indicating that the performance of the AutoSeg tool might depend on the complexity of the segmentation task.

We found that one can quickly achieve a high-accuracy segmentation with little user input for easy segmentation tasks. Participants in the user study achieved more accurate segmentations than when using the Bounded Smart Brush tool with just a single prompt point, with the fastest participant only taking 7 seconds to complete task 2.2. Many participants actively expressed their excitement about the AutoSeg tool when they could create a better segmentation than with the traditional tool in a fraction of the time.

We also noticed the more intuitive prompting approach of AutoSeg compared to the sensitive contour selection of the Level Tracing tool. Using prompts allowed us to focus on the object of interest instead of having to consider the pixel intensity values on which the proposed contour was based.

Limitations When applying AutoSeg to a more difficult segmentation task, we found that the tool was slower and led to less accurate segmentations than the Level Tracing tool or the Bounded Smart Brush. This was also reflected in participants' questionnaire responses, where they rated using the AutoSeg tool in the difficult task as more frustrating than in the easy task.

Generally, we identified three significant characteristics of the AutoSeg tool during the evaluation: its dependence on task complexity, fewer prompts leading to improved accuracy, and the influence of bounding box prompts.

Firstly, we noticed that the tool's performance strongly depends on the segmentation task's complexity. It excels at providing segmentations for clearly defined shapes but struggles with intricate or soft contours. This confirms research on SAM for medical images, which also found that the model's performance depends on the complexity of the task [15]. Intricate shapes can still be segmented but require more user interaction and time: For example, some participants achieved similar segmentation accuracy compared with the Bounded Smart Brush by applying predictions multiple times and repeating the process for smaller regions. However, this required much more time, as shown in the user study results in figure 5.11 of section 5.2.3.

Secondly, initial prompt inputs generally guide the tool to correctly identify some parts of the targeted area, but adding more point prompts or a bounding box does not necessarily improve the prediction. Interestingly, when some prompt inputs are already present, adding another prompt point often alters predictions in unrelated areas, potentially leading to non-contiguous segmentations. This also showed in the user study, when participants aimed to achieve a very accurate segmentation

without working iteratively. They often added a prompt point in one place and then received a prediction that was changed in another area. After they placed another prompt point there, the prediction would change again in a previously marked region, and so on. This aligns with our finding that a significant negative correlation exists between accuracy and completion time for the easy segmentation task. It also confirms existing research showing that SAM is most effective with fewer prompts [61].

Thirdly, the bounding box prompt input has the strongest effect on the prediction. For example, when placing multiple background points and adding a bounding box containing them, the tool often predicts a segmentation that ignores the previously added background points. We also observed in the user study that creating a bounding box with a large margin around the target region affected the prediction negatively. This confirms previously discussed research showing that SAM is most effective when the bounding box encloses the object of interest as tightly as possible [15].

Overall, we conclude that the AutoSeg tool outperforms the evaluated semi-automatic segmentation methods in speed and accuracy for easy tasks. However, the tool cannot compete with traditional algorithmic tools for more difficult segmentations. Hence, our initial assumption that we can use SAM to obtain helpful medical image segmentations is only partially confirmed.

7 Future Work

While the AutoSeg tool has shown to be a valuable addition to the Visian editor, we can identify several areas that allow for further improvements. We will discuss three options for future work, specifically a potential adoption of SAM to the medical domain, the extension of the AutoSeg tool to support 3D segmentations, and the introduction of a copilot-like system for manual segmentation.

Adoption of SAM to the Medical Domain Currently, the AutoSeg tool relies on SAM to generate segmentations for medical images; however, SAM was originally only trained on natural images. This issue shows in our evaluation of the tool, where it is apparent that the lack of training on medical images limits SAM to rely on general knowledge about shapes, contrast, and edges for making predictions. Without access to semantic medical knowledge, such as understanding the specific tissue types of a brain tumor, SAM faces difficulties with complex segmentation tasks requiring such knowledge. Some research has already been done on adapting SAM to the medical domain, for example, by fine-tuning the mask decoder [33] or introducing additional trainable layers in the image encoder [34]. There are also first attempts to remove the need for prompting the model with a bounding box or foreground points, instead relying on text-based prompts that would remove the need for users to specify localized prompts for each image [54]. We suggest further research on how SAM can be adapted or fine-tuned for medical images.

