Rapid Cytomine: foundation models for interactive annotation in computational patholog

Auteur : Vanmechelen, Thibaud
Promoteur(s) : Geurts, Pierre; Marée, Raphaël
Faculté : Faculté des Sciences appliquées
Diplôme : Master en ingénieur civil en informatique, à finalité spécialisée en "intelligent systems"
Année académique : 2024-2025
URI/URL : http://hdl.handle.net/2268.2/23364

Master Thesis

Rapid Cytomine: foundation models for interactive annotation in computational pathology

Author:
Thibaud Vanmechelen

Supervisors:
Pierre Geurts
Raphaël Marée

Master:
Computer Science and Engineering, professional focus in Intelligent Systems

Academic Year: 2024 - 2025

Place: University of Liège - Faculty of Applied Sciences

Master's thesis completed in order to obtain the degree of Master of Science in Computer Engineering by Thibaud Vanmechelen

Acknowledgements

I would first like to thank Mr. Pierre Geurts and Mr. Raphaël Marée, my two supervisors, for their help, their guidance, and valuable feedback throughout my master's thesis. They provided ideas and directions to explore, reviewed my results, and allowed me to work with a high level of autonomy.

I also thank Mr. Ba Thien Lee for the technical support he offered during the integration into Cytomine.

I would like to thank Mr. Noé Gille, the student who worked on the original internship project, for the code and results on which I built my work.

I also thank all the professors at the University of Liège for the theoretical and practical knowledge, as well as the methodology they taught during my studies, which helped me complete this thesis.

Finally, I would like to thank my family for their support throughout this thesis and my studies.

Abstract

This thesis presents the integration of an interactive segmentation tool into the Cytomine platform, based on the Segment Anything Models (SAM and SAM2) published by Meta, in order to simplify the manual annotation process. Cytomine is a collaborative application which is designed for sharing and annotating large biomedical images (with a particular interest for histopathology). It allows users to interact with the large-scale images from digital pathology scanners or other sources, and to annotate as well as analyze them collaboratively. Collecting high-quality annotations in such specialized domains is a significant challenge, as it often requires the involvement of experts (such as researchers or medical professionals), whose time is limited and valuable. Therefore, providing an efficient segmentation tool is extremely important.

To address this challenge, both SAM and SAM2 were first evaluated and compared in a zero-shot setting. A series of fine-tuning experiments under various training configurations were then conducted to assess the performance sensitivity to different parameters, and to eventually identify the best-performing setting. Several post-processing strategies were also explored to enhance the mask quality and usability, and the possible advantages of integrating domain-specific encoders alongside SAM were also investigated.

The best-performing model was integrated into the latest release of Cytomine through new API endpoints as well as a new back-end server. To support future development, a tutorial was also created to guide users and developers through the process of modifying Cytomine to integrate custom API endpoints.

Contents

Acronyms

AI Artificial intelligence.

BCE Binary Cross-Entropy.

IoU Intersection-over-Union.

MAE Mean Absolute Error.

MSE Mean Squared Error.

SAM Segment Anything Model.

SAM2 Segment Anything Model 2.

ViT Vision Transformer.

WSI Whole Slide Image.

Chapter 1

Introduction

1.1 Context of the work

Over the past few years, the advancements in Deep Learning have brought a real revolution across many fields. They have the potential to change the way various tasks are carried out and opened up a completely new range of possibilities/opportunities for tasks that computers can perform.

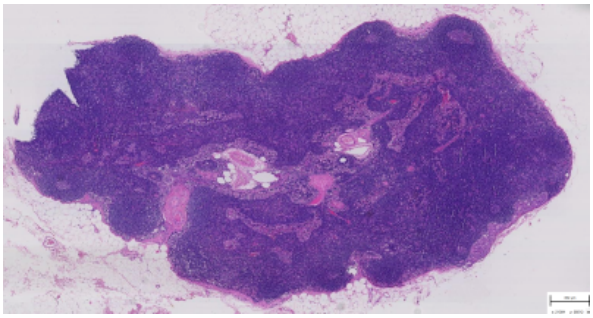
The medical field is an important domain which holds great potential to benefit from these breakthroughs and discoveries. Indeed, AI enables significant new capabilities that can help improve the quality of care that is provided to the patients. From drug discovery and robot-assisted surgery to diagnostic assistance, many tasks could see some considerable performance improvements thanks to AI, which could directly impact the global health and contribute to a better quality of life overall.

Medical image analysis is one particularly promising application of AI, since the performance of convolutional neural networks, and more recently, of transformers, has proven to be extremely strong and reliable. However, a persistent challenge in Deep Learning, and even more in the medical domain in general, is the availability of data. Indeed, these models generally require large amounts of data for their training; data that is both time-consuming and expensive to collect as it must be annotated by healthcare professionals (researchers, doctors, etc...).

In histopathology, an area of research in medical image analysis which focuses on the diagnosis and the study of diseases in tissues, this data scarcity issue becomes even more problematic. The researchers in this field often work with whole slide images (WSIs), which are extremely large scans of tissue samples (an example is shown in Figure 1.1). Analyzing such images is not only time-intensive due to their size but also because of the complexity and the irregularity of the structures to annotate. Compared to other medical fields like radiology, the objects found in histopathology images tend to be more irregular and oddly shaped. Annotating such objects for tasks like segmentation is particularly demanding, as one must carefully delineate each element. However, this delineation/segmentation is important because the size and shape of these elements can play a significant role in the diagnosis (and the underlying treatment).

For all these reasons, building large annotated datasets in histopathology is even more challenging than in many other medical domains, yet just as important if we want to train effective models for downstream tasks. One notable initiative addressing this challenge is Cytomine [Marée et al., 2016], an open-source web platform developed at the University of Liège to support collaborative analysis and annotation of WSIs, which will be detailed in the next section.

Figure 1.1: Example of a WSI. Source: Cytomine Uliège



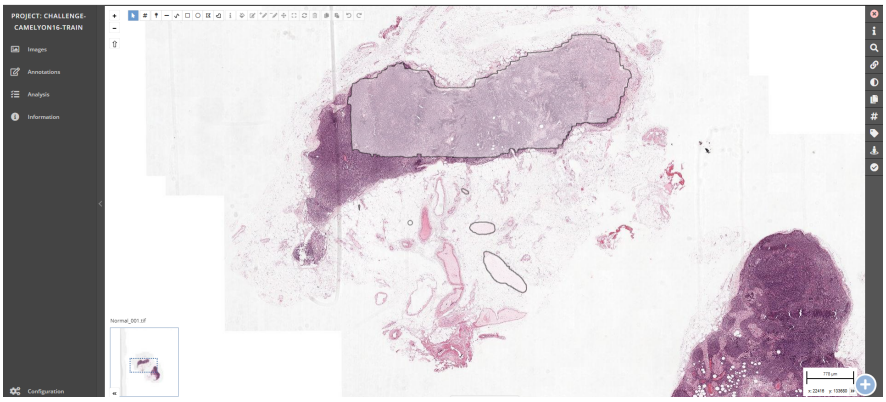
The task of assisting users in delineating/segmenting specific elements has been formalized as interactive segmentation, where a model takes some user inputs and generates the corresponding segmentation masks. The main objective of this task is to reduce the amount of manual work required and to accelerate the annotation process. In recent years, several tools and models have been developed to tackle this problem. Examples include TissueWand [Lindvall et al., 2020], SimpleClick [Q. Liu et al., 2023a], and more recently, two models introduced by Meta: the Segment Anything Model (SAM [Kirillov et al., 2023]) and Segment Anything Model 2 (SAM2 [Ravi et al., 2024]), released in 2023 and 2024 respectively. These two models are presented as some of the first foundation models for segmentation and demonstrate highly accurate zero-shot performance across a wide range of segmentation tasks. Consequently, they constitute a promising approach to simplify the annotation process in WSIs. Their release has attracted global interest from researchers, including those at the University of Liège. In this context, the goal of this master's thesis is to analyze the performance and capabilities of the two models developed by Meta in the context of interactive histopathology image segmentation, and to integrate the best solution into Cytomine.

1.2 Cytomine

As mentioned in the previous section (Section 1.1), Cytomine is an open-source software platform developed by Cytomine Corporation SA. The project was initiated at the University of Liège in 2010 and later evolved into a spin-off company. This software is designed to facilitate the collaborative analysis of WSIs and to enable semi-automatic processing of such images using machine learning algorithms. It provides a set of tools and algorithms for manual segmentation and image analysis. The application can be used for educational, research, or diagnostic purposes and has been adopted by several universities, including the University of Liège and the University of Louvain.

Unfortunately, Cytomine Corporation SA ceased its activities in July 2024. However, the University of Liège has since resumed leadership of both the development and maintenance of the Cytomine platform including its source code (https://github.com/cytomine).

Figure 1.2: Cytomine interface for annotation. Source: Cytomine Uliège



1.3 Problem Statement

The objective of this master's thesis was to analyze how SAM and SAM2, two foundation models for interactive segmentation, could facilitate the interactive segmentation of histopathology images, and to integrate the best-performing solution into Cytomine. More precisely, this work is a continuation of an internship conducted by Mr. Noé Gille [Gille, n.d.], whose goal was to evaluate the effectiveness of foundation models, more particularly SAM, for histopathology image segmentation. This thesis builds upon the results of that internship and aims to extend the analysis further.

The overall objective was divided into several milestones. The first step was to verify and validate the main findings of the internship regarding the zero-shot performance of SAM, and to extend the testing to include SAM2.

After that, I had to expand the experiments related to the fine-tuning of the models, going beyond what was originally done during the internship. Indeed, due to the limited duration of an internship, Mr. Gille did not have time to analyze the impact of several factors on the training process, such as the datasets used, the types of prompts, and other important elements. As part of my thesis, I had to investigate these aspects for both SAM and SAM2, and to compare the results. I also had to analyze how the outputs of such models could be made more user-friendly by applying some post-processing techniques to the segmentation masks.

The following step was to see how the performance of SAM and SAM2 could potentially be improved using domain-specific modules trained on histopathology datasets. Since the segment anything models were originally trained on everyday images, combining them with components specifically trained for histopathology could significantly enhance their performance.

Finally, the last stage of this thesis involved integrating the resulting model into Cytomine (at least in a prototype version), in order to verify whether it could be used effectively within the application environment.

1.4 Thesis Structure

The remainder of this master's thesis is divided into separate chapters, organized as follows:

- **Chapter 2: Background and Related Work**
  In this chapter, I first present the two models that have been analyzed during this thesis. Then, I review the literature to examine how these models have been adapted and used by other researchers.

- **Chapter 3: Project Context and Dataset Description**
  Here, I begin by summarizing the main takeaways from the initial internship and detailing the resources that have been reused. I then describe the datasets used in my experiments.

- **Chapter 4: Zero-shot Capabilities and Initial Evaluation**
  This chapter describes the experiments and results that I obtained when using SAM and SAM2 in a zero-shot setting, without any fine-tuning.

- **Chapter 5: Model Adaptation through Fine-tuning**
  In this chapter, I present the experiments and results obtained when fine-tuning the two models of interest. I also discuss other important aspects such as post-processing the segmentation masks.

- **Chapter 6: Model Extension with Specialized Encoders**
  This chapter describes the experiments and results that I obtained when combining SAM with domain-specific encoders trained on histopathology data.

- **Chapter 7: Integration into Cytomine**
  Here, I briefly describe the Cytomine platform, followed by the steps that I took to integrate the fine-tuned model into it.

- **Chapter 8: Conclusion**
  In the final chapter, I discuss the limitations of the experiments, summarize my contributions, and outline potential directions for future work. I also briefly present the tools and platforms used during this thesis.

- **An Appendix**
  This section contains additional figures and tables presenting the results obtained throughout the thesis, as well as a tutorial detailing the procedure to follow in order to integrate a new API into Cytomine.

Chapter 2

Background and Related Work

This chapter first provides an overview of SAM and SAM2, in order to give sufficient context for understanding the experiments and the modifications made to those models later on. In the second part of this chapter, I describe how researchers around the world have adapted, fine-tuned, or experimented with these two models, both in the context of medical imaging and in more general applications.

2.1 Segment Anything Model (SAM)

Segment Anything [Kirillov et al., 2023] was originally published by Meta on April 5, 2023. This paper introduced a new task called promptable segmentation, a new model named Segment Anything Model, and a new dataset for image segmentation: the SA-1B dataset, which contains 1 billion masks on 11 million images.

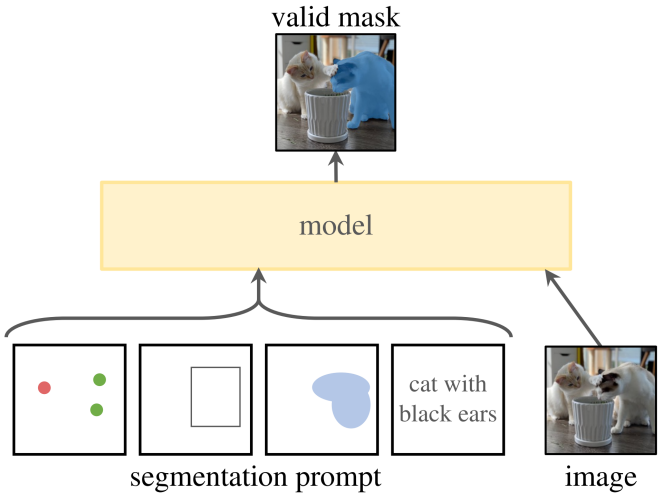
The promptable segmentation task aims to return a valid segmentation mask when given one or more prompts. This task is illustrated in Figure 2.1. As shown, the model can take various prompts as input, including points (positive and negative, representing foreground and background respectively), bounding boxes, masks, and even text. However, text prompts were only included as a proof of concept in the original version and were later removed in SAM2. The model then outputs one or more segmentation masks based on the input.

The idea behind this task is based on recent observations in deep learning. Researchers have observed that foundation models, especially in natural language processing, are capable of generalizing to new data distributions and performing few-shot learning, even on examples that they did not encounter during training. These capabilities come partially from the fact that these models can be prompted. Motivated by this, the researchers at Meta aimed to create a promptable foundation model for segmentation with similar generalization capabilities. Their goal was to develop a model that could be applied in domains with limited data availability.

To train such a model, they needed a very large dataset for segmentation. Existing datasets were not sufficient, so they decided to create their own. This was done through a combination of manual annotation and iterative annotation using the model itself.

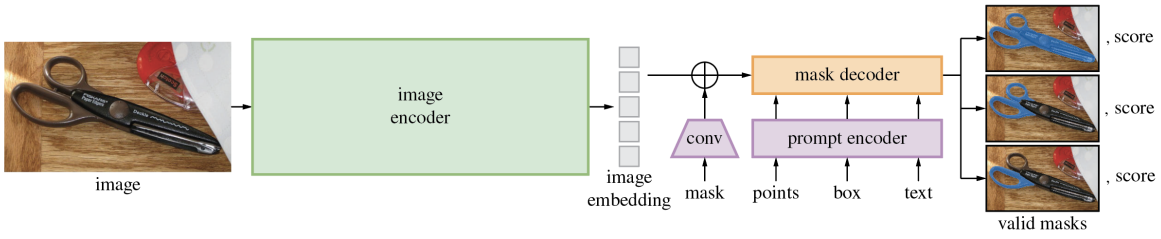
Since the SA-1B dataset is not directly related to the experiments of my thesis, I will not go into further detail.

Figure 2.1: Promptable segmentation task [Kirillov et al., 2023].



Regarding the architecture of the Segment Anything Model [1], it is composed of three main components: the image encoder, the prompt encoder, and the mask decoder. These components are shown in Figure 2.2.

Figure 2.2: Segment Anything Model Architecture [Kirillov et al., 2023].



The image encoder is a pre-trained Vision Transformer (ViT) that has been adapted to handle high-resolution images (up to 1024 × 1024 pixels). It produces an image embedding of size 64 × 64 with 256 channels from the input image. In fact, the researchers developed three versions of the image encoder, each with an increasing number of parameters: ViT-B (approximately 89 million parameters), ViT-L (approximately 308 million), and ViT-H (approximately 637 million).

[1]Note that I will only provide an overview, as an in-depth analysis is outside the scope of this thesis.

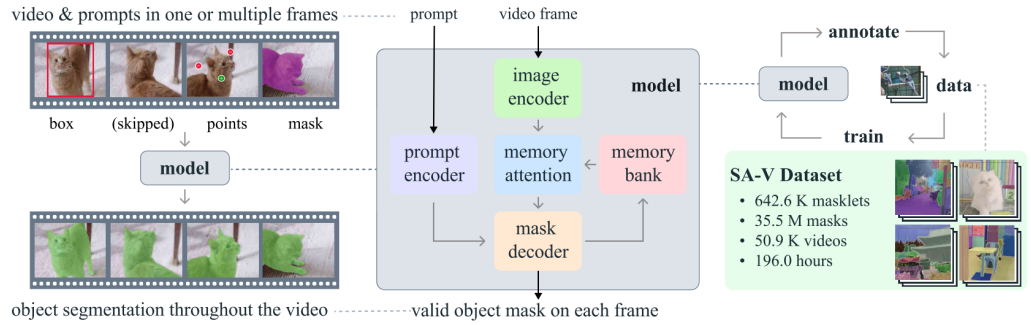
The prompt encoder handles the different types of prompts, distinguishing them into two categories: sparse prompts (points, boxes, text) and dense prompts (masks). Dense prompts are first encoded using a convolutional network into a feature map, then, this feature map is summed element-wise with the image embedding. For sparse prompts, the prompt encoder uses learned embeddings combined with positional encodings. The extension of SAM to support text prompts simply leverages a text encoder from CLIP [Radford et al., 2021].

Finally, the mask decoder uses a modified transformer decoder block architecture followed by a dynamic mask prediction head. It outputs segmentation masks along with predicted Intersection-over-Union (IoU) scores, which can be interpreted as confidence measures for each mask. The decoder can produce either a single mask or up to three masks. Indeed, since promptable segmentation is inherently ambiguous, a single prompt may correspond to multiple valid interpretations, for example, as shown in Figure 2.2, a point prompt on the scissor could result in several plausible segmentations. To address this issue, SAM includes an option to output multiple masks, allowing the user to select the most appropriate one.

2.2 Segment Anything Model 2 (SAM2)

A year after releasing SAM, Meta researchers released its successor, SAM2, on July 29, 2024. This new model is an update of the original SAM, with a modified architecture and additional capabilities. It is presented as a new foundation model designed to solve the promptable segmentation task in both images and videos. The main contributions of this work are the creation of SA-V, the largest video segmentation dataset to date, the introduction of the SAM2 model, and a slight update to the promptable segmentation task introduced in the original paper. These three contributions are illustrated in Figure 2.3.

Figure 2.3: SAM2 Task, Model, and Dataset [Ravi et al., 2024].



(a) Task: promptable visual segmentation **(b) Model:** Segment Anything Model 2 **(c) Data:** data engine and dataset

Regarding the promptable segmentation task described in the original article, and that I summarized in the previous section (Section 2.1), SAM2 introduces an important modification. The task is now referred to as promptable visual segmentation, which "generalizes image segmentation to the video domain" [Ravi et al., 2024]. The model

continues to take prompts (points, boxes, or masks [2]) as input but these prompts can now be placed on any frame of a video. These input prompts define a target segment, for which the model predicts a spatio-temporal mask (that the researchers call "masklet"). Once a masklet is predicted, it can either be refined by providing new prompts on other frames or tracked automatically by the model. This introduces a temporal dimension to the original task. To unify the handling of both images and videos, the authors treat an image as a single-frame video.

In terms of performance, the new SAM2 model is not only more accurate (on the authors' datasets) but also up to six times faster than the original SAM, which is highly interesting in interactive use cases. In the context of video segmentation, it achieves higher accuracy while requiring up to three times fewer user interactions compared to previous models. However, since my thesis does not focus on video segmentation, I will not explore further the technical details of that part.

Figure 2.4: Segment Anything Model 2 Architecture [Ravi et al., 2024].

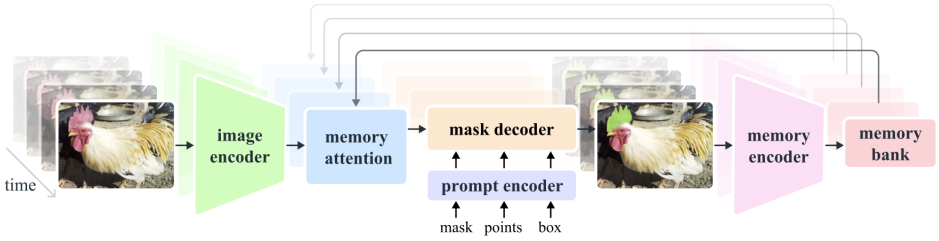


Figure 2.4 illustrates the architecture of SAM2. Like SAM, it includes an image encoder, a prompt encoder, and a mask decoder. However, three new modules have been added: memory attention, a memory encoder, and a memory bank. These components enable real-time video processing by providing the model with a memory that stores information about the target object and past interactions. This memory allows the model to generate masklets throughout the video and to correct/refine them based on observations. However, when applied to images (which are treated as single-frame videos) the memory remains empty, and the model behaves like the original SAM.

The image encoder in SAM2 is now a MAE pre-trained Hiera encoder [Ryali et al., 2023]. This hierarchical encoder uses several feature scales during the processing. Unlike in SAM, the image embeddings produced by the encoder are not fed directly to the decoder. Instead, they are first "conditioned on memories of past predictions and prompted frames" [Ravi et al., 2024]. These memories are generated by the memory encoder and stored in the memory bank for future use.

The memory attention module is responsible for conditioning the current frame's features on the past frames ones, past predictions, and new prompts. It takes the image embedding from the Hiera encoder and conditions it using the contents of the memory bank, producing the final embedding which will be passed to the mask decoder.

[2] In SAM2, the ability to use textual prompts was removed.

The prompt encoder used in SAM2 is identical to the one from the original model However, as already mentioned, textual prompts are no longer supported. The mask decoder is also very similar to the one in SAM, but now it includes an additional head that predicts whether the target object is present in the current frame. Indeed, in video segmentation, the object may disappear at some point.

Finally, the memory encoder encodes the current frame into the memory bank. It does so by using a convolutional module to downsample the output mask, which is then added element-wise to the unconditioned frame embedding from the Hiera encoder. After that, additional convolution layers are used to fuse this information. The memory bank itself is made of two FIFO queues, one for recent frames and one for prompted frames, along with a list of object pointers that provide semantic information about the tracked objects.

2.3 Literature Overview

Due to the recent emergence of foundation models for segmentation, both SAM and SAM2 have attracted significant interest from the research community around the globe. Although these models were initially trained on natural images rather than medical ones, recent works have begun to investigate their applications in the medical domain. Some of these studies specifically focuses on their use in histopathology, which is the primary focus of this master's thesis, while many others investigate more general applications in biomedical imaging. [Lee et al., 2024] offers a comprehensive analysis of several adaptations and integrations of SAM, focusing mainly on publications released in the six months following SAM's release; it highlights the large amount of work done by the scientific community on this model.

2.3.1 Zero-Shot Performance of SAM in Medical Imaging and Pathology

Several papers, such as [Mazurowski et al., 2023], [Deng et al., 2023], and [Y. Zhang et al., 2024], have evaluated the zero-shot performance of SAM in the medical field. Their results are generally consistent.

These studies show that bounding box prompts lead to better segmentation results than point prompts. Indeed, they provide more information about the object's size compared to the points. This conclusion is also supported by [Ren et al., 2023], which evaluated SAM for overhead imagery. SAM's performance varies significantly depending on the specific medical dataset, indicating that its results are not uniform across different medical domains. In general, the model performs better on well-circumscribed objects and when prompts are clear and unambiguous, which is not surprising. When these conditions are not met, performance can drop significantly.

In [Mazurowski et al., 2023], SAM outperformed other interactive segmentation methods such as RITM [Sofiiuk et al., 2021], SimpleClick [Q. Liu et al., 2023b], and FocalClick [X. Chen et al., 2022] in single-point prompt settings. However, when several point prompts are used iteratively, these other methods tend to improve more than SAM, sometimes surpassing it. This suggests that SAM's performance is highly determined by the first prompts, and is not improved significantly with further prompting. The paper also notes that SAM performs slightly better on larger objects, which is an important result as histopathology can involve both large and small objects. The paper hypothesizes that the generalization performance of SAM are due to the fact that it learned the concept of "object", enabling it to segment unfamiliar objects as long as they are well delineated.

The paper [Deng et al., 2023] also focuses on the zero-shot performance of SAM, but this time applies the model directly to digital pathology WSIs. Their findings show that SAM achieves great segmentation performance for large, connected objects, but is not always effective for dense instance segmentation (when different objects are closely grouped together), even when many prompts are used. The authors also highlight several limitations of the model for digital pathology, notably the image resolution (indeed, SAM only accepts 1024×1024 images, while WSIs are much larger), the lack of multi-scale handling, which would be useful in pathology, and the fact that prompt selection is not straightforward, especially when trying to choose effective prompts. However, the paper [Ren et al., 2023] mentions that artificially upsampling the images can help improve performance when dealing with small objects or low-resolution data, this could help dealing with the different resolution of the WSI. They also hypothesize that SAM could be biased by the object size. Moreover, both of these papers emphasize that the position of point prompts inside the ground truth mask does not seem to have much influence on the segmentation accuracy.

Finally, the paper [Y. Zhang et al., 2024], which summarizes the main findings on SAM from several studies, reports that negative point prompts might slightly reduce performance in certain tasks, especially when the background objects closely resemble to the target.

2.3.2 Fine-Tuning and Adapting SAM

After evaluating the zero-shot performance of SAM, several papers have gone further by attempting to fine-tune or modify the model to improve its performance. Some of these studies focus on adapting SAM for the medical domain in general, while others try more specifically on histopathology and digital pathology.

For the general medical domain, two approaches are presented in [Ma et al., 2024] and [Wang et al., 2023], which propose respectively, the MedSAM and SAM$^{\text{Med}}$ frameworks. MedSAM was trained on 1.5 million medical image-mask pairs and outperforms the original SAM on several medical segmentation tasks. However, only a very small portion of its dataset (9,317 image-mask pairs, which is equivalent to 0.6%) is related specifically to pathology. Therefore, it might not perform as well on histopathology

data. Since WSIs have much higher resolution than SAM's input limit, they performed patch extraction in order to fine-tune the model. Moreover, MedSAM is optimized for bounding box prompts, which led to the best results in the zero-shot settings.

On the other hand, SAM^{Med} is composed of two separate modules: SAM Assist, designed for interactive segmentation, and SAM Auto, for automatic segmentation. The SAM Assist module was trained by introducing a new prompt encoder, while keeping both the image encoder and the mask decoder frozen during the training. The automatic version, SAM Auto, uses SAM Assist and adds to it a prompt generator that automatically provides input to the assist module to generate the final segmentation masks. This framework demonstrates an efficient way to adapt SAM to medical tasks, and shows that not all parts of the model need to be fine-tuned in order to improve the performance.

Other papers also avoid fine-tuning directly SAM's image encoder as this is highly computationally intensive. [J. Zhang et al., 2023] addresses this by keeping the SAM image encoder frozen and adding a domain-specific encoder trained on digital pathology datasets. This method enables to introduce pathology-specific features without having to retrain the full model. However, to integrate the new encoder, some dimensionality reduction was required. This paper also shifts the focus from interactive to semantic segmentation. Indeed, the authors replace the original prompt encoder with a prompt generator based on trainable class prompts, but this use case falls outside the scope of my master's thesis, since it focuses on interactive segmentation.

2.3.3 Comparing SAM and SAM2

The study in [Sengupta et al., 2025] compares SAM and SAM2 for medical image segmentation. However, the set of datasets used do not include any pathology dataset, so the findings must be taken with a pinch of salt. The evaluation uses point prompts, which are known to be suboptimal. The results show that SAM2 performs better in some cases (e.g., MRI), but worse in others (e.g., ultrasound, where the image contrast is low). Both models struggle with over-segmentation when object boundaries are unclear. The results seem inconsistent with the ones of the original SAM2 paper [Ravi et al., 2024] and other studies like [Zhu et al., 2024] and [M. Zhang et al., 2024], which claim consistent improvements with SAM2. Therefore, it is important to compare the zero-shot performance of these two models in a more general context than the one of [Sengupta et al., 2025], in order to get a better idea of their true performance.

2.3.4 Fine-Tuning and Adapting SAM2

The paper [Zhu et al., 2024] is an update of the original [Ma et al., 2024]. In this work, the authors introduce MedSAM-2, which builds upon the previous MedSAM model by leveraging SAM2 instead of SAM. The updated model is now capable of handling both 2D and 3D medical image segmentation tasks. The user only needs to provide a prompt for one specific image that targets a specific object. After that, the model can automatically segment the same object in other images. This is made possible by adapting SAM2's video segmentation capabilities to treat 3D scans as video sequences.

Overall, MedSAM-2 demonstrates better performance than the original MedSAM model. Moreover, a key takeaway from their experiments is that SAM2 achieves better scores than SAM, even without additional fine-tuning.

The paper [M. Zhang et al., 2024] is also an update, building upon the original SAM-Path model proposed in [J. Zhang et al., 2023]. While it still focuses on semantic segmentation rather than interactive segmentation, it introduces several important modifications to improve the performance. First, the model incorporates UNI, the largest pretrained vision encoder for histopathology at the time, developed by the Mahmood Lab (affiliated to Harvard) [R. J. Chen et al., 2024].

Moreover, the authors replace the original prompt encoder with a module based on Kolmogorov-Arnold Networks (KANs) [Z. Liu et al., 2025], which uses trainable prompt tokens. These networks rely on activation functions with learned parameters rather than fixed ones. During the training, the image and prompt encoders are kept frozen, while only the mask decoder, an alignment module, and the prompt token module are trained. Overall, SAM2-Path performs better compared to the original SAM-Path model.

2.3.5 Conclusion

To sum up, despite the promise of foundation models like SAM and SAM2, their direct application to histopathology still faces several issues that could cause significant performance drops. These issues include the handling of high-resolution WSIs, dense and small instance segmentation, and adapting to multi-scale tissue structures. This thesis seeks to build on some of these findings, aiming to improve segmentation accuracy in histopathological images, and integrating the best resulting model into the Cytomine platform.

Chapter 3

Project Context and Dataset Description

In this chapter, I detail important aspects of the context surrounding my master's thesis. In particular, I begin by describing some of the initial results from the internship conducted by Mr. Noé Gille. Then, I present the parts of his codebase that I reused for my experiments (Section 3.1). After that, I explain the various evaluation metrics used throughout my work (Section 3.2). Finally, I describe the datasets used in my experiments (Section 3.3).

3.1 Existing Codebase and Initial Results

During his internship, Mr. Gille focused on analyzing SAM in two different settings. The first one involved a zero-shot assessment, where he experimented with the various prompt types to evaluate the model's default performance. After that, he tried to fine-tune the model to see if its performance could be further improved.

In both experiments, he encountered some difficulties. First, the size of WSIs can be extremely large, potentially exceeding one hundred thousand pixels in each dimension, whereas SAM only accepts 1024×1024 images. Moreover, the objects within the WSIs can vary greatly in size, ranging from a few hundred pixels to several million pixels. This makes it necessary to handle images at different scales, as the segmentation is not consistent across those object sizes.

To address these challenges, Mr. Gille adopted a standard approach which is commonly used when working with WSIs: extracting patches around regions of interest to reduce the image size. For both testing and fine-tuning, he extracted patches around the annotations and resized them to fit the input size of SAM.

Since SAM is a promptable model, he also needed to generate prompts for each image. He did so using the corresponding ground truth masks, from which he computed prompts. I will describe this process in more detail later in this section.

3.1.1 Prompting Experiments

One of the key findings of his prompting experiments was that bounding boxes performed better than point prompts alone, a result which is also consistent with findings in the literature, as in [Deng et al., 2023]. This makes sense: bounding boxes provide more spatial context about the object to segment, they give information about both the size and the location. This is especially important during inference, where the true annotations are unavailable and point prompts alone do not give information about the size which is problematic for the image patch extraction. However, the best performance was achieved when combining bounding boxes and positive point prompts. Adding negative point prompts did not yield further improvements and, in some cases, slightly degraded the performance. He also found that the placement of positive point prompts had little effect on the performance. More particularly, he tested two approaches: one where the point prompts were concentrated around the object's center of mass, and another where points were randomly sampled within the mask. The results were very close.

When using mask prompts, the best performance was obtained by combining them with bounding boxes and both positive and negative points. However, this method is clearly not practical in a real interactive segmentation use case, as it requires significant effort from the user. It also worked best on well-defined and moderately sized objects.

Therefore, Mr. Gille identified that the most interesting prompt combination in terms of user effort and accuracy was bounding boxes combined with positive point prompts.

3.1.2 Finetuning Experiments

For the fine-tuning phase, Mr. Gille chose to fine-tune only the prompt encoder and mask decoder of SAM. This is because the image encoder is the most computationally intensive component, with the highest number of parameters. In contrast, the prompt encoder and decoder contain less than 10 million parameters combined. Fine-tuning the image encoder would also require GPUs with very high memory (e.g., A100 with 80GB VRAM).

He conducted three fine-tuning experiments using three different datasets (that I describe later in this chapter). Each experiment followed a different data strategy:

- In-domain experiment: The model was trained and tested on the same dataset, to measure performance improvement when training and test data come from the same distribution.
- Cross-domain experiment: The model was trained on two datasets and tested on the third (held-out) dataset, to evaluate the generalization.
- All-in experiment: The model was trained on all datasets to assess the global performance.

In each experiment, the model was trained using all prompt types to ensure the performance would be enhanced for all of them. As expected, the best results were achieved in the third setup (all datasets), while the worst were observed in the first (single dataset). Overall, fine-tuning led to performance improvements of up to 55% compared to zero-shot prompting, highlighting the importance of fine-tuning the model.

3.1.3 Reused Codebase from the Internship

Regarding the part of the code [1] that I reused from the internship: first of all, I directly reused all the components related to retrieving the datasets from Cytomine, as well as the dataset manipulation methods. Indeed, the original repository already included code to extract the various annotations from the WSIs in each dataset, thus, I saw no need to reimplement that part.

The way this code works is fairly straightforward. It loops through all annotations linked to a project on Cytomine (with each project corresponding to a dataset) and, for each annotation, creates a window around it. This window is centered on the annotation and scaled according to its size. A dezoom factor is applied so that the window is not tight around the annotation; the idea is to create a broader region of interest that includes some of the surrounding context. Indeed, this is important, because having a larger context can help the model distinguish the object from the background more clearly. Once created, the image patch is resized to fit the resolution expected by SAM (1024×1024) and stored in the dataset locally.

Regarding the dataset manipulation, and more specifically the splitting into training, validation, and test sets, I also reused the code from the internship. This was indeed important because Cytomine is a collaborative annotation platform, and two issues could arise if the data is not split carefully. The first issue is that a specific object in an image may have been annotated by several people, leading to multiple annotations for this same element. These annotations must not be split across different subsets, as it would cause data leakage: the model could end up being trained and tested on the same object, which would bias the performance evaluation. The second issue is that annotations from the same WSI should be avoided across different subsets. Even if the annotations themselves are different, they still come from the same image, and this could introduce some bias. To avoid both of these problems, all annotations related to the same image are grouped together into the same subset.

The code related to the custom dataset class and prompt generation was also reused from the internship. In fact, Mr. Gille had implemented a set of methods for generating the various types of prompts, and I directly reused those. For bounding boxes, the minimum and maximum coordinates of the ground truth mask are used to define a box (represented by the top-left and bottom-right points) around the object. To simulate slight inaccuracies that might occur when an actual human draws a bounding box, small random perturbations are added to these coordinates. More specifically, the code samples four random integers in the interval $[-20; 20]$ and applies those values as

[1] See Section 8.3 for the GitHub repository URL.

pixel-level shifts to the left, right, top, and bottom sides of the box, respectively. This added randomness helps the model learn to be robust to imperfect user inputs.

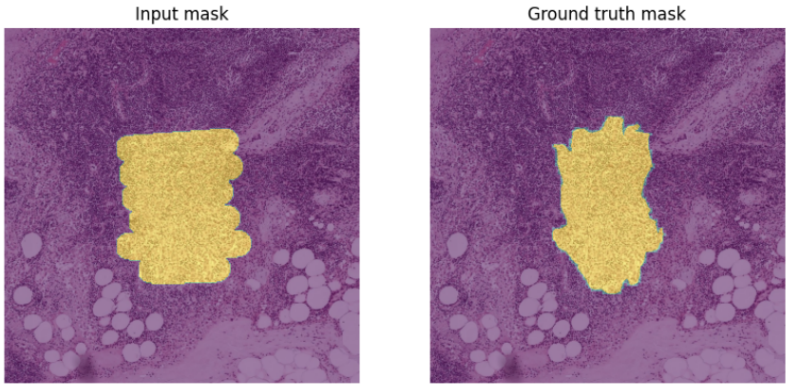
For point prompts, there are two different processes depending on the type of point: positive or negative. For positive points, I reused the setting where the sampling favors points near the center of the object. I found this approach more appropriate for simulating realistic interactive segmentation: in real use, users are more likely to click near the center of the object that they want to segment rather than near the edge. And since the findings of [Deng et al., 2023] showed that the exact placement of point prompts has little influence on model performance in medical image segmentation, this generation setting will likely not affect badly the accuracy of the model. The probability of a pixel x being selected as a prompt is given by:

$$P(X = x) = \frac{\min\left\{\text{dist}(x, y)^k \;\middle|\; y \in I,\ y = 0\right\}}{\sum_{z \in I} \min\left\{\text{dist}(z, y)^k \mid y \in I,\ y = 0\right\}}$$

Here, X is a random variable over the set of pixels x that belong to the mask I. This formula computes the distance to the nearest background pixel (where $y = 0$) and normalizes the result. The output is a probability distribution that favors pixels closer to the object's center. In this formula, I set k to 4.

For the negative point prompts, they are chosen randomly from pixels outside the ground truth mask but within the bounding box. Indeed, selecting completely random points outside the mask would not make much sense, as they could end up very far from the object, which would not reflect typical user behavior. In practice, users would likely place negative points near the object, to signal boundaries or adjacent regions.

Figure 3.1: Example of mask (from the internship report)



For the mask prompts, Mr. Gille assumed a setting where the user has access to a "paint-brush" tool and highlights the annotation using horizontal lines. An example of such a mask is shown in Figure 3.1. As we can see, the mask prompt consists of six horizontal lines spanning the full height of the annotation. I reused this same method and also explored additional mask generation techniques. These will be detailed in later chapters.

Finally, I also reused the part of the code to declare the slightly modified version of SAM used for the fine-tuning, as well as the training and testing scripts even thought I had to heavily modify them.

3.2 Evaluation Metrics

To evaluate the model performance, I initially selected the same four metrics used by Mr. Noé Gille to allow for easier comparison of results: Dice score, IoU score, Precision, and Recall. These are commonly used metrics for segmentation tasks and were therefore appropriate.