Extension of AutoSeg to Support 3D Segmentations Currently, SAM can only generate segmentations for 2D images. This constraint has limited our initial implementation of the AutoSeg tool to individual, two-dimensional slices of 3D medical images. To improve upon this, we identify different options that could be explored to leverage SAM for 3D segmentations. The first approach entails the user providing an initial bounding box for the target area in the first slice. The tool would then automatically progress through subsequent slices, each time adjusting the bounding box based on the previous segmentation. Since areas of interest appear similar across neighboring slices, this iterative process should effectively segment the target region in the 3D volume. An alternative option is to obtain a 2D segmentation for a given slice and then change to a perpendicular view aligned on the same center point. Here, the previous two-dimensional segmentation would appear only as a line but allow the placement of foreground prompt points along it. SAM could then generate a prediction based on these prompt inputs, resulting in a second segmentation positioned perpendicular to the first. This process could be iteratively repeated for the third view and other slices until all voxels of the target region have been segmented. Lastly, and more fundamentally, the architecture of SAM could be altered to process three-dimensional input, such as information from adjacent slices in medical scans. This adjustment might involve modifying the image encoder to accept 3D inputs, adapting the prompt encoder to process 3D prompt information, and changing the mask decoder to output 3D segmentations. Preliminary efforts in this direction have been documented [24], suggesting the feasibility of such an extension.

Copilot-Like System for Manual Segmentation Currently, AutoSeg is implemented as a separate tool within Visian, requiring users to switch back to traditional tools should they wish to correct segmentations manually. This workflow could be enhanced by drawing inspiration from the recent release of "GitHub Copilot"[1], which employs LLMs to suggest code snippets to developers, akin to a copilot providing helpful directions to a driver. We envision a "Segmentation Copilot" in Visian that might function similarly, automatically displaying predictions based on already segmented areas. Such an approach could lead to a workflow where users start a segmentation using the Brush tool, with the copilot subsequently proposing a segmentation based on these initial strokes. Users could either accept this suggestion or continue manually refining the initial segmentation, thus providing further guidance to the tool. With the current version of SAM, this could be achieved by automatically placing foreground points on the initial brush strokes and adding background points on areas where the Eraser tool was applied. This approach might lead to an even more intuitive user experience than what AutoSeg currently offers, as the copilot would solely augment the segmentation process instead of acting as a separate tool.

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

8 Conclusion

The aim of this thesis is to explore the possibility and methods of implementing an interactive segmentation tool for medical images in the Visian editor. This exploration is driven by the invaluable insights that medical images offer, supporting clinicians in tasks like patient diagnosis, treatment tracking, or surgery planning. However, manual segmentation is a labor-intensive task requiring a domain specialist's expertise. As a result, a significant amount of research is being dedicated to automating this process through semi- or fully-automatic methods.

Our work in the bachelor project led us to recognize the Visian editor as a modern and versatile tool for medical image viewing and editing. Although well-equipped for manual segmentation, Visian lacked tools that employed machine learning to assist users in the segmentation process. This finding coincided with the release of the "Segment Anything Model" by Meta, which demonstrated impressive capabilities in natural image segmentation, outperforming all previous state-of-the-art methods. The release sparked our interest in how SAM's capabilities could be leveraged in an interactive tool for medical image segmentation and led to the creation of the "AutoSeg" tool.

We started the thesis by investigating different techniques for medical image segmentation, highlighting semi-automatic approaches as the most promising due to their ability to generalize to different tasks and consider user input to improve their results. Furthermore, we compared existing tools like 3D-Slicer or ITK-Snap but found them lacking in speed and user-friendliness. We then analyzed the architecture of SAM and assessed preliminary research on its suitability for medical images, discovering both strengths and weaknesses of the model. We also examined Visian's interface and tools to understand how a novel segmentation method could be integrated into the existing system.

Based on the aggregated knowledge of Visian, SAM, and existing medical imaging tools, we then defined requirements that aimed to fully utilize SAMs capabilities and create a better segmentation experience than already existing methods can offer. Consequently, these requirements informed our implementation decisions and design approach for the new tool.

We integrated AutoSeg as a new segmentation tool into the Visian web app, allowing users to provide bounding boxes or point prompts to generate a segmentation prediction for slices of medical images. After receiving a prediction, users can interactively add more prompt inputs to refine the segmentation or accept and apply it to the current layer. For this to work, the tool automatically fetches the required image embedding from a separately deployed service.

After finishing the implementation, we evaluated the tool's overall user experience by comparing it with existing methods and conducting a user study. The focus of the study was to answer how easy it is to learn the new tool and how well it performs in terms of speed and accuracy. Informed by our research into SAM's applicability for medical images, we deliberately structured the evaluations to include both an easy and a difficult segmentation task based on image attributes that influenced SAM's performance.

Our findings indicated that AutoSeg was generally easy to learn thanks to its simple interface, especially compared to existing methods in other tools. However, we also identified potential usability improvements that could further enhance the experience for new users. Furthermore, our results confirmed previous research findings that SAM's performance strongly depends on the segmentation task's complexity. AutoSeg performed quite well on easy tasks but showed worse accuracy and speed on more complicated tasks.