In addition, I introduced several other evaluation metrics. The first is the inference time, since, as discussed in the Background chapter, one of the main advantages of SAM2 over SAM is its speed; up to six times faster compared to SAM. This is particularly important because the final model is intended to be integrated into the Cytomine platform for interactive use. Speed is therefore essential to ensure a good user experience.

I also introduced several qualitative metrics based on various shape descriptors, to evaluate the "usability" of the masks for real users. Indeed, in practice and in the context of interactive segmentation, a mask should not only be accurate but also easy to edit. To assess this, I used the following descriptors: compactness, solidity, perimeter smoothness, number of connected components, and size retention. In the rest of this thesis, to combine these metrics across the different datasets that I use, I follow a consistent approach: the metrics are averaged over all the individual elements across the datasets, rather than computing separate averages per dataset.

In the following paragraphs, I describe each metric in more details and provide a corresponding figure to facilitate visualization. In each of these figures, hatched zones (if present) represent the regions used in the computation.

The Dice score measures the overlap between the predicted mask and the ground truth mask. It is computed as follows:

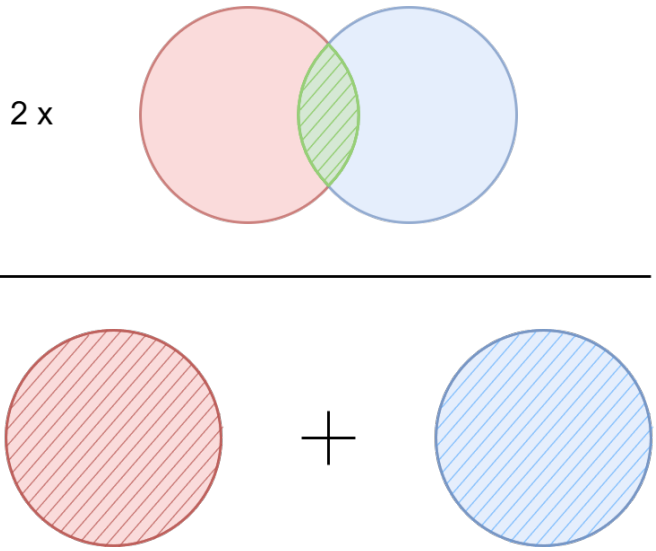
$$Dice = \frac{2 * TP}{FP + 2 * TP + FN}$$

Where:

- TP: true positives (= number of pixels correctly predicted which belongs to the object, shown in hatched green in Figure 3.2)
- FP: false positives (= number of pixels predicted as being part of the object but which are not)
- FN: false negatives (= number of pixels predicted as not being part of the object but which are)

The factor 2 in the numerator makes this score more sensitive to the overlap, and therefore more focused on the similarity between the two masks. It is more forgiving regarding false positives and false negatives compared to the IoU, since it also includes a factor 2 in the denominator, giving more weight to the true positive pixels. A Dice score of 0 means that there is no overlap between the two masks, while a score of 1 means perfect overlap. Overall, this score is better suited for small objects or regions, which is particularly interesting in this use case, as masks in histopathology can often be very small.

Figure 3.2: Dice score



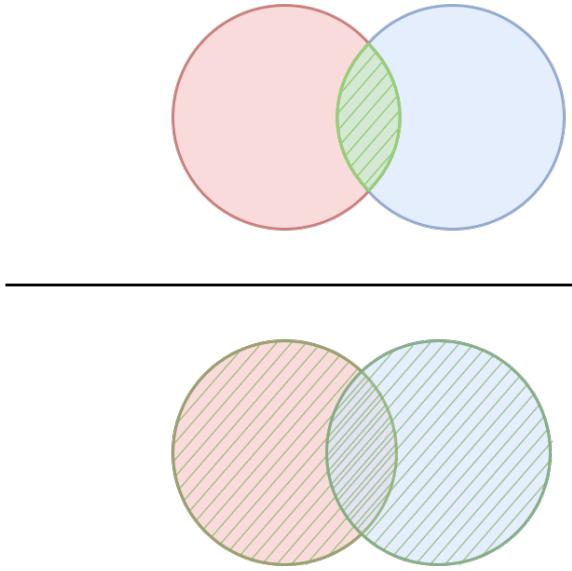
The IoU score also measures how much of the predicted object pixels overlap with the ground truth object pixels, relative to the total area covered by both masks. It is given by this formula:

$$IoU = \frac{TP}{FP + TP + FN}$$

As with the Dice score, an IoU score of 0 means no overlap, while a score of 1 means perfect overlap. Compared to the Dice score, it penalizes overpredictions (FP) and underpredictions (FN) more, since the denominator does not give extra weight to the true positives. Therefore, it is more penalizing for small objects but gives a better estimate of over- or underprediction.

In Figure 3.3, the hatched region at the numerator represents the overlap between the two sets, while the one at the denominator represents the union of those sets.

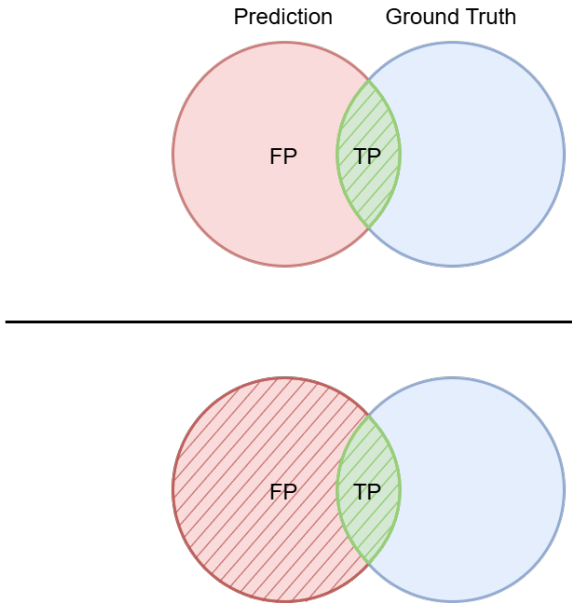
Figure 3.3: IoU score



The precision measures how many of the predicted positive pixels are actually correct. It represents the proportion of correctly predicted positive pixels. The formula is the following:

$$Precision = \frac{TP}{FP + TP}$$

Figure 3.4: Precision

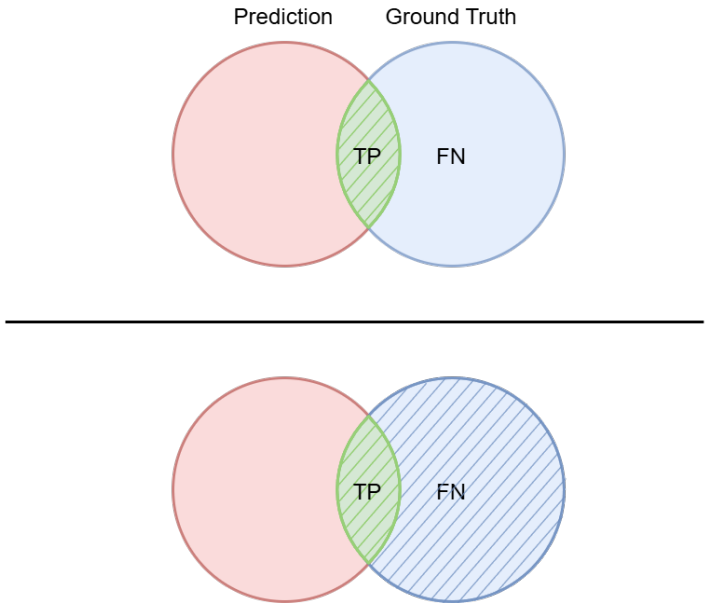


The true positive pixels correspond to the intersection between the predicted mask and the ground truth, while the false positive pixels are the predicted ones that do not overlap with the ground truth.

The recall measures how many of the actual positive pixels in the ground truth were correctly identified. It is computed as follows:

$$Recall = \frac{TP}{FN + TP}$$

Figure 3.5: Recall



Here, the true positive pixels still correspond to the intersection between the predicted mask and the ground truth, but in this case, the false negative pixels are those that belong to the ground truth but that are not covered by the prediction.

The inference time is simply measured using Python `time` library by recording the duration between sending the input to the model and receiving the output mask. It shows how fast the model is in practice.

Now looking at the more qualitative measures, compactness measures how tightly packed a shape is. Basically, it compares the shape of the mask to a circle (which is the most compact shape possible) and outputs a score indicating how close the mask is to a circle. A value of 1 corresponds to a perfect circle. It is computed as follows:

$$Compactness = \frac{4\pi * Area}{Perimeter^2}$$

In Figure 3.6, two shapes are shown to illustrate the concept of compactness. The first shape (a), which is a circle, is perfectly compact because it maximizes area relative to its perimeter. On the other hand, the second shape (b) is highly irregular, and thus has a lower compactness due to its larger perimeter relative to its area.

Figure 3.6: Compactness. Source: University of Iowa

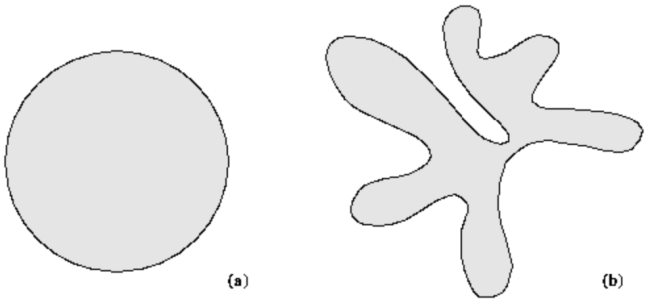


Figure 6.25 *Compactness: (a) Compact, (b) non-compact.*

Solidity is simply the ratio between the area of the object and the area of its convex hull (the smallest convex shape that encloses the object). This measure is useful to identify holes or dents in the predicted masks. It is computed as follows:

$$Solidity = \frac{Area}{Convex\ Hull\ Area}$$

Figure 3.7: Solidity. Research Gate

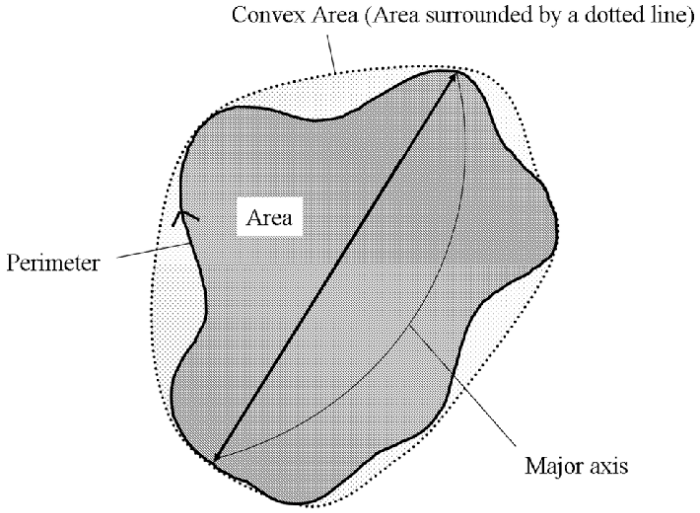
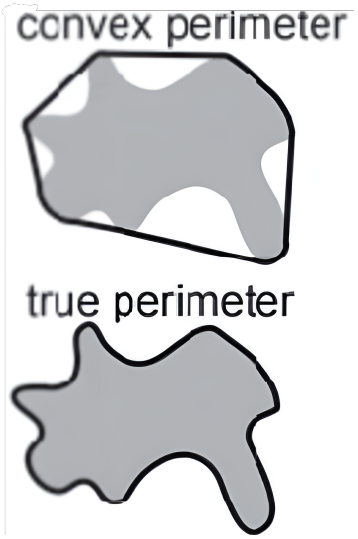


Figure 3.7 illustrates the concept of solidity by showing the area of the object as well as its convex hull. The convex area which is outlined by the dotted line represents the smallest convex shape that fully encloses the object. Therefore, the solidity score is computed as the ratio between the actual area of the object and this convex area.

The perimeter smoothness ratio compares the perimeter of the output mask to the one of a polygon fitted around the mask. It helps identify whether the boundaries are irregular or noisy and gives a general indication of the smoothness of the contour. It is calculated as:

$$Perimeter\ smoothness\ ratio = \frac{perimeter}{polygon\ perimeter}$$

Figure 3.8: Perimeter smoothness. Research Gate



In Figure 3.8, which exemplifies the concept of perimeter smoothness, the first shape represents the convex perimeter of a polygon fitted around the object, while the second one shows the true perimeter. The ratio of the lengths of those two perimeters defines the smoothness ratio.

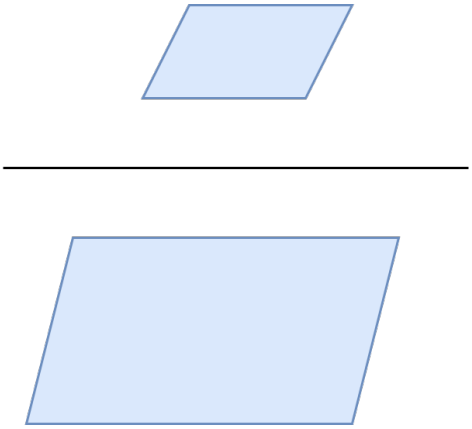
The number of connected components provides an indication of the mask's fragmentation. It tells how many separate parts make up the mask. Ideally, this number should be low, as we want to avoid small, noisy regions in the prediction. A few large components are easier for users to handle.

Finally, size retention measures how much of the original mask is preserved during post-processing. It compares the area of the post-processed mask to that of the original output mask. Ideally, this value should be close to 1, as we do not want to remove too much information during post-processing. It is computed as:

$$Size\ retention\ ratio = \frac{Area\ post\_processed\ mask}{Area\ mask}$$

In Figure 3.9, the polygon at the numerator is the one obtained after post-processing, while the one at the denominator is the one before post-processing.

Figure 3.9: Size retention ratio



3.3 Datasets

For my experiments, I used four datasets available on ULiège's Cytomine platform: LBTD-AGDC10, LBTD-NEO04, Camelyon16, and CHU-ANAPATH-NST-DL. The first three were also used in the internship. I reused and combined them for the zero-shot evaluation and the fine-tuning. The last one was selected to evaluate the generalization capabilities of the models. The number of annotations in each dataset is shown in Table 3.1:

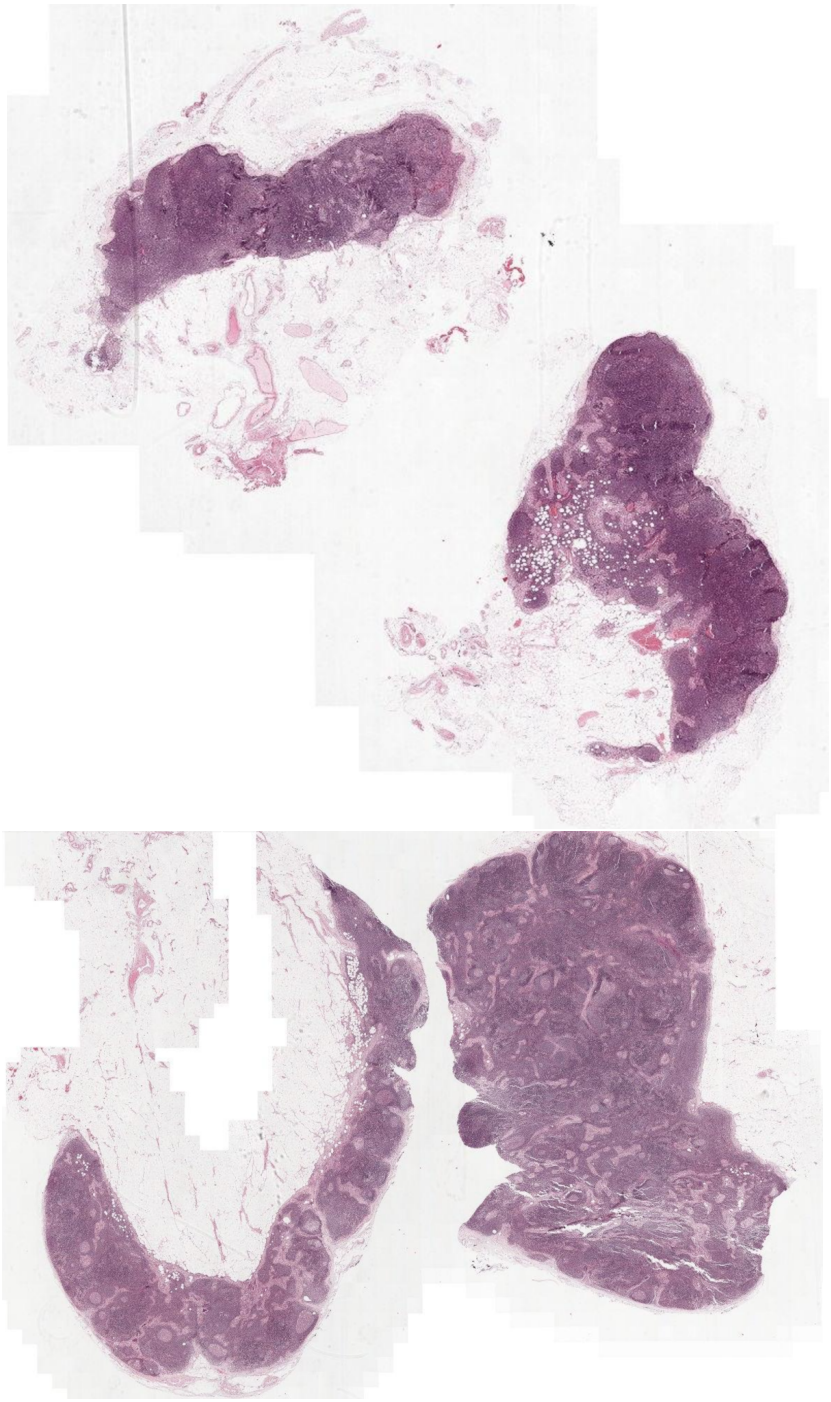
Table 3.1: Dataset compositions

Dataset	Training	Validation	Test	Total
LBTD-AGDC10	815	182	202	1199
LBTD-NEO04	1028	254	256	1538
Camelyon16	1679	477	441	2597
CHU-ANAPATH	0	0	13290	13290

These various datasets all contain WSIs; however these images correspond to different types of tissues. Indeed, histopathology focuses on the study of tissues, but these can vary significantly in both their type and their appearance. These differences are very important, since the various tissue types may look quite different from one another. Therefore, I will briefly present the content of each dataset and provide two examples of WSIs that can be found inside of them.