Lastly, we discussed these observations in detail and identified the limitations of the tool, proposing future work that could be done to address them.

By integrating the new AutoSeg tool into Visian, we showed that it is possible to leverage SAM's capabilities for medical image segmentation. We not only showed this in an isolated test environment but released the AutoSeg tool as part of Visian, making it available to all users. This way, it can be used for clinical purposes and to support researchers in creating labeled datasets to train future models, thereby accelerating research in the domain.

We conclude that medical image segmentation is an inherently challenging task that offers many opportunities to significantly improve the diagnosis and treatment of patients in clinical care. The AutoSeg tool represents another step towards a streamlined segmentation experience in this field. We hope it will be of use to clinicians and researchers in accelerating their work and helping patients in a better way.

Bibliography

[1] M. Antonelli, A. Reinke, S. Bakas, K. Farahani, A. Kopp-Schneider, B. A. Landman, G. Litjens, B. Menze, O. Ronneberger, R. M. Summers, B. van Ginneken, M. Bilello, P. Bilic, P. F. Christ, R. K. G. Do, M. J. Gollub, S. H. Heckers, H. Huisman, W. R. Jarnagin, M. K. McHugo, S. Napel, J. S. G. Pernicka, K. Rhode, C. Tobon-Gomez, E. Vorontsov, J. A. Meakin, S. Ourselin, M. Wiesenfarth, P. Arbeláez, B. Bae, S. Chen, L. Daza, J. Feng, B. He, F. Isensee, Y. Ji, F. Jia, I. Kim, K. Maier-Hein, D. Merhof, A. Pai, B. Park, M. Perslev, R. Rezaiifar, O. Rippel, I. Sarasua, W. Shen, J. Son, C. Wachinger, L. Wang, Y. Wang, Y. Xia, D. Xu, Z. Xu, Y. Zheng, A. L. Simpson, L. Maier-Hein, M. J. Cardoso. "The Medical Segmentation Decathlon". *Nature Communications* 13.1 (July 2022), p. 4128. DOI: 10.1038/s41467-022-30695-9 (cit. on p. 42).

[2] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. "Attention is All you Need". *NIPS'17* (2017). DOI: 10.5555/3295222.3295349 (cit. on p. 25).

[3] U. Baid et al. "The RSNA-ASNR-MICCAI BraTS 2021 Benchmark on Brain Tumor Segmentation and Radiogenomic Classification". *arXiv.org* (Sept. 2021). arXiv:2107.02314 [cs]. DOI: 10.48550/arXiv.2107.02314 (cit. on p. 43).

[4] S. Bakas, H. Akbari, A. Sotiras, M. Bilello, M. Rozycki, J. S. Kirby, J. B. Freymann, K. Farahani, C. Davatzikos. "Advancing The Cancer Genome Atlas glioma MRI collections with expert segmentation labels and radiomic features". *Scientific Data* 4 (Sept. 2017), p. 170117. DOI: 10.1038/sdata.2017.117 (cit. on p. 43).

[5] I. N. Bankman. "Handbook of medical image processing and analysis (Second Edition)". *Academic Press* (Jan. 2009), p. 73. DOI: 10.1016/B978-012373904-9.50012-X (cit. on pp. 8, 12, 14).

[6] Y. Boykov, Marie-Pierre Jolly, M.-P. Jolly. "Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images". *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001* 1 (July 2001), pp. 105–112. DOI: 10.1109/iccv.2001.937505 (cit. on p. 15).

[7] N. Brady. *TechEmpower Web Framework Performance Comparison*. Feb. 2021. URL: https://www.techempower.com/benchmarks/#section=data-r20&test=query&l=zijzen-6bj (visited on 07/26/2023) (cit. on p. 31).

[8] T. B. Brown, B. Mann, N. Ryder, Nick Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, Pranav Shyam, G. Sastry, Girish Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, Aditya Ramesh, D. M. Ziegler, Daniel M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei. "Language Models are Few-Shot Learners". *NIPS'20* (May 2020). DOI: 10.5555/3495724.3495883 (cit. on p. 23).

[9] Bundesministerium für Gesundheit. *Marktzugangsvoraussetzungen für Medizinprodukte*. Aug. 2022. URL: https://www.bundesgesundheitsministerium.de/themen/gesundheitswesen/medizinprodukte/marktzugangsvoraussetzungen.html (visited on 08/09/2023) (cit. on p. 14).

[10] H. Chen, C. Gomez, C.-M. Huang, M. Unberath. "Explainable medical imaging AI needs human-centered design: guidelines and evidence from a systematic review". *npj Digital Medicine* 5.1 (Oct. 2022), p. 156. DOI: 10.1038/s41746-022-00699-2 (cit. on p. 14).

[11] J. Chen, E. C. Frey. "Medical Image Segmentation via Unsupervised Convolutional Neural Network". Jan. 2020. DOI: 10.48550/arXiv.2001.10155 (cit. on p. 13).

[12] J. P. Chin, V. A. Diehl, K. L. Norman. "Development of an instrument measuring user satisfaction of the human-computer interface". *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '88. New York, NY, USA: Association for Computing Machinery, May 1988, pp. 213–218. DOI: 10.1145/57167.57203 (cit. on p. 49).

[13] Chuanfei Hu, Xinde Li. "When SAM Meets Medical Images: An Investigation of Segment Anything Model (SAM) on Multi-phase Liver Tumor Segmentation". *arXiv.org* (2023). DOI: 10.48550/arxiv.2304.08506 (cit. on p. 26).

[14] E. C. Covert, K. Fitzpatrick, J. Mikell, R. K. Kaza, J. D. Millet, D. Barkmeier, J. Gemmete, J. Christensen, M. J. Schipper, Y. K. Dewaraja. "Intra- and inter-operator variability in MRI-based manual segmentation of HCC lesions and its impact on dosimetry". *EJNMMI Physics* 9.1 (Dec. 2022), p. 90. DOI: 10.1186/s40658-022-00515-6 (cit. on p. 13).

[15] D. Cheng, Ziyuan Qin, Zekun Jiang, Shaoting Zhang, Qicheng Lao, Kang Li. "SAM on Medical Images: A Comprehensive Study on Three Prompt Modes". *arXiv.org* (2023). DOI: 10.48550/arxiv.2305.00035 (cit. on pp. 27, 61, 62).

[16] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, N. Houlsby. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". *International Conference on Learning Representations* (2020). DOI: 10.48550/arXiv.2010.11929 (cit. on p. 25).

[17] G. Du, X. Cao, J. Liang, X. Chen, Q. Zeng, Y. Zhan. "Medical Image Segmentation based on U-Net: A Review". *Journal of Imaging Science and Technology* 64.2 (Mar. 2020), pp. 20508–1–20508–12. DOI: 10.2352/j.imagingsci.technol.2020.64.2.020508 (cit. on p. 14).

[18] S. Egger, T. Hossfeld, R. Schatz, M. Fiedler. "Waiting times in quality of experience for web based services". *2012 Fourth International Workshop on Quality of Multimedia Experience*. July 2012, pp. 86–96. DOI: 10.1109/QoMEX.2012.6263888 (cit. on pp. 56, 59).

[19] B. J. Erickson, P. Korfiatis, Z. Akkus, T. L. Kline. "Machine Learning for Medical Imaging". *Radiographics* 37.2 (Mar. 2017), pp. 505–515. DOI: 10.1148/rg.2017160130 (cit. on p. 13).

[20] A. Fedorov, R. Beichel, J. Kalpathy-Cramer, J. Finet, Jean Christophe Fillion-Robin, J.-C. Fillion-Robin, S. Pujol, C. Bauer, D. Jennings, F. M. Fennessy, M. Sonka, J. M. Buatti, Stephen R. Aylward, S. R. Aylward, S. R. Aylward, J. V. Miller, S. Pieper, R. Kikinis. "3D Slicer as an image computing platform for the Quantitative Imaging Network." *Magnetic Resonance Imaging* 30.9 (Nov. 2012), pp. 1323–1341. DOI: 10.1016/j.mri.2012.05.001 (cit. on pp. 12, 16).

[21] Y. Fu, Y. Lei, T. Wang, W. J. Curran, T. Liu, X. Yang. "A review of deep learning based methods for medical image multi-organ segmentation". *Physica Medica* 85 (May 2021), pp. 107–122. DOI: 10.1016/j.ejmp.2021.05.003 (cit. on p. 13).

[22] D. Gandlur, T. Lively, I. Stepanyan. *Fast, parallel applications with WebAssembly SIMD · V8*. Nov. 2022. URL: https://v8.dev/features/simd (visited on 07/30/2023) (cit. on p. 35).

[23] K. Glisson, M. Vilanova, F. Monsen. *Introducing Dispatch*. Feb. 2020. URL: https://netflixtechblog.com/introducing-dispatch-da4b8a2a8072 (visited on 07/26/2023) (cit. on p. 31).