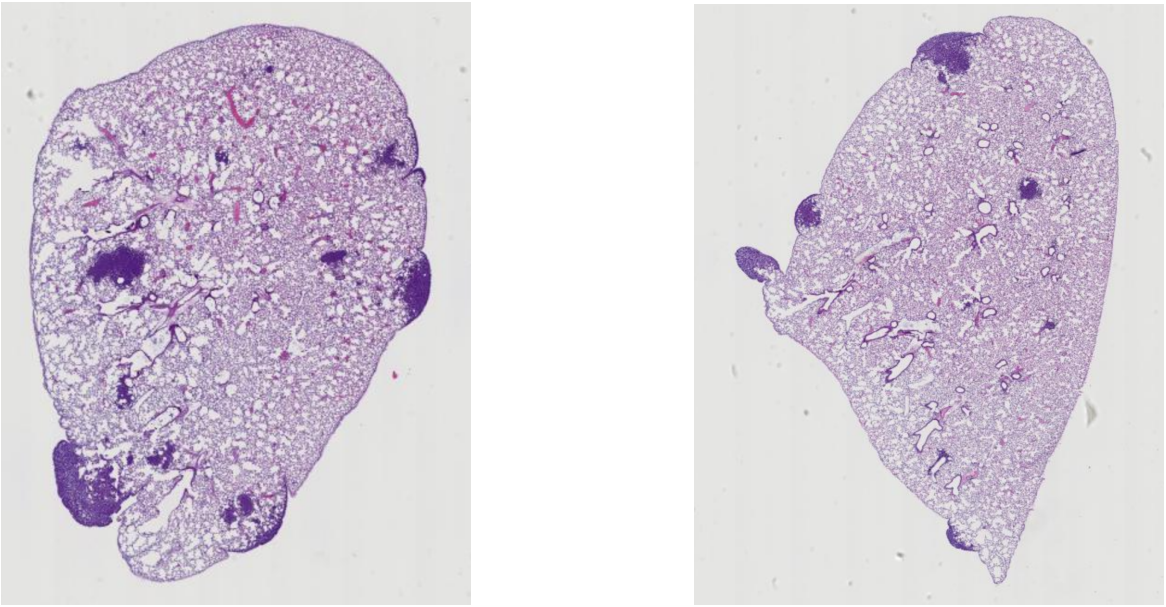
The Camelyon16 dataset is a publicly available dataset which contains images of both healthy and cancerous tissue. It was originally released for a segmentation challenge [Ehteshami Bejnordi et al., 2017] and is very popular in histopathology. This dataset focuses on WSIs of lymph nodes, more particularly sentinel lymph nodes obtained from breast cancer patients. Each slide is stained with hematoxylin and eosin and is also accompanied by some detailed pixel-level annotations of the metastatic regions. Figure 3.10 shows two examples from this dataset.

Figure 3.10: Examples from Camelyon16 dataset.



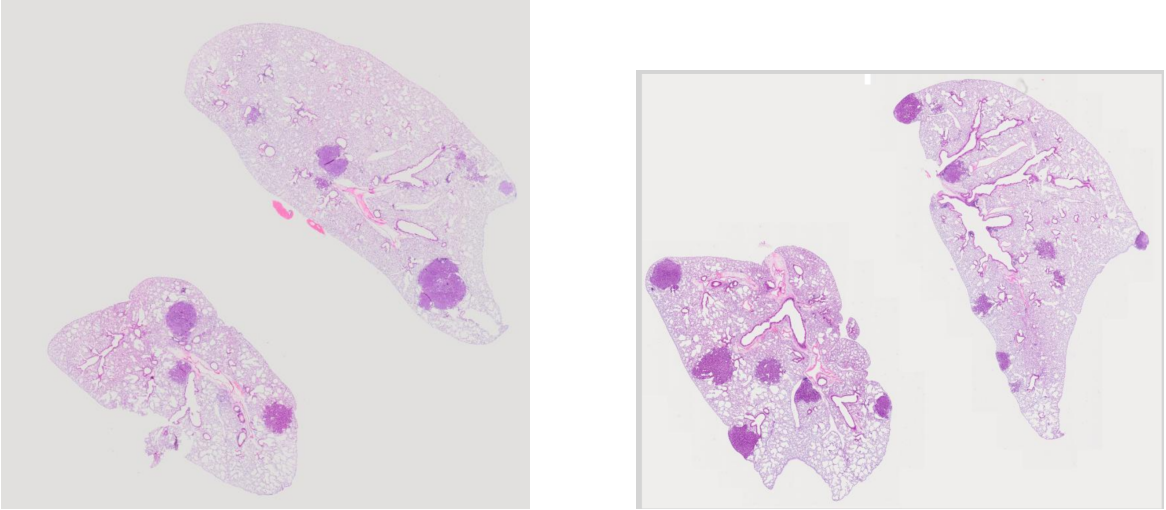
In contrast, the LBTD-AGDC10 dataset is a private dataset, which contains a collection of WSIs of lung tissue. It includes two main categories of annotations: lung adenocarcinoma and non-adenocarcinoma tissue. It thus focuses on distinguishing cancerous regions from non-cancerous regions within the lung tissue. Figure 3.11 shows two examples from this dataset.

Figure 3.11: Examples from LBTD-AGDC10 dataset.



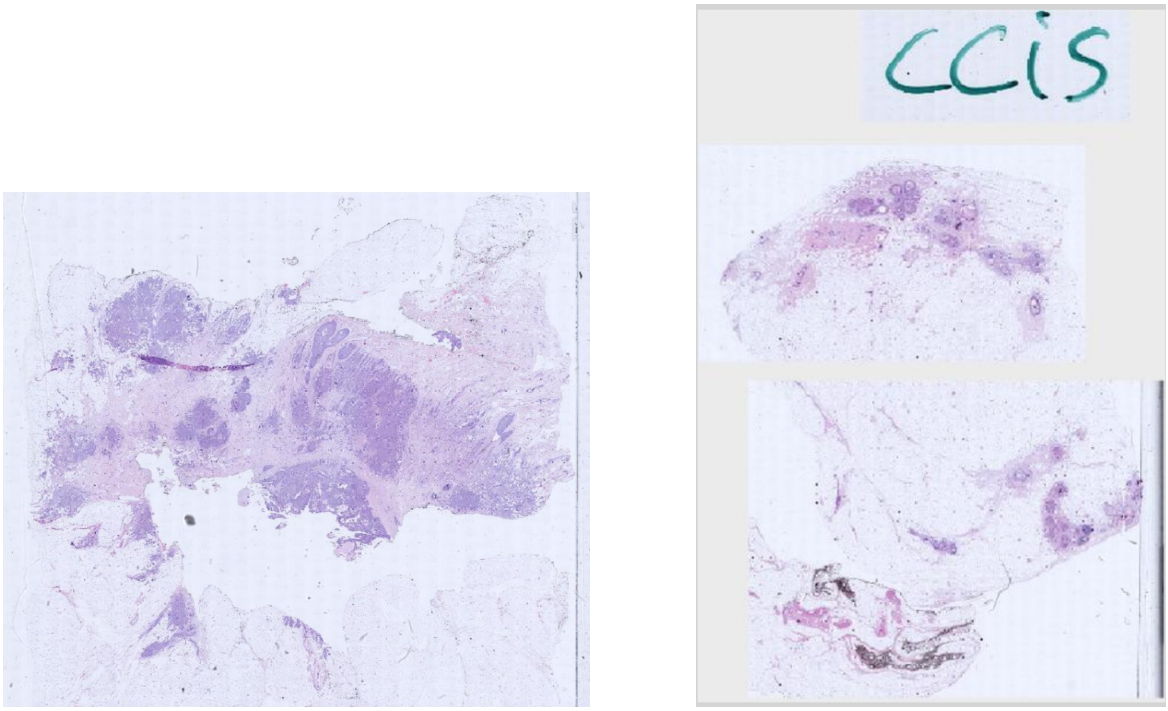
The LBTD-NEO04 dataset is very similar to the LBTD-AGDC10 one. It is also a private dataset that focuses on lung tissue, but it includes a wider range of annotated elements. These include bronchi, cartilage, inflammatory regions, muscle tissue, red blood cells, and more. Therefore, it provides more diverse annotations compared to LBTD-AGDC10. Two examples for this dataset are again displayed in Figure 3.12.

Figure 3.12: Examples from LBTD-NEO04 dataset.



Finally, the CHU-ANAPATH-NST-DL dataset is a private dataset originating from the CHU hospital. It contains annotations for a wide range of histopathological structures, including calcifications, healthy glands, in situ carcinoma, and infiltrated regions, etc... This dataset is particularly interesting because it includes tissue structures that were not present in the previous datasets. Therefore, it provides a valuable benchmark to evaluate how well the models trained on the other three datasets can generalize to new, unseen structures. Figure 3.13 shows two examples from CHU-ANAPATH-NST-DL.

Figure 3.13: Examples from CHU-ANAPATH-NST-DL.



Chapter 4

Zero-shot Capabilities and Initial Evaluation

4.1 Methodology

In this chapter, I analyze the zero-shot performance of both SAM and SAM2. Indeed, as mentioned in the Literature Overview section, there is currently no clear consensus in the scientific community regarding which one of the two models performs better in the medical field. Therefore, it is important to compare them across the various Cytomine datasets introduced in the previous chapter.

Regarding the structure of this chapter, I begin by describing the experiments that I conducted to define a new mask prompt generation process (Section 4.2). Indeed, the approach originally proposed during the internship might not be fully suitable for this use case. Therefore, a new approach might be interesting.

Then, I evaluate the performance of both models when prompted with different types of prompts, as well as various prompt combinations, to identify which configurations yield the best results (Section 4.3). I also analyze the results per dataset to determine whether the model performance varies depending on the dataset used (Section 4.4).

After that, I assess the performance of both models in their automatic mode (Section 4.5). This mode allows the models to generate their own prompts without any user input. I briefly explain how this mode works and show why it is not well-suited for the context of this thesis.

Finally, I investigate how performance varies depending on the image encoder that is used (Section 4.6). As described in the Background chapter, both SAM and SAM2 offer several encoder variants with increasing numbers of parameters. I analyze which encoder leads to the best performance, whether the differences between them are significant, and how these choices impact the inference time. Indeed, the inference time is an important consideration, given that the model is intended for interactive use in Cytomine.

4.2 Mask Prompt Generation Process

4.2.1 Motivation for a new Mask Prompt Generation Process

The mask prompt generation process introduced during the internship (called "scribbled masks") might not be suitable for use in Cytomine or to achieve good generalization performance. In his approach, Mr. Gille assumed that users would have access to a "paintbrush" tool, allowing them to highlight the annotations by drawing several horizontal lines.

First, this method may not be very practical for users, as tracing multiple lines for a single annotation could be time-consuming and quite inefficient.

More importantly, Cytomine currently does not offer any such paintbrush tool. Therefore, this type of mask cannot be created without modifying the front-end of the application. However, implementing and integrating such a tool could require significant work to ensure it functions smoothly and is user-friendly.

There are also significant concerns about the assumptions made in this method. Indeed, for example, it always uses the same number of horizontal lines. While this might not affect the zero-shot performance analysis, it could harm importantly the generalization during fine-tuning (this was not investigated during the internship). Moreover, the horizontal lines are always perfectly straight, which would likely not be the case in real usage, and we do not know how such idealized input prompts could affect the model performance.

Finally, a last concern is that, in the original process, all the horizontal lines are drawn entirely inside the ground truth mask and then dilated to fully cover the annotation. This setup assumes perfect alignment, which may not be the case in practical use. In reality, a user could easily misplace one or more lines.

For all these reasons, it is important to develop a new generation process that makes less assumptions about the mask structure. A more realistic approach would likely improve the model's generalization performance when it is used by actual users.

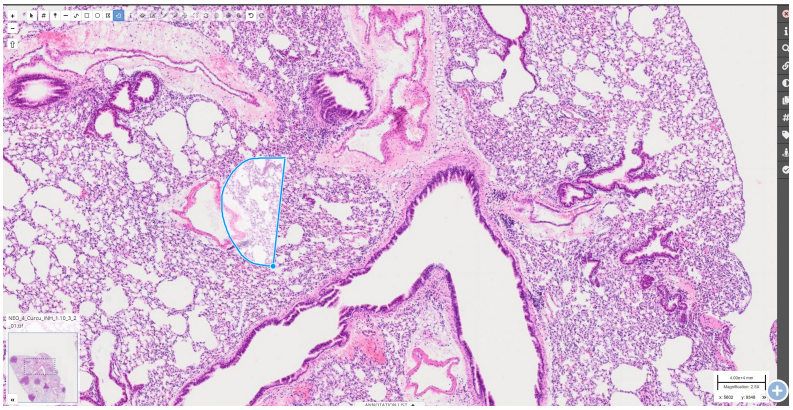
4.2.2 New Mask Prompt Generation Process

To find a new mask generation process, I started from the idea that users would prefer a fast and simple way to create masks. I also wanted this process to rely only on tools already available in Cytomine, or tools that would require minimal additional development.

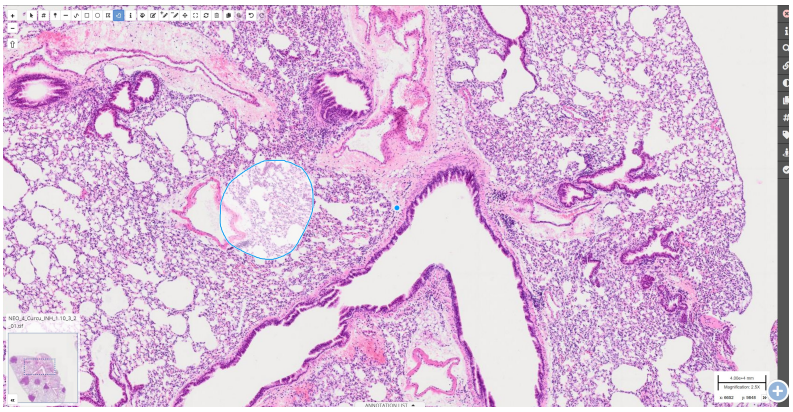
One easy and commonly used idea is the use of a "lasso"-type tool that allows users to circle the object of interest, creating a kind of irregular shape. This would be a quick interaction for users, requiring only to enclose the region of interest. Moreover, Cytomine already includes such a tool, called the freehand polygon, which can be used to draw this kind of mask.

This tool and an example of an annotation made with it are shown in Figure 4.1. The tool works very simply: the user clicks and holds while circling around the object.

Figure 4.1: Freehand polygon tool. Source: Cytomine Uliège



(a) Circling the object



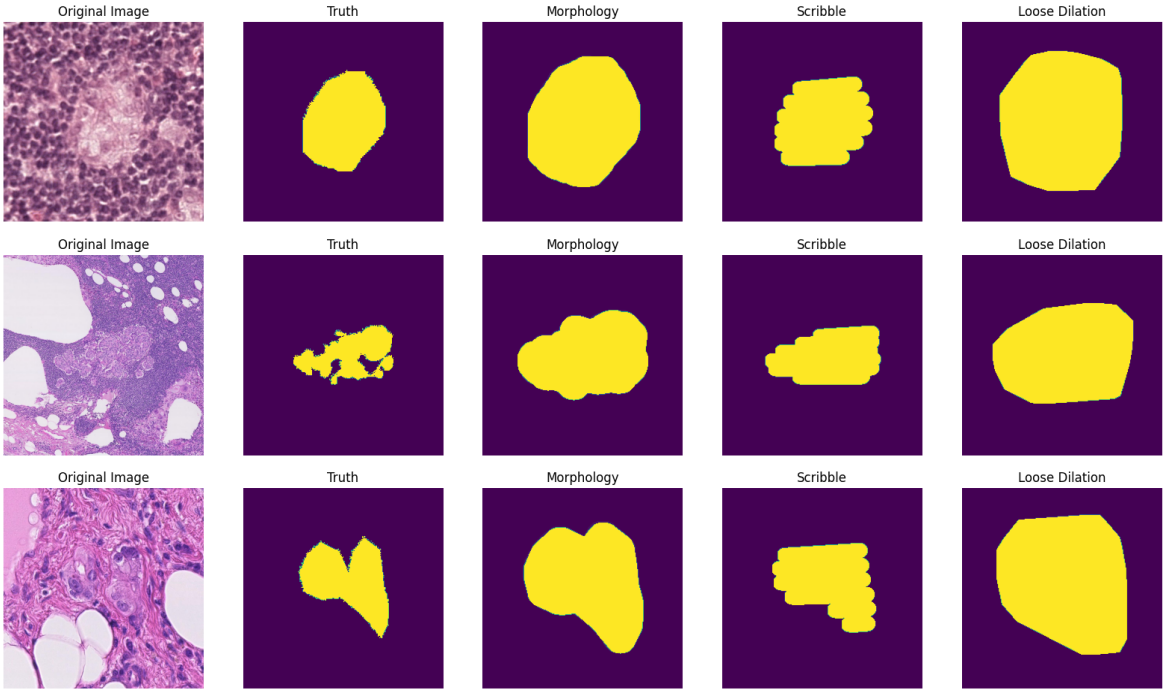
(b) Completed annotation

This idea seemed promising, as it does not require any changes to the Cytomine interface. The only remaining challenge was to artificially generate such masks to use them for evaluation and training, in order to avoid requiring the experts to revisit and modify all of their already existing annotations.

To do so, I started from the ground truth masks. I first extracted their contours and applied random noise to simulate the natural imprecision of human users. The noise level was selected randomly from the interval $[-20,\ 20]$. I then applied a convex hull transform to the modified contours. This operation returns the smallest convex shape that contains the original contour. It aligns well with the idea of circling an object: indeed, the goal is not to follow every cavity or hole in the object's shape, but to enclose the object in a single, rough outline (a "potatoid" shape). Finally, I applied a large dilation operation to the convex hull to make it even less tight around the object and to simplify the mask further. This final step creates smoother, looser shapes that look more like what a real user might draw with a lasso tool. In the end, this artificial mask generation process is simple and consistent with my initial philosophy: creating a mask that is fast and realistic to generate.

To better illustrate what the generated masks look like, Figure 4.2 shows an array of examples. The first column contains patches extracted from Cytomine. The second shows the corresponding ground truth masks. The third column, labeled Morphology, displays masks obtained by applying dilation directly to the ground truth. I included this (which is also used in the internship) to compare against a mask that remains relatively close to the original one but has been slightly altered. It helps to assess whether retaining more shape details improves the performance. The fourth column, labeled Scribble, shows the masks created with the internship's scribble method. And, the final column, Loose Dilation, presents the masks generated using the new process that I just described.

Figure 4.2: Different Mask types



From Figure 4.2, we can see that the masks produced by the new process achieve the intended goal: they approximate a loose, "potatoid" contour of the original object. Based on these results, I found the approach to be satisfactory and selected it for my other experiments.

4.2.3 Experiment on Mask Types

Table 4.1 presents the results of comparing the different mask types for both SAM and SAM2. The two models were tested on the three datasets LBTD-AGDC10, LBTD-NEO04, and Camelyon16 combined into a single evaluation set. Each model was tested four times using only mask prompts, with a different mask generation method for each run.

The first experiment involved providing the models with the ground truth masks. The second used the "morphology" masks. And, the last two experiments used the "scribbled" and "loose dilation" masks. This setup allows to evaluate the zero-shot performance of the two models across the different types of mask prompts.

The table reports several evaluation metrics. These include the four standard metrics used in the internship and that I described in the previous chapter: Dice score, IoU, precision, and recall. These are reported for each mask type. Moreover, the table also includes the values of these same metrics computed on the input mask itself, before being processed by the model. These are marked with the "input" label in the table. Indeed, this enables us to observe how much the performance drops between the original input mask and the predicted output mask.

For easier comparison, the best scores for each metric between the two models are highlighted in bold.

From Table 4.1, we can make several observations. First, we see that SAM2 clearly outperforms SAM. Indeed, for every mask type (except for the recall metric on "morphology" masks), SAM2 achieves significantly better results, often two to four times higher. This contrasts strongly with the results of [Sengupta et al., 2025], where SAM sometimes performed better. However, it is important to note that their experiments only involved point prompts, not mask prompts. This suggests that SAM2 might have brought significant improvements particularly for mask prompts.

Another key observation is that the overall performance remains quite poor. The best result is obtained with SAM2 using "morphology" masks, achieving a Dice score of 0.572 and an IoU of 0.430. However, an IoU score of 0.430 indicates that less than half of the total area covered by either the prediction or the ground truth is correctly segmented. The results for the other mask types are even worse. Moreover, "morphology" masks are not practical in real use, as they require the users to manually approximate the full outline of the object. Therefore, they represent an overly optimistic scenario compared to real use.