[24] S. Gong, Y. Zhong, W. Ma, J. Li, Z. Wang, J. Zhang, P.-A. Heng, Q. Dou. "3DSAM-adapter: Holistic Adaptation of SAM from 2D to 3D for Promptable Medical Image Segmentation". *arXiv.org* (June 2023). DOI: 10.48550/arXiv.2306.13465 (cit. on p. 63).

[25] GROW.DDG1.D.4. *Guidance on Qualification and Classification of Software in Regulation (EU) 2017/745 – MDR and Regulation (EU) 2017/746 – IVDR*. Oct. 2019. URL: https://ec.europa.eu/docsroom/documents/37581 (visited on 08/09/2023) (cit. on p. 14).

[26] Y. Guo, Y. Gao, D. Shen. "Deformable MR Prostate Segmentation via Deep Feature Learning and Sparse Patch Matching". *IEEE transactions on medical imaging* 35.4 (Apr. 2016), pp. 1077–1089. DOI: 10.1109/TMI.2015.2508280 (cit. on p. 13).

[27] 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". *Medical Image Analysis* 35 (Jan. 2017), pp. 18–31. DOI: 10.1016/j.media.2016.05.004 (cit. on p. 13).

[28] S.-C. Huang, A. Pareek, M. Jensen, M. P. Lungren, S. Yeung, A. S. Chaudhari. "Self-supervised learning for medical image classification: a systematic review and implementation guidelines". *npj Digital Medicine* 6.1 (Apr. 2023), pp. 1–16. DOI: 10.1038/s41746-023-00811-0 (cit. on p. 13).

[29] Ivan Mikhailov, B. Chauveau, N. Bourdel, Alberto Bartoli. "A Deep Learning-Based Interactive Medical Image Segmentation Framework". *AMAI@MICCAI* (2022). DOI: 10.1007/978-3-031-17721-7_11 (cit. on pp. 13–15).

[30] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". *North American Chapter of the Association for Computational Linguistics* (2019). DOI: 10.18653/v1/n19-1423 (cit. on p. 23).

[31] W. Ji, S. Yu, J. Wu, K. Ma, C. Bian, Q. Bi, J. Li, H. Liu, L. Cheng, Y. Zheng. "Learning Calibrated Medical Image Segmentation via Multi-rater Agreement Modeling". *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. ISSN: 2575-7075. June 2021, pp. 12336–12346. DOI: 10.1109/CVPR46437.2021.01216 (cit. on p. 13).

[32] O. Jones. *Magnetic Resonance Imaging (MRI) Scanning - Principles - TeachMeAnatomy*. Apr. 2018. URL: https://teachmeanatomy.info/the-basics/imaging/magnetic-resonance-imaging-mri/ (visited on 08/14/2023) (cit. on p. 26).

[33] Jun Ma, Bo Wang. "Segment Anything in Medical Images". *arXiv.org* (2023). DOI: 10.48550/arXiv.2304.12306 (cit. on pp. 26, 27, 63).

[34] Junde Wu, Rao Fu, Huihui Fang, Yuanpei Liu, Zhao-Yang Wang, Yanwu Xu, Yueming Jin, T. Arbel. "Medical SAM Adapter: Adapting Segment Anything Model for Medical Image Segmentation". *arXiv.org* (2023). DOI: 10.48550/arxiv.2304.12620 (cit. on p. 63).

[35] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Doll'ar, Ross B. Girshick. "Masked Autoencoders Are Scalable Vision Learners". *Computer Vision and Pattern Recognition* (2021). DOI: 10.1109/cvpr52688.2022.01553 (cit. on p. 25).

[36] M. Kass, A. Witkin, D. Terzopoulos. "Snakes: Active contour models". *International Journal of Computer Vision* 1.4 (Jan. 1988), pp. 321–331. DOI: 10.1007/BF00133570 (cit. on p. 17).

[37] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, R. Girshick. "Segment Anything". *arXiv.org* (Apr. 2023). DOI: 10.48550/arXiv.2304.02643 (cit. on pp. 8, 9, 14, 23, 25).

[38] T. L. Kline, P. Korfiatis, M. E. Edwards, J. D. Blais, F. S. Czerwiec, P. C. Harris, B. F. King, V. E. Torres, B. J. Erickson. "Performance of an Artificial Multi-observer Deep Neural Network for Fully Automated Segmentation of Polycystic Kidneys." *Journal of Digital Imaging* 30.4 (May 2017), pp. 442–448. DOI: 10.1007/s10278-017-9978-1 (cit. on pp. 12–14).

[39] Konstantin Sofiiuk, Ilya A. Petrov, Anton Konushin. "Reviving Iterative Training with Mask Guidance for Interactive Segmentation". *International Conference on Information Photonics* (2021). DOI: 10.1109/icip46576.2022.9897365 (cit. on pp. 23, 27).

[40] M. Larobina, L. Murino. "Medical Image File Formats". *Journal of Digital Imaging* 27.2 (Apr. 2014), pp. 200–206. DOI: 10.1007/s10278-013-9657-9 (cit. on p. 20).

[41] J. R. Lewis. "IBM computer usability satisfaction questionnaires: Psychometric evaluation and instructions for use". *International Journal of Human–Computer Interaction* 7.1 (Jan. 1995), pp. 57–78. DOI: 10.1080/10447319509526110 (cit. on p. 49).

[42] R. Likert. "A technique for the measurement of attitudes". *Archives of Psychology* 22 140 (1932), pp. 55–55 (cit. on pp. 52, 54).

[43] A. K. Liu, A. Liu, S. Jiang, S. Sampath, A. Amini, J. Y. Wong, J. Wong. "Applying the Turing Test to contouring: Are Machine-Generated Contours Indistinguishable From Human Generated Ones?" *International Journal of Radiation Oncology Biology Physics* 105.1 (Sept. 2019). DOI: 10.1016/j.ijrobp.2019.06.2173 (cit. on p. 13).

[44] Liu, Qin, Xu, Zhenlin, Bertasius, Gedas, Niethammer, Marc. "SimpleClick: Interactive Image Segmentation with Simple Vision Transformers". *arXiv.org* (Oct. 2022). DOI: 10.48550/arxiv.2210.11006 (cit. on p. 27).

[45] J. Long, E. Shelhamer, T. Darrell. "Fully Convolutional Networks for Semantic Segmentation". 2015, pp. 3431–3440. URL: https://openaccess.thecvf.com/content_cvpr_2015/html/Long_Fully_Convolutional_Networks_2015_CVPR_paper.html (cit. on p. 13).

[46] Maciej A. Mazurowski, Haoyu Dong, Han Gu, Jichen Yang, N. Konz, Y. Zhang. "Segment Anything Model for Medical Image Analysis: an Experimental Study". *arXiv.org* (2023). DOI: 10.48550/arxiv.2304.10517 (cit. on p. 27).

[47] D. S. Mackin, D. Mackin, D. Mackin, X. Fave, L. Zhang, D. Fried, D. V. Fried, J. Yang, B. A. Taylor, B. E. Taylor, E. Rodriguez-Rivera, C. Dodge, Cristina Dodge, A. K. Jones, Aaron Kyle Jones, L. E. Court. "Measuring Computed Tomography Scanner Variability of Radiomics Features." *Investigative Radiology* 50.11 (Nov. 2015), pp. 757–765. DOI: 10.1097/rli.0000000000000180 (cit. on p. 14).

[48] H. McGrath, P. Li, R. Dorent, R. Bradford, S. Saeed, S. Bisdas, S. Ourselin, J. Shapey, T. Vercauteren. "Manual segmentation versus semi-automated segmentation for quantifying vestibular schwannoma volume on MRI". *International Journal of Computer Assisted Radiology and Surgery* 15.9 (Sept. 2020), pp. 1445–1455. DOI: 10.1007/s11548-020-02222-y (cit. on pp. 13, 14).

[49] B. H. Menze, A. Jakab, S. Bauer, J. Kalpathy-Cramer, K. Farahani, J. Kirby, Y. Burren, N. Porz, J. Slotboom, R. Wiest, L. Lanczi, E. Gerstner, M.-A. Weber, T. Arbel, B. B. Avants, N. Ayache, P. Buendia, D. L. Collins, N. Cordier, J. J. Corso, A. Criminisi, T. Das, H. Delingette, Ç. Demiralp, C. R. Durst, M. Dojat, S. Doyle, J. Festa, F. Forbes, E. Geremia, B. Glocker, P. Golland, X. Guo, A. Hamamci, K. M. Iftekharuddin, R. Jena, N. M. John, E. Konukoglu, D. Lashkari, J. A. Mariz, R. Meier, S. Pereira, D. Precup, S. J. Price, T. R. Raviv, S. M. S. Reza, M. Ryan, D. Sarikaya, L. Schwartz, H.-C. Shin, J. Shotton, C. A. Silva, N. Sousa, N. K. Subbanna, G. Szekely, T. J. Taylor, O. M. Thomas, N. J. Tustison, G. Unal, F. Vasseur, M. Wintermark, D. H. Ye, L. Zhao, B. Zhao, D. Zikic, M. Prastawa, M. Reyes, K. Van Leemput. "The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS)". *IEEE transactions on medical imaging* 34.10 (Oct. 2015), pp. 1993–2024. DOI: 10.1109/TMI.2014.2377694 (cit. on p. 43).