Focusing now on the two mask generation methods of interest ("scribbled" and "loose dilation") we see that, a bit surprisingly, the scribbled masks perform slightly better in most metrics. With the exception of recall, the scribbled masks have higher performance all across the board, often by several percentage points. On reflection, this result is not entirely unexpected. Indeed, the scribbled masks are tighter around the ground truth, which reduces the chance of the models predicting pixels that do not belong to the object, therefore, the precision increases (by over 15% for SAM2). On the other hand, they tend to result in more false negatives compared to the "loose dilation" masks, meaning that the models miss some parts of the mask. This is consistent with the behavior of "loose dilation" masks, since they provide a broader input and therefore help the models to capture more of the mask area. Despite this slight advantage in accuracy, I will still focus on the "loose dilation" masks in the following sections because of the limitations of the scribbled approach that I already discussed.

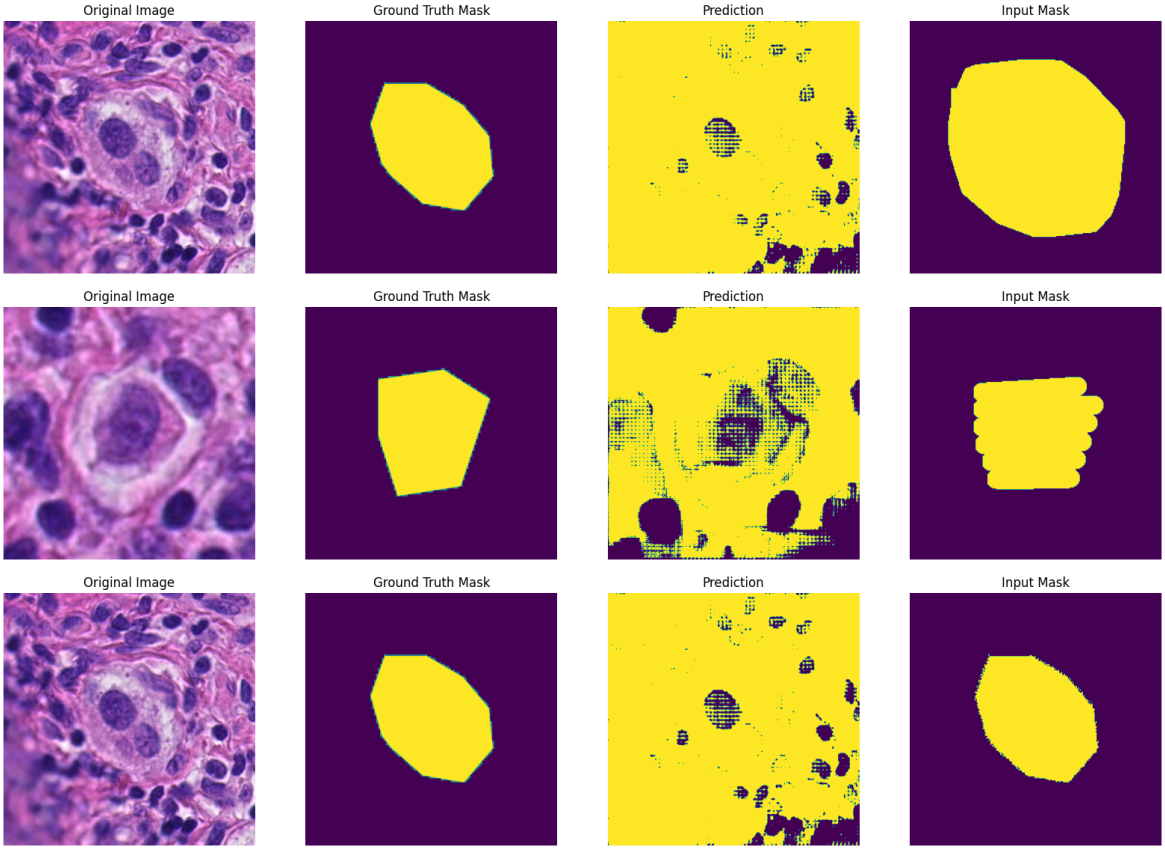
Finally, still in Table 4.1, if we compare the input masks to the masks predicted by the models, we observe a consistent performance drop. This is particularly noticeable when the ground truth masks are used as input, which is to be expected, as the input mask scores are perfect for these. With all the other mask types, the predicted masks perform worse in every metric except the precision. This makes sense since the input masks generated by the morphology, scribbled, and loose dilation methods typically cover more area than the ground truth. Therefore, as the models often predict smaller or more refined masks, this improves the precision.

Table 4.1: Comparison of the different Mask Types

Prompts	Metric	SAM	SAM2
Ground Truth	Dice	0.107	**0.475**
	IoU	0.063	**0.360**
	Precision	0.168	**0.725**
	Recall	0.290	**0.514**
	Dice input	1.0	1.0
	IoU input	1.0	1.0
	Precision input	1.0	1.0
	Recall input	1.0	1.0
Morphology	Dice	0.256	**0.572**
	IoU	0.154	**0.430**
	Precision	0.158	**0.571**
	Recall	**0.894**	0.732
	Dice input	0.599	0.599
	IoU input	0.435	0.435
	Precision input	0.435	0.435
	Recall input	0.999	0.999
Scribble	Dice	0.111	**0.460**
	IoU	0.066	**0.339**
	Precision	0.167	**0.659**
	Recall	0.301	**0.523**
	Dice input	0.815	0.815
	IoU input	0.696	0.696
	Precision input	0.725	0.725
	Recall input	0.948	0.948
Loose Dilation	Dice	0.121	**0.431**
	IoU	0.072	**0.303**
	Precision	0.151	**0.496**
	Recall	0.331	**0.604**
	Dice input	0.494	0.494
	IoU input	0.334	0.334
	Precision input	0.335	0.335
	Recall input	0.999	0.999

In fact, I was quite surprised by how low the performance was, especially compared to the results presented in the original Segment Anything papers. To better understand the issue, I inspected several predicted masks. These confirmed the disappointing performance, because the masks were indeed often far from accurate. Figure 4.3 shows some examples of poor predictions (from SAM2), with one example per mask type.

Figure 4.3: Poor Mask Segmentation Examples (SAM2)



Looking at Figure 4.3, we can clearly observe that the model significantly over-segments the objects. Nearly the entire patch are segmented in those examples, rather than just the target object. This result was unexpected, but it may be explained by two key factors. First, both models were originally trained on natural images, which differ greatly from histopathological images. Second, the input masks used during training likely followed a very different distribution from the ones used in my experiments. These differences could help explain the poor performance observed in this setting.

4.3 Prompting Performance Evaluation

In this section, I assess the performance of the two Segment Anything models across different input prompt types and some of their combinations. As in the previous section (Section 4.2), the models were evaluated on the three datasets: LBTD-AGDC10, LBTD-NEO04, and Camelyon16 combined into a single evaluation set. For each prompt or prompt combination, I report the same four metrics used earlier: Dice score, IoU, precision, and recall.

The prompts that I tested include point prompts, box prompts, a combination of boxes with positive and negative points, and a combination of masks, boxes, and points. I did not test negative point prompts on their own. Indeed, based on findings from both the internship and the literature, negative points alone produced poor results. Since there are already many possible prompt combinations, I chose to save time by avoiding non-informative or unrealistic configurations. Moreover, testing negative prompts alone does not make much sense from a user perspective: it is more natural for a user to indicate where an object is present, rather than where it is not (it also requires less prompts).

For point prompts, I tested two configurations: one with a single point and one with five point prompts. This choice was based on the internship setup by Mr. Noé Gille and allows for consistency in comparisons. It also helps to evaluate whether providing more prompts improves the performance. Since the number of prompts remains relatively low, it could still have a significant impact, especially considering that previous studies suggested performance plateaus only after a larger number of prompts (they tested up to 20 prompts).

Additionally, for negative point prompts, I evaluated two variations: one where the negative points were selected within the bounding box, and another where they were placed anywhere outside the mask. This was done to assess whether the location of the negative prompts has an influence on the models performance.

The results of these experiments are displayed in Tables 4.2 and 4.3. Table 4.2 contains the results related to box and point prompts, while Table 4.3 includes those involving masks. In both tables, the "+" symbol represents positive point prompts and the "−" symbol represents negative ones. The label "inside" refers to negative points placed within the bounding box, while "anywhere" means they were selected from any location outside the mask. The number of points used in each experiment is also indicated, directly in Table 4.2, and in parentheses in Table 4.3.

From Table 4.2, several observations can be made. First, the performance varies significantly depending on the type of prompt used. In particular, the point prompts perform noticeably worse than the bounding boxes, regardless of the number of points. The difference in Dice score between the two types ranges from 0.18 to nearly 0.5. This clearly shows that point prompts are not ideal and confirms the findings in the literature, they are actually the least effective prompts in this experiment. We can also observe that increasing the number of points from one to five slightly improves

performance across all metrics. In this case, using more prompts does help, but the gain remains small compared to using a more informative prompt type, such as bounding boxes. Regarding the difference between using box prompts alone versus combining them with points, the variation is also relatively small, suggesting that the presence of the bounding box itself is what contributes the most to the models performance.

Another key observation is that SAM2 consistently outperforms SAM, which aligns with the results of the previous section (Section 4.2). This confirms that SAM2 appears to be the stronger model in terms of raw segmentation accuracy.

As mentioned earlier and supported by the literature, bounding boxes are the most effective prompts. They achieve a Dice score of 0.831 and an IoU of 0.728, indicating strong performance. Even more promising, however, is the combination of a bounding box with a single positive point prompt, which yields the best overall results: a Dice score of 0.834 and an IoU of 0.731. This thus makes it the most effective configuration in this experiment, even though the improvement is small compared to using bounding boxes alone. This outcome is intuitive: while the bounding box defines the region of interest, the additional point prompt helps the models to focus on the specific object to segment within that region. We also see that increasing the number of points when used in combination with a bounding box leads to a slight drop in performance. Indeed, this was also noted in the literature, and here we can clearly observe that adding more points, whether positive or negative, tends to reduce slightly the overall accuracy. For example, adding more positive points decreases performance slightly (-0.01 Dice), and the drop is even more pronounced with negative points (up to -0.04 Dice).

Finally, the combination of bounding boxes with negative point prompts performs slightly worse than the combination with positive point prompts. Positive prompts are therefore the better choice here. Moreover, using negative points inside the bounding box gives better results than placing them anywhere else. This is logical: placing negative prompts closer to the object helps to define its boundaries more precisely, whereas distant prompts carry less useful information.

In the end, the results of this experiment are both logical and consistent with previous research. The best-performing setting is the combination of a bounding box with a single positive point prompt. Overall, the performance achieved here is significantly better than what was observed in the mask-prompt experiments presented in the previous section (Section 4.2).

In Table 4.3, I present the results of the experiments conducted using two mask types: the scribbled masks proposed by Mr. Gille and the "loose dilation" masks that I proposed. The goal was to compare the effectiveness of the two approaches when combined with additional prompts. In the previous section (Section 4.2), we observed that scribbled masks slightly outperformed loose dilation masks when used alone; here, I wanted to see whether this advantage remains when other prompt types are added. For both SAM and SAM2, the table includes two separate columns, one for each mask type, allowing to directly compare across the different prompt combinations.

Table 4.2: Prompting Evaluation

Prompt types	Metric	SAM	SAM2
1 point (+)	Dice	0.300	**0.495**
	IoU	0.208	**0.393**
	Precision	0.473	**0.725**
	Recall	**0.726**	0.649
5 points (+)	Dice	0.399	**0.651**
	IoU	0.289	**0.531**
	Precision	0.419	**0.766**
	Recall	**0.857**	0.740
Box	Dice	0.798	**0.831**
	IoU	0.685	**0.728**
	Precision	0.781	**0.816**
	Recall	0.860	**0.878**
Box + 1 point (+)	Dice	0.793	**0.834**
	IoU	0.674	**0.731**
	Precision	0.763	**0.812**
	Recall	0.881	**0.888**
Box + 5 points (+)	Dice	0.776	**0.823**
	IoU	0.652	**0.715**
	Precision	0.750	**0.779**
	Recall	0.870	**0.905**
Box + 1 point (-) inside	Dice	0.787	**0.827**
	IoU	0.669	**0.722**
	Precision	0.760	**0.811**
	Recall	0.863	**0.873**
Box + 1 point (-) anywhere	Dice	0.767	**0.824**
	IoU	0.645	**0.719**
	Precision	0.730	**0.809**
	Recall	0.863	**0.872**
Box + 5 points (-) inside	Dice	0.741	**0.803**
	IoU	0.611	**0.689**
	Precision	0.689	**0.782**
	Recall	0.853	**0.857**
Box + 5 points (-) anywhere	Dice	0.639	**0.779**
	IoU	0.495	**0.661**
	Precision	0.549	**0.742**
	Recall	0.844	**0.863**

As in the previous experiment, I did not test all possible prompt combinations in order to save time. Specifically, I did not distinguish between negative point prompts placed inside the bounding box versus those placed anywhere. Based on earlier results, I directly used negative point prompts inside the bounding box (specifically, the input mask's bounding box this time), as they previously showed better performance. Similarly, for combinations involving masks, boxes, and points, I only included configurations with a

single point prompt, since the results from Table 4.2 showed that increasing the number of points tended to reduce performance when used with bounding boxes.

Looking at the results in Table 4.3, we again observe a high degree of variability depending on the prompt combination. Combinations that rely solely on masks or masks with points generally perform worse than those that include bounding boxes, once again showing the effectiveness of box prompts. For example, using a scribbled mask with five positive point prompts yields a Dice score of 0.689 and an IoU of 0.571, which are both lower than the ones of the combination of a mask with a bounding box, which achieves a Dice score of 0.717 and an IoU of 0.583. This performance gap is slightly more important for SAM compared to SAM2.

Except for the recall, SAM2 continues to outperform SAM, in line with the observations from the previous experiments. The performance gap between the models varies depending on the prompt type, with some differences being small while others are more significant. Notably, some prompt combinations result in extremely low performance. As expected from earlier results, the mask-only prompts perform the worst. However, the combinations of masks and point prompts also give relatively poor results.

On the other hand, adding a bounding box prompt with a mask, regardless of whether points are included, leads to a considerable improvement. For SAM2, this configuration boosts the Dice score by 0.257 and similarly improves the IoU. A similar pattern is seen for SAM. These results confirm once again that bounding boxes provide valuable information to the models. However, from a user perspective, this combination is less practical. While providing a bounding box is relatively quick, creating a mask in addition to that requires more effort. Although one might consider extracting a bounding box from a mask automatically, the performance of "mask plus box" combinations is still lower than using a box alone, as shown in the previous table.

The best-performing configuration in this table is the one with a mask, a bounding box, and a single positive point prompt. This setup achieves a Dice score of 0.754 and an IoU of 0.628. Nevertheless, these values are still lower than the best results from the previous experiment. This suggests that adding a mask to box and point prompts does not improve, and may even reduce the overall accuracy. However, this effect is not uniform. For example, adding a mask to point prompts generally improves results, but adding a mask to box based combinations seems to reduce the performance. The only clear benefit of adding a mask is an increase in the recall value: indeed, the models seem to miss fewer true pixels. However, this reduces the precision, indicating a higher rate of false positives.

Finally, we observe that scribbled masks consistently outperform loose dilation masks across all configurations. This is perfectly consistent with the previous findings, but due to the limitations discussed earlier, scribbled masks are not a viable option in practice, as they cannot currently be created in Cytomine.

Overall, the results from this experiment are weaker than those from the previous one. This suggests that using masks as prompts is not particularly beneficial for two reasons. First, they do not deliver the best accuracy. Second, they are less practical for the user, as drawing a mask is more time-consuming than providing a bounding box.

Table 4.3: Prompting Evaluation (for Masks)

Prompt types				**SAM**		**SAM2**	
Box	Points (+)	Points (-)	Metric	Loose	Scribble	Loose	Scribble
-	-	-	Dice	**0.121**	0.111	0.431	**0.460**
			IoU	**0.072**	0.066	0.303	**0.339**
			Precision	0.151	**0.167**	0.496	**0.659**
			Recall	**0.331**	0.301	**0.604**	0.523
-	Yes (1)	-	Dice	0.242	**0.245**	0.512	**0.516**
			IoU	0.149	**0.152**	0.409	**0.414**
			Precision	0.204	**0.227**	0.816	**0.839**
			Recall	**0.941**	0.922	**0.568**	0.553
-	-	Yes (1)	Dice	0.110	**0.140**	0.429	**0.440**
			IoU	0.072	**0.097**	0.328	**0.342**
			Precision	0.150	**0.227**	0.505	**0.566**
			Recall	0.174	**0.199**	**0.594**	0.564
-	Yes (5)	-	Dice	0.214	**0.224**	0.627	**0.689**
			IoU	0.123	**0.133**	0.512	**0.571**
			Precision	0.124	**0.134**	0.636	**0.767**
			Recall	**0.998**	0.997	**0.852**	0.781
-	-	Yes (5)	Dice	0.163	**0.277**	0.484	**0.550**
			IoU	0.114	**0.204**	0.379	**0.448**
			Precision	0.219	**0.376**	0.584	**0.717**
			Recall	0.233	**0.357**	0.527	**0.531**
Yes	-	-	Dice	0.351	**0.696**	0.451	**0.717**
			IoU	0.221	**0.553**	0.302	**0.583**
			Precision	0.226	**0.590**	0.305	**0.606**
			Recall	0.922	**0.925**	**0.976**	0.946
Yes	Yes (1)	-	Dice	0.314	**0.605**	0.503	**0.754**
			IoU	0.190	**0.457**	0.349	**0.628**
			Precision	0.191	**0.485**	0.353	**0.656**
			Recall	**0.993**	0.962	**0.986**	0.946
Yes	-	Yes (1)	Dice	0.397	**0.733**	0.474	**0.710**
			IoU	0.259	**0.600**	0.326	**0.573**
			Precision	0.272	**0.651**	0.334	**0.595**
			Recall	0.889	**0.904**	0.92	**0.946**
Yes	Yes (1)	Yes (1)	Dice	0.359	**0.657**	0.497	**0.731**
			IoU	0.226	**0.513**	0.343	**0.598**
			Precision	0.229	**0.543**	0.347	**0.618**
			Recall	**0.983**	0.954	**0.981**	0.956

4.4 Performance across Datasets

In the previous section (Section 4.3), I evaluated the performance of various prompt settings to identify which combinations/prompt types gave the best results. These experiments were conducted on a combined evaluation set that included three datasets: LBTD-AGDC10, LBTD-NEO04, and Camelyon16. One relevant question one could ask is whether the model performance varies significantly across individual datasets. In other words, could the models perform very well on certain datasets while underperforming on other datasets ? To check this, I decided to evaluate both models separately on each dataset. For this experiment, I chose to use the box-only prompt, which was the second-best performing setting overall, just behind the combination of box and one point prompt. I selected this configuration because it offers a more consistent basis for comparison across datasets in my opinion. Indeed, when combining a box with a point prompt, two sources of variability are introduced: the random placement of the point within the mask and the shifting noise applied to the box coordinates. If the performance scores between datasets were very close, this added variability could make it harder to detect meaningful differences. Using only the box prompt introduces less variation, making it a more stable and reliable option for comparing the results across datasets.

Table 4.4: Prompting Evaluation across Datasets

Datasets	Metric	SAM	SAM2
LBTD-AGDC10	Dice	0.784	**0.796**
	IoU	0.670	**0.685**
	Precision	0.738	**0.759**
	Recall	**0.882**	0.878
LBTD-NEO04	Dice	0.805	**0.860**
	IoU	0.698	**0.771**
	Precision	0.831	**0.855**
	Recall	0.824	**0.890**
Camelyon16	Dice	0.819	**0.838**
	IoU	0.709	**0.734**
	Precision	0.792	**0.826**
	Recall	**0.881**	0.880

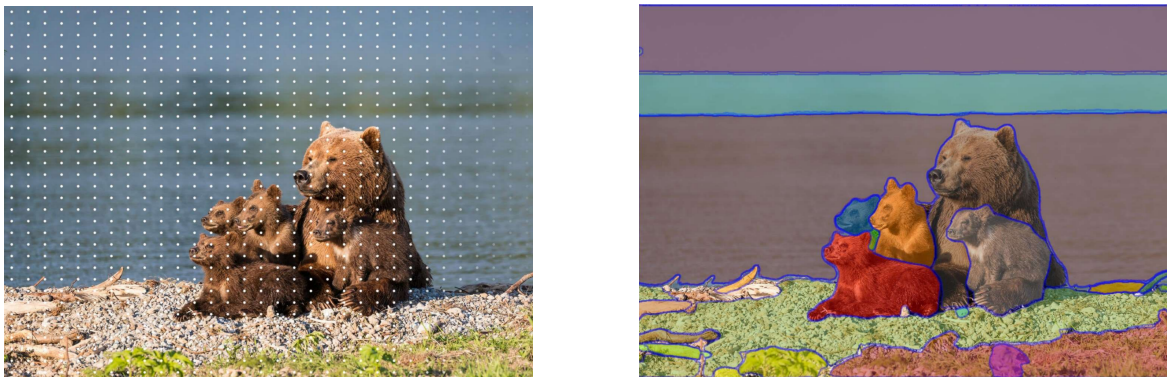
The results of this experiment are presented in Table 4.4. From the table, we observe that the performance across datasets is generally consistent. Of course, the models perform slightly better on some datasets than others: SAM2 achieves its best performance on LBTD-NEO04, while SAM performs best on Camelyon16. However, the differences are relatively small, for instance, the largest gap in Dice score is only 0.064. This suggests that the models generalize reasonably well across all datasets and do not suffer from significant performance drops on specific ones. It is also interesting to note that the dataset on which each model performs best is not the same.

4.5 Automatic Mode Evaluation

In this section, I analyze the performance of the automatic mode of both SAM and SAM2, which was originally proposed in [Kirillov et al., 2023], as a complementary mode of execution for the model in their pipeline. The term "automatic" here does not mean that no prompts are used, indeed both models require prompts to function. Instead, it refers to the automated generation of prompts.

This automatic mode is based on a simple idea which is to generate a grid of point prompts. Indeed, given a user-defined number of points (along one dimension of the image), the predictor creates a grid. Each of these prompts produces segmentation masks, which results in a large number of masks per image. To reduce the redundancy, non-maximal suppression is then applied to remove the overlapping or duplicate masks. Moreover, only the masks considered to be confident and stable by the model are retained. This enables to reduce the total number of masks but still leaves a large number per image. Figure 4.4 shows two images illustrating this process. The first image corresponds to the grid of input points used for segmentation, and the second displays the resulting masks. One important limitation of this approach is that the models have no information about which object is the target. As a result, many of the generated masks may not correspond to any meaningful or relevant element. To handle this issue, I applied a filtering step by selecting only the mask with the highest predicted IoU score, according to the models' internal confidence estimation. Although this might not always select the mask that best matches the ground truth object, it shows the one the models are most confident in.

Figure 4.4: Automatic Mode Segmentation. Source: GitHub



To evaluate the automatic mode, I used the same three datasets as in previous experiments: LBTD-AGDC10, LBTD-NEO04, and Camelyon16, still combined into a single evaluation set. I applied the automatic mode to each patch extracted from the original WSIs and selected the mask with the highest predicted IoU, which was then compared to the ground truth using the same evaluation metrics as before.

The results are summarized in Table 4.5.

Table 4.5: Automatic Mode Evaluation

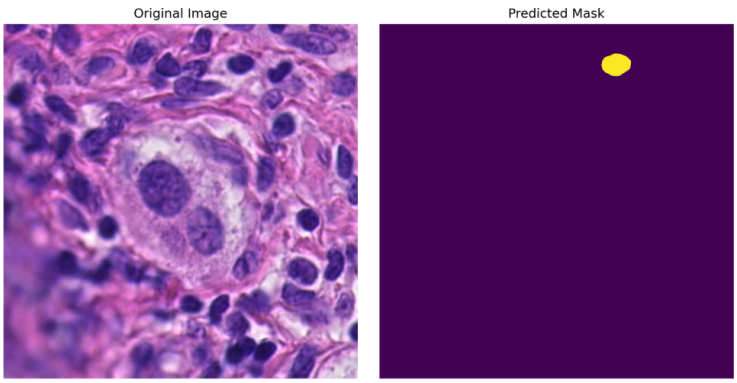
	Metric	SAM	SAM2
Automatic Mode	Dice	**0.053**	0.011
	IoU	**0.037**	0.006
	Precision	**0.095**	0.006
	Recall	**0.129**	0.061

The performance is extremely poor, indeed, these are the worst results observed so far. With the exception of recall, all metrics are below 0.1, indicating that the models completely fail to segment the target object accurately. This underperformance is likely due to the selection strategy: the mask with the highest predicted IoU may not correspond to the target object. To confirm this hypothesis, I visually inspected the segmentation outputs.

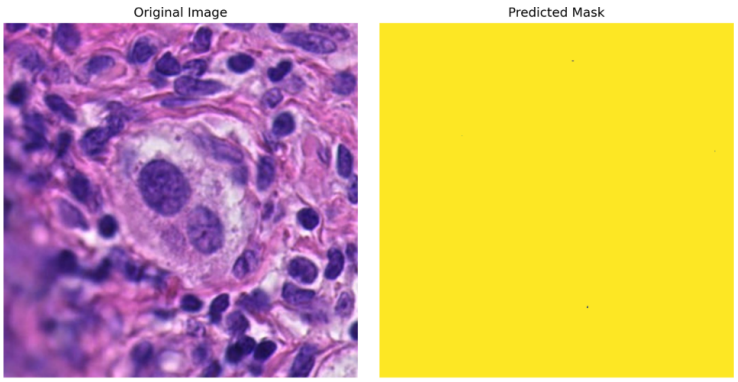
Figure 4.5 presents two examples from SAM and SAM2, where the target object was the central cell. In the case of SAM, the selected mask clearly does not correspond to the target object, this results in zero Dice and IoU scores for that example. For SAM2, the issue is totally different: the model over-segments and predicts a mask that covers the entire image patch. Thus, this reduces all the evaluation metrics.

Figure 4.5: Poor Mask Segmentation from Automatic Mode

(a) Poor Mask Segmentation from Automatic Mode (SAM)

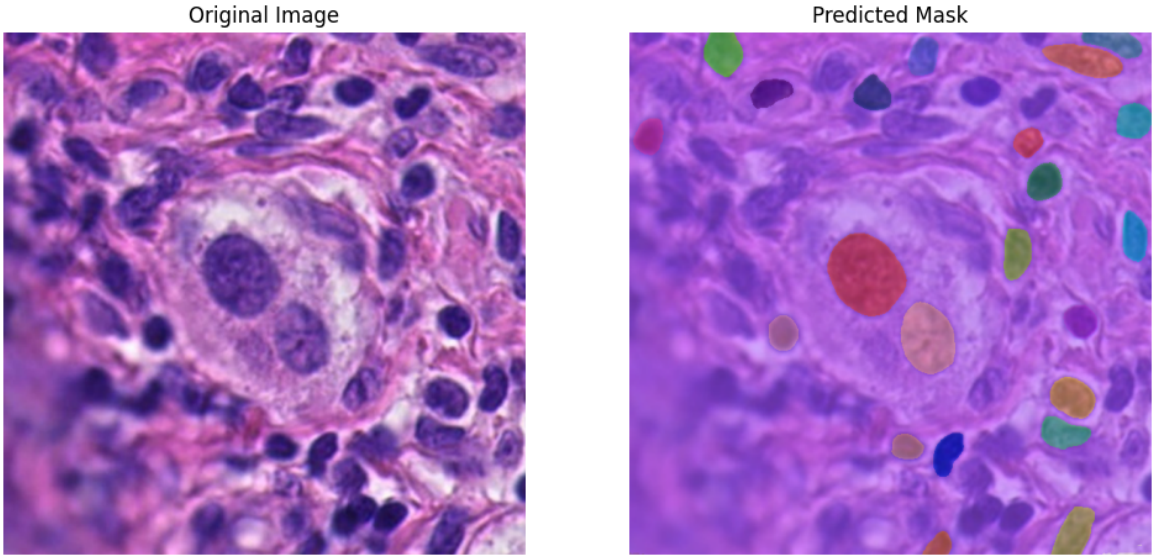


(b) Poor Mask Segmentation from Automatic Mode (SAM2)



While the behavior observed in SAM was not surprising, since the selected masks are unlikely to always match the actual target, the result for SAM2 was more confusing. To investigate it further, I examined all the masks generated by SAM2's automatic mode before the selection step. Figure 4.6 shows the full set of masks produced by the model. We can clearly observe that the model indeed treats the entire patch as a single large object, and this is the mask in which it is the most confident. We also see that although a few small masks near the center align partially with the actual cell, none correspond totally to the target. Additional examples are provided in Appendix A, and they confirm the same issues: incorrect mask selection, and frequent over-segmentation.

Figure 4.6: Poor Mask Segmentation from Automatic Mode (SAM2): Complete Segmentation



In conclusion, the automatic segmentation mode of both SAM and SAM2 yields unsatisfactory results on the histopathology data and should not be used in Cytomine.

4.6 Image Encoder Analysis

In this section, I detail the experiments conducted on the different image encoders available for SAM and SAM2. As mentioned in the Background chapter, both models have been trained with multiple versions of their image encoders, each with an increasing number of parameters. SAM includes three encoder variants: ViT-B (89M parameters), ViT-L (308M), and ViT-H (637M), while SAM2 offers four encoders: Hiera-Tiny (38.9M), Hiera-Small (46M), Hiera-Base-plus (80.8M), and Hiera-Large (224.4M).

In the original papers [Kirillov et al., 2023] and [Ravi et al., 2024], the authors observed that using larger encoders led to slightly better performance. However, these results were obtained on natural images, which the models were trained on. Therefore, it is interesting to verify whether this trend still holds for our histopathology datasets. Moreover, since the final goal is to integrate the model into Cytomine, the processing

speed must also be taken into account to ensure that the interactive segmentation is responsive. I thus need to find a trade-off between the segmentation quality, the inference speed, and the available computational resources (indeed, the largest encoders could not be fine-tuned with the hardware available on the Alan cluster from Montefiore [Montefiore, 2018] due to insufficient VRAM).

As in the "Performance across Datasets" section, I only used box prompts for this experiment. These prompts were kept consistent across encoders within each model but were not shared between the models. I used this approach for practicality, especially given that previous sections had already shown that SAM2 outperforms SAM, therefore, ensuring consistency between the two models was not important to compare the performance. The results are presented in Tables 4.6 and 4.7.

Table 4.6: Encoders Experiment for SAM

Metric	**vit_b**	**vit_l**	**vit_h**
Dice	0.832	0.827	**0.833**
IoU	0.734	0.730	**0.737**
Precision	0.819	0.847	**0.849**
Recall	**0.881**	0.841	0.847
Time (s)	**0.228**	0.553	0.996

Table 4.7: Encoders Experiment for SAM2

Metric	**hiera_t**	**hiera_s**	**hiera_b**	**hiera_l**
Dice	0.849	0.847	0.847	**0.850**
IoU	0.755	0.755	0.753	**0.758**
Precision	0.864	0.872	0.832	**0.881**
Recall	0.863	0.848	**0.889**	0.846
Time (s)	**0.061**	0.071	0.124	0.294

From these tables, several observations can be made. First, SAM2 is significantly faster at inference than SAM. In fact, the corresponding encoders of SAM2 are nearly four times faster than those of SAM, which makes this model well-suited for interactive segmentation. Therefore, combined with its better overall performance, SAM2 appears to be the more promising model for deployment within Cytomine.

Second, we can note that the differences in performance between encoder variants are quite small. For SAM, the Dice score varies by only 0.006 across encoders, and for SAM2, by just 0.003. Similarly, we observe low variation for the IoU score. This suggests that increasing the encoder size does not significantly improve the performance. That being said, the largest encoders, ViT-H for SAM and Hiera-Large for SAM2, did achieve the highest scores, although by a very small margin. However, some smaller encoders occasionally performed better than larger ones; for instance, ViT-B slightly outperformed ViT-L in one run, and a similar result was observed for Hiera-Tiny. Nevertheless, these inconsistencies remain very minor and could be due to random variation.

To sum up, the different encoder variants had only a limited impact on the models performance. As a result, in the next chapter dedicated to fine-tuning, I chose to use ViT-B and Hiera-Base-plus for SAM and SAM2, respectively. Indeed, these encoders offered a good balance between training time and inference speed, and they were compatible with the available hardware while still providing competitive results.

Chapter 5

Model Adaptation through Fine-tuning

5.1 Methodology

In this chapter, I analyze the performance of SAM and SAM2 when fine-tuned on histopathology data from Cytomine. While the previous chapter focused on zero-shot performance, here I assess how well the models perform after being trained on this new image domain through supervised training.

Since SAM2 previously showed better zero-shot performance, it is interesting to verify whether this advantage holds after fine-tuning, or if SAM performs better in this new setting.

This chapter begins with an experiment on the impact of loss functions on the fine-tuning of both models (Section 5.2). Indeed, during his internship, Mr. Noé Gille simplified the original loss function of SAM, and similar approaches are quite common in the literature. Therefore, I wanted to evaluate whether using simpler loss functions can improve the performance compared to the original ones.

The next two experiments focus on two other aspects. The first one analyzes how the choice of the training datasets affects the fine-tuning results (Section 5.3). In fact, as described in the "Project Context and Dataset Description" chapter, Mr. Gille conducted experiments using different combinations of the three datasets, but he did not exhaust all configurations. Therefore, I extended this analysis with a more complete evaluation. The second experiment focuses on the role of prompt types during the training (Section 5.4). For example, since bounding box prompts showed the best performance in the results of the previous sections, one key question could be: Should I fine-tune the model using only box prompts, or should all types of prompts be used during training ? Mr. Gille performed his experiments using all prompt types, so I investigated whether more targeted training improves performance on specific prompt types.

After that, I also experimented with fine-tuning the entire models, meaning that I included the image encoder this time (Section 5.5). In all the previous experiments, I had only fine-tuned the prompt encoder and the mask decoder. This choice was partially done to stay consistent with the internship work by Mr. Noé Gille, who did not fine-tune the encoder either. There were also more practical reasons behind that choice. Indeed, the image encoder is the largest part of the model, and fine-tuning it requires a lot of computing power (above all if you want to maintain a large batch size, however Meta advises to use at least one A100 GPU). For example, the paper [Zhu et al., 2024] mentions that they used 64 A100 GPUs with 80 GB of memory each, which is clearly not something I had access to. Moreover, since I had many different models to train for the previous experiments, keeping the training limited to the lighter parts of the model made much more sense in terms of time and resources. On top of that, I did not have access to large amounts of data, and both SAM and SAM2 have a huge number of parameters. So fine-tuning the full model could easily lead to overfitting or forgetting. This risk has already been mentioned in papers like [L. Zhang et al., 2024] and [Gu et al., 2024], which point out that fully fine-tuning SAM on small datasets often leads to worse results. Thus, this experiment allowed me to test that claim myself. In the literature, researchers generally used one of two main ways in order to deal with this problem. The first is to perform parameter-efficient fine-tuning, as done in [Gu et al., 2024] and [Teuber et al., 2025], and the second is to use domain-specific encoders to incorporate directly some domain-knowledge, like in [J. Zhang et al., 2023] or [M. Zhang et al., 2024]. In my work, I did not look into parameter-efficient fine-tuning, but in the next chapter I will focus on the second option, testing domain-specific modules for histopathology.

The experiments above helped me to identify the best-performing configurations. However, since the models will be used for interactive segmentation within Cytomine, a high accuracy alone is not sufficient. Indeed, the masks must also be usable (clean, editable, and user-friendly). To address this, I explored several post-processing techniques to improve the mask quality regarding that aspect, which I describe in the corresponding section (Section 5.7).

Finally, I examined how the models respond to a shift in the distribution of mask prompts (Section 5.8). As discussed in the zero-shot analysis chapter, generalization regarding a potential distribution shift is important since the mask prompts will be created manually by users in Cytomine. This experiment thus evaluates how the performance is affected when the mask prompts used at inference time differ from those seen during the fine-tuning.

5.2 Fine-Tuning with Different Loss Functions

In this section, I first present the original loss function of SAM, along with the simplified version that I investigated. I then present the experimental results for this model. After that, I repeat the same process for SAM2.

5.2.1 Loss Experiment for SAM

The original loss function of SAM consists of three distinct components: a focal loss, a dice loss, and a mean squared error (MSE) loss between the predicted IoU score and the true IoU (computed between the predicted and ground truth masks). The respective weights are 20.0, 1.0, and 1.0, giving the following loss function:

$$L_{\text{total}} = 20.0 \cdot L_{\text{focal}} + 1.0 \cdot L_{\text{dice}} + 1.0 \cdot L_{\text{IoU}}$$

The focal loss is designed to address the potential class imbalance, indeed this is an important consideration in segmentation tasks since the background pixels can heavily outnumber the actual object pixels. This helps especially for the detection of small objects. The focal loss is defined as:

$$L_{\text{focal}} = \frac{1}{N} \sum_{j=1}^{N} \sum_{k \in \Omega} \alpha_t^{(j,k)} (1 - p_t^{(j,k)})^{\gamma} \cdot \text{BCE}(x^{(j,k)}, y^{(j,k)})$$

Here, α_t is a balancing factor, p_t is the predicted probability for the true class (the foreground), N is the number of objects in the batch, Ω is the set of all pixels in each image, and BCE denotes the binary cross-entropy loss. The index j refers to the current object being considered (which is equivalent to an image in the batch, since each image contains exactly one object in this setup), while k denotes the current pixel. Finally, x and y represent the predicted logit and the ground truth label for that pixel, respectively.

The dice loss measures the overlap between the predicted and the ground truth masks. It helps the model to predict masks that align well with the true objects, and it is defined as follows:

$$L_{\text{dice}} = \frac{1}{N} \sum_{j=1}^{N} \left(1 - \frac{2 \sum_k p_k^{(j)} y_k^{(j)} + 1}{\sum_k p_k^{(j)} + \sum_k y_k^{(j)} + 1} \right)$$

In this equation, p_k and y_k are the predicted and the ground truth values for the pixel k, and N is again the number of objects. The "+1" term in both the numerator and denominator is added to prevent division by zero.

Finally, the MSE loss term compares the predicted IoU value to the actual IoU between the predicted and the ground truth masks:

$$L_{\text{IoU}} = \frac{1}{N} \sum_{j=1}^{N} \left(\hat{s}_{\text{IoU}}^{(j)} - \text{IoU}^{(j)} \right)^2$$

Here, $\hat{s}_{\text{IoU}}^{(j)}$ is the predicted IoU value for the object j, and $\text{IoU}^{(j)}$ is the actual IoU value.

This original loss was designed for training on natural images in a general context, and was also used during the iterative training with interactive prompts. However, in my case, I am fine-tuning the models on a very specific domain (histopathology). Thus, using a different loss function could be more appropriate. Indeed, it is a common approach in the literature to select another loss which is more adapted to the use case during the fine-tuning. The original loss has already trained the model to understand general mask shapes and boundaries, so switching to a simpler or more targeted loss for fine-tuning makes sense. Many papers have done this with SAM, a few examples are [Gu et al., 2024], [Ma et al., 2024], and [Song et al., 2024].

The alternative loss that I used, which was initially proposed by Mr. Noé Gille during his internship, is the binary cross-entropy (BCE) with logits (from PyTorch). This loss is well-suited for binary pixel classification and is commonly used when the goal is to have a high pixel-wise accuracy, for example, it was used in both [Song et al., 2024] and [Ma et al., 2024]. Since it is simpler, it can also provide more stable fine-tuning on small datasets. Its formula is:

$$L_{\text{BCE}} = \frac{1}{N|\Omega|} \sum_{j=1}^{N} \sum_{k \in \Omega} \left[-y_j^{(k)} \log \sigma(x_j^{(k)}) - (1 - y_j^{(k)}) \log(1 - \sigma(x_j^{(k)})) \right]$$

For this experiment, I used all prompt types during fine-tuning. For the point prompts, I used multiple points per mask to improve results for both one- and multi-point inference. The negative point prompts were sampled within the bounding box prompts since the previous chapter showed that it was the best setting. For the mask prompts, I used the loose dilation masks that I introduced earlier. To clarify, when I say that I used all prompt types, this does not mean that I provided a fixed combination of all prompts at once during training. Instead, I followed an iterative training approach that included the full combination of prompts as well as its individual components. For example, if I considered the combination of bounding boxes and positive point prompts, the model would be trained not only on that combination, but also separately on box prompts alone and positive point prompts alone. This strategy ensured that the model was exposed to a variety of prompt combinations, not just a single fixed one. I applied the same principle in all fine-tuning experiments that are described in the rest of this chapter.

Since I did not fine-tune the image encoder, I precomputed all the image embeddings and stored them to reduce the training time. Indeed, passing the image through the encoder is the slowest part of inference, thus caching the embeddings helped me speed up the training significantly.

Regarding the training settings, I used a batch size of 64, since it was the largest size supported by the hardware. I performed 30 epochs with a learning rate of 1×10^{-5}. In fact, the loss plateaued around that point, thus I stopped the training. This low learning rate helped avoid aggressive updates of the model's pre-trained weights, which is important when fine-tuning such large models. Moreover, it provided better results than higher ones I tested. For the data, I used the three datasets introduced earlier, and

applied a 70%-15%-15% split for training, validation, and testing respectively within each dataset.

The results of this experiment are shown in Table 5.1. I used all metrics from the previous chapter (Dice, IoU, Precision, Recall) for the evaluation. I reported the performance for each dataset separately, as well as the average across all three of them (column "All"). This allowed me to check whether the performance improvement was uniform across the datasets, or whether one dataset dominated the overall gain. The row labeled "Custom" refers to results with the BCE loss; the "Original" row shows results with the original SAM loss.

Table 5.1: Results of Loss Experiment for SAM

Loss	Metric	LBTD-AGDC10	LBTD-NEO04	Camelyon16	All
Custom	Dice	0.864	0.924	0.898	**0.897**
	IoU	0.781	0.864	0.822	**0.824**
	Precision	0.867	0.924	0.902	0.900
	Recall	0.874	0.931	0.901	**0.903**
Original	Dice	0.818	0.898	0.851	0.856
	IoU	0.717	0.825	0.757	0.767
	Precision	0.919	0.961	0.950	**0.946**
	Recall	0.753	0.853	0.785	0.796

From Table 5.1, we can observe that the model fine-tuned using the simpler BCE loss function performs better overall than the one trained with the original SAM loss. More specifically, it achieves a Dice score that is 0.041 higher and an IoU score that is 0.057 higher. It also performs better across all metrics except for the precision, indicating that it tends to predict more pixels as belonging to the mask when they actually do not. Thus, it may slightly over-segment compared to the second model.

This result might seem a bit surprising. Indeed, one might expect the original loss, designed specifically for the model, to perform better. However, several hypotheses could explain this phenomenon. First, the BCE loss is simpler and may provide a more stable optimization objective, making it easier for the model to train effectively. Second, the BCE might implicitly encourage inclusiveness of pixels, and this over-segmentation could result in a better overlap with the ground truth. This would benefit metrics like Dice and Recall, at the expense of slightly reduced precision, which is what we observe here. Indeed, while the precision is lower, the recall is notably higher for the BCE-trained model, meaning it misses fewer true positive pixels from the ground truth mask.

For the distribution of performance across the datasets, it is relatively consistent. While some differences exist, none of them are extreme. The models performed best on the LBTD-NEO04 dataset and worst on LBTD-AGDC10.

In the end, the BCE loss led to better overall results compared to the original SAM loss. For this reason, I chose to use BCE in the remaining experiments of this chapter.

5.2.2 Loss Experiment for SAM2

The original loss function of SAM2 is very similar to that of SAM. More specifically, the original paper describes two variations of the loss. The first one is a linear combination of focal loss, Dice loss, and IoU loss. The second is an extension of the first one, where they added a cross-entropy loss term used to predict whether the target object is visible in the frame. Indeed, since SAM2 performs video segmentation, the target object may momentarily disappear from the frame. Since my thesis only deals with images, I used the first version of the loss, which contains the same components as in SAM, with a few small differences. One small change is that the IoU loss term in SAM2 uses Mean Absolute Error (MAE) instead of Mean Squared Error (MSE). Moreover, a sigmoid activation is applied to the IoU logits, it ensures that the predicted score remains within the [0,1] interval. The authors performed these modifications because they led to better results. The final IoU loss term that they used is given by:

$$L_{\text{IoU}} = \frac{1}{N} \sum_{j=1}^{N} \left| \hat{s}_{\text{IoU}}^{(j)} - \text{IoU}^{(j)} \right|$$

Here, N is still the number of objects in the batch, $\hat{s}_{\text{IoU}}^{(j)}$ is the IoU value predicted by the model, and $\text{IoU}^{(j)}$ is the ground-truth IoU for that object.

As for SAM, it is quite common in the literature to modify the loss function when fine-tuning SAM2. For example, [Yan et al., 2024] used a combination of BCE and Dice losses, while [T. Chen et al., 2024] used BCE with IoU loss. In my case, I chose a BCE + IoU loss, this choice was inspired by both [T. Chen et al., 2024] and [Ghosh, 2025]. The BCE component helps the model to learn pixel-wise accuracy, while the IoU term ensures that the predicted IoU score reflects the true segmentation overlap (thus, the quality of the predicted mask). The final loss function that I used is then:

$$L_{\text{total}} = \frac{1}{N|\Omega|} \sum_{j=1}^{N} \left[-\sum_{k \in \Omega} \left(y_j^{(k)} \log \sigma(x_j^{(k)}) + (1 - y_j^{(k)}) \log(1 - \sigma(x_j^{(k)})) \right) + \lambda \cdot \left| \hat{s}_{\text{IoU}}^{(j)} - \text{IoU}^{(j)} \right| \right]$$

The fine-tuning process for SAM2 followed the same structure as the one of SAM: I used the same prompt types, I also precomputed the image embeddings, and used identical training, validation, and testing splits.

For the training parameters, I set λ to 0.05 in my custom loss. Since the hierarchical encoder used in SAM2 produces larger image representations, I had to reduce the batch size to 16. I ran the training for 30 epochs, since the loss plateaued beyond that point. I kept the learning rate at 1×10^{-5}, as it gave stable results during testing. Moreover, I added weight decay to the Adam optimizer with a value of 1×10^{-4}. Indeed, this is a common regularization technique to reduce overfitting which is particularly useful when the datasets have a reduced size (which is the case here). This choice was also supported by several papers such as [M. Zhang et al., 2024], [L. Zhang et al., 2024], and [Ghosh, 2025].

The results of this second experiment are shown in Table 5.2, which follows the same structure as Table 5.1. As before, I broke down the results per dataset, and the "Custom" row refers to the loss function that I described above, while the "Original" row refers to the version of the SAM2 loss used in the original paper.

Table 5.2: Results of Loss Experiment for SAM2

Loss	Metric	LBTD-AGDC10	LBTD-NEO04	Camelyon16	All
Custom	Dice	0.875	0.931	0.905	**0.906**
	IoU	0.796	0.876	0.834	**0.837**
	Precision	0.870	0.924	0.908	0.904
	Recall	0.893	0.945	0.910	**0.915**
Original	Dice	0.836	0.911	0.870	0.874
	IoU	0.741	0.845	0.783	0.790
	Precision	0.924	0.963	0.956	**0.951**
	Recall	0.779	0.871	0.809	0.819

From Table 5.2, we can also see that the custom (simplified) loss function leads to better results than the original SAM2 loss. More particularly, we can draw the exact same conclusion as in the experiment with SAM. The model trained with the custom loss achieves a Dice score that is 0.032 higher and an IoU score that is 0.047 higher. However, its precision is also slightly lower than the one of the model trained with the original loss. On the other hand, the recall is much higher, about 0.1 higher, it suggests that the custom loss may again be causing a bit of over-segmentation compared to the second model. As before, the same hypotheses could explain this result. Overall, the performance remains relatively consistent across the datasets. The best results are again observed on LBTD-NEO04, while LBTD-AGDC10 remains the most challenging for both models.

If we now compare these results to those from Table 5.1, we see that SAM2 still performs slightly better than SAM, which is in line with the earlier zero-shot evaluation. The performance difference remains quite small: the best SAM2 model outperforms its SAM counterpart by 0.009 in Dice score and 0.013 in IoU. Nevertheless, this confirms that SAM2 is overall the better-performing model.

In the end, training SAM2 with the custom loss provides better results than using the original loss. So, I will keep this setting for the next experiments. And as already suggested in the zero-shot analysis, SAM2 remains the more promising model between the two.

5.3 Fine-Tuning across Different Datasets

In this section, I detail the results of the experiments related to the analysis of the influence of the training sets on the models performance.

To conduct this study, I reused the dataset splits for LBTD-AGDC10, LBTD-NEO04, and Camelyon16 that I defined in the previous section (Section 5.2). I then considered all possible combinations of these three datasets in order to perform a comprehensive evaluation. And, for each combination, I fine-tuned both SAM and SAM2. For the training parameters, I reused the exact same settings that I established in the previous section (Section 5.2) for both models. Indeed, these configurations had already shown good performance in the loss function experiment, so I saw no reason to modify them. I also used all prompt types for fine-tuning, though the detailed analysis of the prompt type influence is presented in the next section (Section 5.4).

Table 5.3 shows the results of this experiment for SAM. The first three columns indicate which datasets were included in the training: "Yes" means the dataset was used, while "-" indicates that it was not. The same four metrics as before are reported. Moreover, I broke down again the results by dataset and also averaged them in an "All" column.

Table 5.3: Results for Experiment across Datasets (SAM)

Train			**Test**				
LBTD-AGDC10	LBTD-NEO04	Camelyon16	Metric	LBTD-AGDC10	LBTD-NEO04	Camelyon16	All
-	-	-	Dice	0.547	0.641	0.611	0.605
			IoU	0.442	0.537	0.511	0.503
			Precision	0.509	0.633	0.578	0.577
			Recall	0.744	0.768	0.754	0.755
Yes	-	-	Dice	0.848	0.870	0.831	0.845
			IoU	0.758	0.792	0.733	0.754
			Precision	0.867	0.914	0.848	0.869
			Recall	0.846	0.854	0.835	0.842
-	Yes	-	Dice	0.830	0.916	0.869	0.873
			IoU	0.735	0.851	0.777	0.788
			Precision	0.808	0.915	0.850	0.858
			Recall	0.883	0.924	0.901	0.903
-	-	Yes	Dice	0.838	0.892	0.890	0.879
			IoU	0.744	0.816	0.810	0.797
			Precision	0.859	0.930	0.899	0.898
			Recall	0.836	0.872	0.888	0.873
Yes	Yes	-	Dice	0.857	0.920	0.874	0.882
			IoU	0.770	0.858	0.785	0.801
			Precision	0.857	0.919	0.857	0.873
			Recall	0.873	0.927	0.904	0.903
Yes	-	Yes	Dice	0.857	0.900	0.890	0.886
			IoU	0.772	0.830	0.811	0.808
			Precision	0.872	0.929	0.908	0.906
			Recall	0.856	0.888	0.880	0.877
-	Yes	Yes	Dice	0.850	0.922	0.895	0.892
			IoU	0.762	0.861	0.818	0.817
			Precision	0.847	0.924	0.904	0.897
			Recall	0.874	0.926	0.894	0.898
Yes	Yes	Yes	Dice	**0.864**	**0.924**	**0.898**	**0.897**
			IoU	**0.781**	**0.864**	**0.822**	**0.824**
			Precision	**0.867**	**0.924**	**0.902**	**0.900**
			Recall	**0.874**	**0.931**	**0.901**	**0.903**

From Table 5.3, several observations can be made. First, and unsurprisingly, the best configuration is the one where the model is trained using all three datasets. This is expected, indeed training on the full range of data distributions enables the model to better generalize across the entire test set (which is also composed of these three datasets).

Second, the worst-performing configuration is, as expected, the zero-shot evaluation. In this case, the model has not been trained at all, thus its performance is logically poor. However, we clearly see that training has significantly improved the results: the Dice score increases by 0.292 and the IoU score by 0.321 when comparing the best fine-tuned model to the zero-shot one. This confirms the importance of fine-tuning. Among the configurations where only a single dataset is used for training, the best results are obtained with the Camelyon16 dataset. This suggests that this dataset brings more useful knowledge to the model than the other two. A possible explanation is that Camelyon16 is the largest dataset, which could help the model to learn more effectively. Regarding the combinations involving two datasets, the best results come from using LBTD-NEO04 and Camelyon16.

Finally, we see that performance remains relatively consistent across datasets. No significant drops are observed, which indicates a good level of generalization from the models across the various data distributions.

Table 5.4 presents the results for SAM2, and follows the same format as Table 5.3.

From this table, we can draw similar conclusions to those from the previous one. The best-performing model is again the one trained on all three datasets, while the worst corresponds to the zero-shot evaluation.

Camelyon16 remains the most effective dataset when training on a single one, which aligns with the previous findings. Similarly, the combination of LBTD-NEO04 and Camelyon16 continues to provide the best performance among the two-dataset configurations. The results across the three datasets are still fairly consistent with no extreme variation.

We also observe, once again, the superior performance of SAM2 compared to SAM. When comparing the best models for both, SAM2 achieves a Dice score that is 0.009 higher and an IoU score that is 0.013 higher.

To conclude, this experiment confirms that using all the available datasets for fine-tuning is the most effective approach. However, the datasets do not contribute equally to the performance improvement. More particularly, when Camelyon16 is already included, adding the other datasets leads to only small gains (just a few thousandths in most cases). These gains are not really significant and come at the cost of longer training times.

Table 5.4: Results for Experiment across Datasets (SAM2)

Train			Test				
LBTD-AGDC10	LBTD-NEO04	Camelyon16	Metric	LBTD-AGDC10	LBTD-NEO04	Camelyon16	All
-	-	-	Dice	0.643	0.726	0.662	0.675
			IoU	0.515	0.608	0.545	0.555
			Precision	0.578	0.660	0.626	0.625
			Recall	0.859	0.909	0.804	0.844
Yes	-	-	Dice	0.866	0.894	0.810	0.844
			IoU	0.783	0.825	0.717	0.760
			Precision	0.869	0.912	0.829	0.860
			Recall	0.877	0.895	0.822	0.853
-	Yes	-	Dice	0.847	0.926	0.883	0.887
			IoU	0.757	0.868	0.799	0.808
			Precision	0.827	0.919	0.875	0.876
			Recall	0.893	0.940	0.901	0.909
-	-	Yes	Dice	0.853	0.917	0.903	0.896
			IoU	0.765	0.854	0.829	0.822
			Precision	0.846	0.918	0.899	0.892
			Recall	0.880	0.925	0.913	0.909
Yes	Yes	-	Dice	0.871	0.929	0.884	0.893
			IoU	0.789	0.873	0.801	0.817
			Precision	0.870	0.926	0.883	0.891
			Recall	0.885	0.939	0.896	0.905
Yes	-	Yes	Dice	0.873	0.918	0.904	0.901
			IoU	0.792	0.857	0.832	0.830
			Precision	0.871	0.926	0.909	0.905
			Recall	0.888	0.921	0.906	0.906
-	Yes	Yes	Dice	0.860	0.931	**0.906**	0.902
			IoU	0.776	0.875	**0.834**	0.832
			Precision	0.846	0.922	**0.906**	0.897
			Recall	0.895	0.945	**0.911**	0.917
Yes	Yes	Yes	Dice	**0.875**	**0.931**	0.905	**0.906**
			IoU	**0.796**	**0.876**	0.834	**0.837**
			Precision	**0.870**	**0.924**	0.908	**0.904**
			Recall	**0.893**	**0.945**	0.910	**0.915**

5.4 Fine-Tuning with Varying Prompt Types

In this section, I present the experiments that I conducted to evaluate the impact of the different prompt type configurations during the fine-tuning of both models. As specified in Section 5.1, the main goal of these experiments was to determine whether fine-tuning a model with a specific prompt type or combination could lead to better performance for that setting, compared to training on all prompt types together.

To do this, I selected a set of prompt combinations to test. Most of these were based on the ones I already analyzed in the zero-shot evaluation chapter. I reused the box prompts, positive and negative point prompts, and their various combinations. I also considered some combinations involving mask prompts. However, to save time (since fine-tuning each configuration is time-consuming), I chose not to test every possible combination. For example, I again excluded experiments using only negative point prompts, for the same reasons explained in Chapter 4. As for combinations involving both box and mask prompts, I left out most of them because, as discussed earlier, they are not user-friendly and are unlikely to be adopted in practice on Cytomine. Therefore, I included only one experiment using all prompt types together, without exploring every of these "box-mask" combinations in detail.

As in the previous section (Section 5.3), I used multiple points per mask during training, sampled the negative point prompts within the bounding box, and used the loose dilation masks for the mask prompts. The training, validation, and test sets were exactly the same as in the previous section, and I kept the same training parameters.

The results for SAM are shown in Table 5.5. The table uses the same structure and evaluation metrics as Table 5.3, with results shown per dataset as well as overall. The main difference is in the first four columns, which now represent the different prompt types. The same notation is used for combinations: "Yes" indicates that the prompt type was used, and "-" indicates that it was not.

From Table 5.5, we observe that the best performance is achieved when the model is trained using all prompt types. In this setting, the model reaches a Dice score of 0.897 and an IoU score of 0.824. One possible explanation for this result is that, since the model is trained not only on the full combination but also on its subsets, it sees more training cases than the other configurations. This greater exposure could help the model to generalize better and thus improve its overall performance.

Among the single prompt types, the best results are obtained with mask prompts. This is a bit surprising, as they slightly outperform box prompts by 0.008 in Dice score and 0.011 in IoU. This finding differs from the zero-shot analysis, where box prompts provided the best results. However, as previously discussed, the robustness of mask prompts under a potential distribution shift still needs to be investigated (and this will be analyzed in the final section of this chapter, Section 5.8).

Overall, the results across the different prompt configurations remain quite close. The gap between the lowest and highest Dice scores is only 0.029, and for IoU, it is just 0.045. This suggests that while combining multiple prompt types may improve the performance slightly, the gain is not particularly large. We also see, once again, that the model performs fairly consistently across the three datasets, without any significant performance drop on any of them.

Finally, if we focus on the combination of box and positive point prompts, which was the best-performing setup in the zero-shot experiments, it is no longer the case. However, the difference is minimal: the Dice score is just 0.02 lower than the best configuration, which indicates that it remains a very effective choice.

Table 5.5: Results for Experiment with Varying Prompt Types (SAM)

Prompt types				Test				
Box	Mask	Points (+)	Points (-)	Metric	LBTD-AGDC10	LBTD-NEO04	Camelyon16	All
Yes	-	-	-	Dice	0.832	0.910	0.870	0.873
				IoU	0.741	0.843	0.782	0.789
				Precision	0.829	0.913	0.878	0.877
				Recall	0.858	0.917	0.872	0.881
Yes	-	Yes	-	Dice	0.843	0.909	0.876	0.877
				IoU	0.751	0.841	0.788	0.794
				Precision	0.844	0.912	0.885	0.883
				Recall	0.866	0.916	0.878	0.885
Yes	-	-	Yes	Dice	0.838	0.910	0.873	0.875
				IoU	0.746	0.843	0.785	0.792
				Precision	0.851	0.916	0.888	0.888
				Recall	0.844	0.912	0.868	0.875
Yes	-	Yes	Yes	Dice	0.844	0.908	0.878	0.878
				IoU	0.754	0.841	0.792	0.797
				Precision	0.850	0.912	0.891	0.888
				Recall	0.858	0.915	0.875	0.882
-	Yes	-	-	Dice	0.843	0.910	0.882	0.881
				IoU	0.752	0.843	0.799	0.800
				Precision	0.856	0.916	0.881	0.885
				Recall	0.847	0.912	0.890	0.887
-	Yes	Yes	-	Dice	0.848	0.913	0.885	0.884
				IoU	0.759	0.847	0.803	0.805
				Precision	0.860	0.921	0.894	0.894
				Recall	0.854	0.914	0.886	0.887
-	Yes	-	Yes	Dice	0.843	0.915	0.884	0.883
				IoU	0.756	0.852	0.803	0.806
				Precision	0.852	0.917	0.893	0.891
				Recall	0.849	0.920	0.884	0.886
-	Yes	Yes	Yes	Dice	0.849	0.911	0.882	0.883
				IoU	0.760	0.845	0.799	0.803
				Precision	0.858	0.918	0.896	0.894
				Recall	0.856	0.914	0.879	0.883
-	-	Yes	-	Dice	0.838	0.896	0.867	0.868
				IoU	0.742	0.821	0.774	0.779
				Precision	0.844	0.906	0.877	0.878
				Recall	0.861	0.902	0.873	0.878
-	-	Yes	Yes	Dice	0.840	0.905	0.876	0.876
				IoU	0.749	0.836	0.789	0.793
				Precision	0.853	0.909	0.894	0.889
				Recall	0.852	0.912	0.870	0.877
Yes	Yes	Yes	Yes	Dice	**0.864**	**0.924**	**0.898**	**0.897**
				IoU	**0.781**	**0.864**	**0.822**	**0.824**
				Precision	**0.867**	**0.924**	**0.902**	**0.900**
				Recall	**0.874**	**0.931**	**0.901**	**0.903**

From Table 5.6, we can see that the best configuration is once again the one where all prompt types are used, likely for the same reasons discussed earlier. Among the individual prompt types, the mask prompt still gives the best results, although the difference compared to the box prompt is still small.

The performance across the different configurations is still very close. This time, the gap between the highest and lowest Dice scores is 0.031, and for the IoU, it is 0.048. This confirms that adding more prompt types does not lead to a significant improvement in performance.

Table 5.6: Results for Experiment with Varying Prompt Types (SAM2)

Prompt types				**Test**				
Box	Mask	Points (+)	Points (-)	Metric	LBTD-AGDC10	LBTD-NEO04	Camelyon16	All
Yes	-	-	-	Dice	0.845	0.919	0.884	0.885
				IoU	0.755	0.858	0.801	0.806
				Precision	0.845	0.921	0.892	0.890
				Recall	0.867	0.925	0.886	0.892
Yes	-	Yes	-	Dice	0.855	0.917	0.884	0.887
				IoU	0.768	0.856	0.801	0.808
				Precision	0.860	0.923	0.901	0.898
				Recall	0.874	0.921	0.879	0.889
Yes	-	-	Yes	Dice	0.849	0.917	0.886	0.886
				IoU	0.763	0.856	0.806	0.810
				Precision	0.851	0.916	0.892	0.890
				Recall	0.871	0.926	0.891	0.896
Yes	-	Yes	Yes	Dice	0.854	0.915	0.885	0.886
				IoU	0.768	0.854	0.803	0.809
				Precision	0.859	0.918	0.899	0.895
				Recall	0.870	0.921	0.882	0.890
-	Yes	-	-	Dice	0.863	0.925	0.898	0.898
				IoU	0.780	0.869	0.824	0.826
				Precision	0.862	0.924	0.903	0.900
				Recall	0.874	0.931	0.900	0.903
-	Yes	Yes	-	Dice	0.866	0.924	0.898	0.898
				IoU	0.783	0.866	0.821	0.825
				Precision	0.868	0.924	0.907	0.903
				Recall	0.882	0.931	0.896	0.902
-	Yes	-	Yes	Dice	0.863	0.921	0.897	0.896
				IoU	0.780	0.863	0.822	0.824
				Precision	0.857	0.916	0.902	0.896
				Recall	0.880	0.933	0.898	0.904
-	Yes	Yes	Yes	Dice	0.862	0.919	0.895	0.894
				IoU	0.778	0.860	0.817	0.820
				Precision	0.868	0.924	0.907	0.903
				Recall	0.875	0.924	0.890	0.896
-	-	Yes	-	Dice	0.844	0.904	0.873	0.875
				IoU	0.752	0.835	0.782	0.789
				Precision	0.855	0.916	0.894	0.891
				Recall	0.865	0.907	0.865	0.876
-	-	Yes	Yes	Dice	0.852	0.913	0.884	0.885
				IoU	0.764	0.849	0.801	0.805
				Precision	0.852	0.913	0.897	0.892
				Recall	0.877	0.922	0.883	0.892
Yes	Yes	Yes	Yes	Dice	**0.875**	**0.931**	**0.905**	**0.906**
				IoU	**0.796**	**0.876**	**0.834**	**0.837**
				Precision	**0.870**	**0.924**	**0.908**	**0.904**
				Recall	**0.893**	**0.945**	**0.910**	**0.915**

We also continue to observe the trend noted in the previous sections: SAM2 performs better than SAM. The best-performing SAM2 model achieves a Dice score 0.009 higher and an IoU score 0.013 higher than the best model based on SAM.

In summary, the best performance is achieved when using all prompt types during training. However, the differences between the various configurations are small, meaning that adding extra prompt types does not lead to large gains. We also confirm the earlier trend that SAM2 continues to outperform SAM after fine-tuning.

5.5 Fine-Tuning the Entire Models

In this section, I analyze the effect of fine-tuning the entire models, including the image encoder, on the global performance. In the previous sections, I only fine-tuned the prompt encoder and the mask decoder. This time, the image encoder is also included in the training process.

As in the earlier experiments, I used the same datasets for training and included all of them, since this configuration gave the best results. I also kept all prompt types during training, as the previous section (Section 5.4) showed that it offered slightly better performance. However, this time, I did not pre-compute the image embeddings since the image encoder is now updated during training.

Most of the training parameters were kept the same as in the previous sections. I did not modify the number of epochs, the learning rate, the weight decay, or other settings. However, I had to adjust the batch sizes for both models. Indeed, previously, I could use larger batch sizes because the image embeddings were pre-computed and not involved in the backpropagation. But now, since the image encoder is being trained, gradients must be computed for it as well, which significantly increases the memory usage. As a result, I reduced the batch size to 3 for SAM and 4 for SAM2.

The results of this experiment are shown in Table 5.7 for both SAM and SAM2.

Table 5.7: Results of Fine-Tuning the Entire Models

Model	Metric	LBTD-AGDC10	LBTD-NEO04	Camelyon16	All
SAM	Dice	0.860	0.913	0.873	0.881
	IoU	0.774	0.850	0.787	0.801
	Precision	0.907	0.948	0.937	0.934
	Recall	0.829	0.888	0.826	0.844
SAM2	Dice	0.871	0.921	0.890	**0.894**
	IoU	0.790	0.861	0.810	**0.819**
	Precision	0.905	0.948	0.941	**0.936**
	Recall	0.849	0.900	0.850	**0.863**

From Table 5.7, we can see that SAM2 continues to outperform SAM, although the difference remains quite small. Moreover, we see that both models trained with all parameters, including the image encoder, actually perform slightly worse than the best models trained in the previous sections, where only the prompt encoder and mask decoder were fine-tuned. For example, the best SAM model from Table 5.5 achieves a Dice score that is 0.016 higher and an IoU that is 0.023 higher. Similarly, the best SAM2 model from Table 5.6 performs better by 0.012 Dice and 0.005 IoU compared to the fully fine-tuned version.

This result is somewhat surprising, as one might expect that fine-tuning the full model would lead to better performance. However, there are two main hypotheses that could explain this result.

The first reason could be related to the batch size. Indeed, since training the full models requires computing and storing gradients for the image encoder, I had to significantly reduce the batch sizes. As a result, the gradient estimates are based on fewer samples, which can make the training less stable or slower to converge. In the previous experiments, I used batch sizes of 16 and 64, while here I had to reduce them to 3 and 4. This difference could have negatively impacted the training dynamics and thus, the final performance.

The second explanation is linked to overfitting, as mentioned in [L. Zhang et al., 2024]. Fine-tuning the full models, which have a very large number of parameters, on relatively small datasets can lead to overfitting or even forgetting. In this case, not fine-tuning the image encoder meant training with about 80 million fewer parameters. When the full model is fine-tuned, the image encoder may overfit to the limited data, and thus, it loses some of its generalization capabilities, which causes a slight drop in performance.

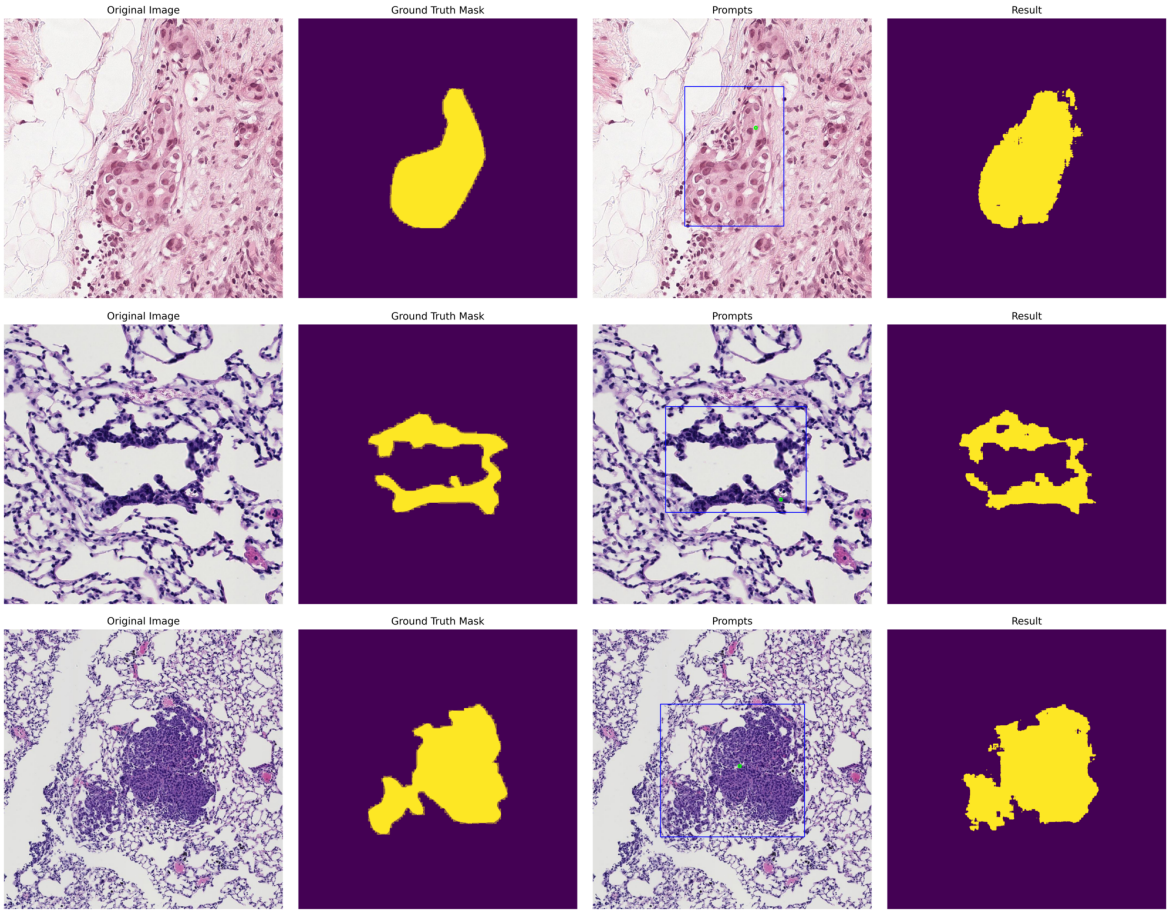
To summarize, fully fine-tuning the models did not lead to better results, in fact, it slightly reduced the performance. Given that full fine-tuning also requires significantly more time and resources, it is thus more practical and effective to fine-tune only the prompt encoder and the mask decoder.

5.6 Segmentation Visualization

The previous sections allowed me to identify the best configuration to obtain the most effective model, which turned out to be SAM2 trained on all datasets using all prompt types. I also determined the best setup for SAM. However, before moving on to post-processing strategies, it is important to first visualize the masks produced by these models in order to evaluate their consistency and to check whether post-processing is actually necessary. Indeed, although the performance metrics were generally strong, it is still important to verify that the masks align with those scores and to detect any potential problems (for example the possible over-segmentation that I mentioned earlier). For this reason, in this section I present a few example segmentations produced by the best-performing models for both SAM and SAM2.

Figure 5.1 and Figure 5.2 show two sets of examples corresponding to segmentations produced by SAM and SAM2, respectively. Both figures follow the same structure: the first column shows the original image, the second displays the ground truth mask, the third shows the prompts used for the models, with the bounding box shown in blue and a single positive point prompt in green (I used this combination for readability), and the final column shows the predicted mask generated by the model.

Figure 5.1: Fine-Tuning Segmentation Examples (SAM)

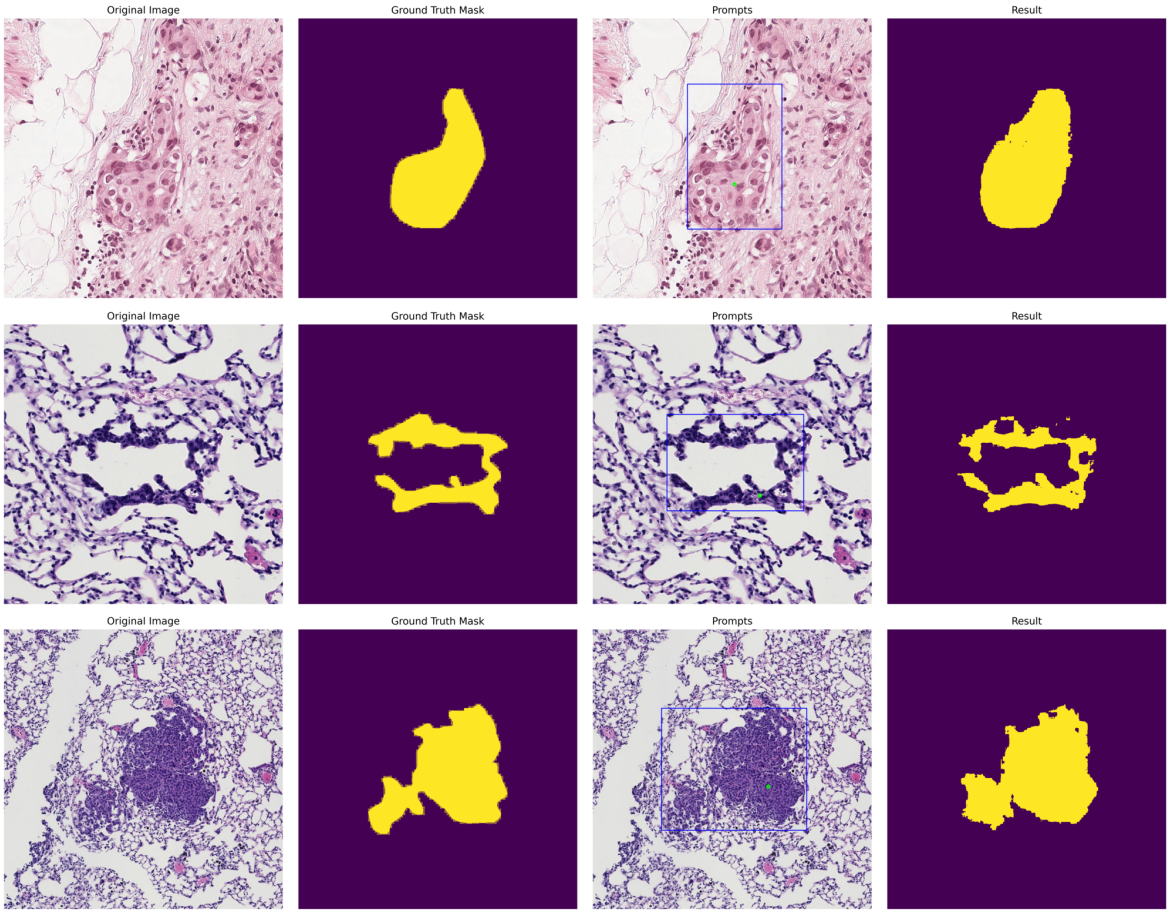


From Figure 5.1, we can see that the predicted masks generally align well with the ground truth, which confirms the good metrics obtained in the previous experiments. In particular, subfigures 2 and 3 show strong correspondence between the predictions and the ground truth. Subfigure 1 is slightly less accurate, indeed the predicted mask deviates more from the true shape.

Despite these overall good results, we can still observe a few issues. Firstly, the predicted masks contain small holes (regions within the mask that should be filled but that are missing). This is observable in all three examples and fixing it would likely improve the performance metrics since it would increase the overlap with the ground truth.

Secondly, the masks show a bit of noise. For example, in subfigure 1, the predicted mask includes a small artifact in the top right that is disconnected from the main region. In subfigure 2, the bottom left part of the mask is incorrectly segmented as disconnected when it should be continuous. Subfigure 3 also shows minor noise. These imperfections prove that some additional post-processing is needed in order to clean up the masks and improve their usability.

Figure 5.2: Fine-Tuning Segmentation Examples (SAM2)



From Figure 5.2, we can make similar observations as before. The predicted masks still match the ground truth well, and they appear slightly cleaner than those produced by SAM. There are fewer visible artifacts in these predictions compared to those shown in Figure 5.1. However, there are still some issues. For example, in the second subfigure, we can still see small amounts of noise, and all three masks contain holes. Overall, the results look better than those from SAM, which is consistent with the performance metrics.

In conclusion, the masks predicted by the models are generally satisfactory and follow the ground truth well. However, post-processing is clearly needed to address recurring issues like holes and minor noise.

5.7 Exploring Post-Processing Strategies

In this section, I present the different post-processing strategies that I explored. As previously mentioned, to ensure that the output masks are "user-friendly" and easy for users to edit, some post-processing is necessary. The raw masks predicted by the models often contain noise or small disconnected regions, as shown in the examples from the previous section (Section 5.6). Therefore, it is important to clean up these masks. Moreover, if the post-processing is well designed, it might also help to improve the model's performance.

I investigated three main post-processing approaches, which I will now describe in detail. The first one relies on traditional computer vision techniques using OpenCV [Bradski, 2000]. The second is a deep learning-based method using a model called CascadePSP, proposed in [Cheng et al., 2020]. The third approach reuses the predicted masks as input prompts to the models to try to refine the results.

5.7.1 Post-Processing using OpenCV

OpenCV is the world's most widely used computer vision library. Given its popularity and reliability, it made sense to test whether it could help improve the predicted masks using some classical image processing methods.