[50] P. Molino, Y. Dudin, S. S. Miryala. *Ludwig v0.2 Adds New Features and Other Improvements to its Deep Learning Toolbox*. July 2019. URL: https://www.uber.com/blog/ludwig-v0-2/ (visited on 07/26/2023) (cit. on p. 31).

[51] D. Müller, F. Kramer. "MIScnn: a framework for medical image segmentation with convolutional neural networks and deep learning". *BMC Medical Imaging* 21.1 (Jan. 2021), p. 12. DOI: 10.1186/s12880-020-00543-7 (cit. on p. 14).

[52] K. O'Shea, R. Nash. "An Introduction to Convolutional Neural Networks". *arXiv.org* (Nov. 2015). DOI: 10.48550/arXiv.1511.08458 (cit. on p. 25).

[53] Oliver Jones. *Anatomical Planes - Coronal - Sagittal - Transverse - TeachMeAnatomy*. Sept. 2022. URL: https://teachmeanatomy.info/the-basics/anatomical-terminology/planes/ (visited on 08/14/2023) (cit. on p. 20).

[54] J. N. Paranjape, N. G. Nair, S. Sikder, S. S. Vedula, V. M. Patel. "AdaptiveSAM: Towards Efficient Tuning of SAM for Surgical Scene Segmentation". *arXiv.org* (Aug. 2023). DOI: 10.48550/arXiv.2308.03726 (cit. on p. 63).

[55] C. Pinter, A. Lasso, G. Fichtinger. "Polymorph segmentation representation for medical image computing". *Computer Methods and Programs in Biomedicine* 171 (Apr. 2019), pp. 19–26. DOI: 10.1016/j.cmpb.2019.02.011 (cit. on p. 16).

[56] S. Primakov, A. Ibrahim, J. E. van Timmeren, Guangyao Wu, Simon A Keek, Manon Beuque, R. W. Y. Granzier, Elizaveta Lavrova, Madeleine Scrivener, Sebastian Sanduleanu, Esma Kayan, I. Halilaj, Anouk Lenaers, Jianlin Wu, René Monshouwer, Xavier Geets, H. A. Gietema, L. E. Hendriks, O. Morin, Arthur Jochems, H. C. Woodruff, P. Lambin. "Automated detection and segmentation of non-small cell lung cancer computed tomography images". *Nature Communications* 13.1 (June 2022). DOI: 10.1038/s41467-022-30841-3 (cit. on p. 13).

[57] R. Ranjbarzadeh, A. B. Kasgari, S. J. Ghoushchi, S. Anari, M. Naseri, M. Bendechache. "Brain tumor segmentation based on deep learning and an attention mechanism using MRI multi-modalities brain images." *Scientific Reports* 11.1 (May 2021), pp. 10930–10930. DOI: 10.1038/s41598-021-90428-8 (cit. on p. 8).

[58] F. Renard, S. Guedria, N. D. Palma, N. Vuillerme. "Variability and reproducibility in deep learning for medical image segmentation". *Scientific Reports* 10.1 (Aug. 2020), p. 13724. DOI: 10.1038/s41598-020-69920-0 (cit. on pp. 12, 13).

[59] O. Ronneberger, P. Fischer, T. Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by N. Navab, J. Hornegger, W. M. Wells, A. F. Frangi. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28 (cit. on pp. 13, 14).

[60] C. Rorden, M. Brett. "Stereotaxic display of brain lesions". eng. *Behavioural Neurology* 12.4 (2000), pp. 191–200. DOI: 10.1155/2000/421719 (cit. on p. 12).

[61] Saikat Roy, T. Wald, Gregor Koehler, Maximilian R. Rokuss, Nico Disch, Julius Holzschuh, David Zimmerer, K. Maier-Hein. "SAM.MD: Zero-shot medical image segmentation capabilities of the Segment Anything Model". *arXiv.org* (2023). DOI: 10.48550/arXiv.2304.05396 (cit. on pp. 27, 62).

[62] T. Sakinis, F. Milletari, H. R. Roth, P. Korfiatis, P. M. Kostandy, K. A. Philbrick, Kenneth A. Philbrick, Z. Akkus, Z. Xu, D. Xu, B. J. Erickson. "Interactive segmentation of medical images through fully convolutional neural networks." *arXiv.org* (Mar. 2019). DOI: 10.48550/arXiv.1903.08205 (cit. on pp. 14, 15).

[63] N. Shareef, D. L. Wang, R. Yagel. "Segmentation of medical images using LEGION". *IEEE transactions on medical imaging* 18.1 (Jan. 1999), pp. 74–91. DOI: 10.1109/42.750259 (cit. on pp. 12, 13).

[64] Sheng He, Rina Bao, Jingpeng Li, J. Stout, A. Bjørnerud, P. Grant, Yangming Ou. "Computer-Vision Benchmark Segment-Anything Model (SAM) in Medical Images: Accuracy in 12 Datasets". *arXiv.org* (2023). DOI: 10.48550/arXiv.2304.09324 (cit. on pp. 26, 27).

[65] B. A. Skourt, A. E. Hassani, A. Majda, Aicha Majda. "Lung CT Image Segmentation Using Deep Neural Networks". *Procedia Computer Science* 127 (Jan. 2018), pp. 109–113. DOI: 10.1016/j.procs.2018.01.104 (cit. on p. 8).

[66] C. M. Smith, J. C. Smith, S. K. Williams, J. J. Rodriguez, J. B. Hoying. "Automatic thresholding of three-dimensional microvascular structures from confocal microscopy images." *Journal of Microscopy* 225.3 (Mar. 2007), pp. 244–257. DOI: 10.1111/j.1365-2818.2007.01739.x (cit. on p. 15).

[67] Tao Zhou, Yizhe Zhang, Yi Zhou, Ye Wu, Chen Gong. "Can SAM Segment Polyps?" *arXiv.org* (2023). DOI: 10.48550/arxiv.2304.07583 (cit. on p. 26).

[68] Y. Varun. *Understanding DICE COEFFICIENT*. URL: https://kaggle.com/code/yerramvarun/understanding-dice-coefficient (visited on 07/02/2023) (cit. on p. 26).

[69] V. Vezhnevets, V. Konushin. ""GrowCut Interactive Multi-Label ND Image Segmentation By Cellular Automata". *Graphicon* 1 (Nov. 2004). URL: https://www.graphicon.ru/oldgr/en/publications/text/gc2005vk.pdf (visited on 08/09/2023) (cit. on p. 18).

[70] G. Wang, W. Li, M. A. Zuluaga, R. Pratt, P. A. Patel, M. Aertsen, T. Doel, A. L. David, Jan Deprest, J. Deprest, S. Ourselin, T. Vercauteren. "Interactive Medical Image Segmentation using Deep Learning with Image-specific Fine-tuning". *IEEE Transactions on Medical Imaging* 37 (2017). DOI: 10.1109/tmi.2018.2791721 (cit. on p. 15).

[71] R. Wang, T. Lei, R. Cui, B. Zhang, H. Meng, A. K. Nandi. "Medical image segmentation using deep learning: A survey". *IET Image Processing* 16.5 (2022), pp. 1243–1267. ISSN: 1751-9667. DOI: 10.1049/ipr2.12419 (cit. on p. 13).

[72] I. Wolf, M. Vetter, I. Wegner, T. Böttger, M. Nolden, M. Schöbinger, M. Hastenteufel, T. Kunert, H.-P. Meinzer. "The medical imaging interaction toolkit". *Medical Image Analysis* 9.6 (Dec. 2005), pp. 594–604. DOI: 10.1016/j.media.2005.04.005 (cit. on p. 18).

[73] J. Wu, S. Poehlman, M. D. Noseworthy, M. V. Kamath. "Texture feature based automated seeded region growing in abdominal MRI segmentation". *Journal of Biomedical Science and Engineering* 2.1 (Feb. 2009), pp. 1–8. DOI: 10.4236/jbise.2009.21001 (cit. on p. 15).

[74] Yichi Zhang, Rushi Jiao. "How Segment Anything Model (SAM) Boost Medical Image Segmentation?" *arXiv.org* (2023). DOI: 10.48550/arxiv.2305.03678 (cit. on p. 27).

[75] P. A. Yushkevich, J. Piven, H. C. Hazlett, R. G. Smith, S. Ho, J. C. Gee, G. Gerig. "User-guided 3D active contour segmentation of anatomical structures: Significantly improved efficiency and reliability". *NeuroImage* 31.3 (July 2006), pp. 1116–1128. DOI: 10.1016/j.neuroimage.2006.01.015 (cit. on pp. 12, 17).

[76] B. Zhao, Y. Tan, W. Y. Tsai, L. Schwartz, L. H. Schwartz, L. H. Schwartz, L. Lu. "Exploring Variability in CT Characterization of Tumors: A Preliminary Phantom Study." *Translational Oncology* 7.1 (Feb. 2014), pp. 88–93. DOI: 10.1593/tlo.13865 (cit. on p. 14).

[77] F. Zhao, X. Xie. "An overview on interactive medical image segmentation". *The Annals of the BMVA* 2013.7 (Aug. 2013). URL: http://www.bmva.org/annals/2013/2013-0007.pdf (cit. on p. 12).

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature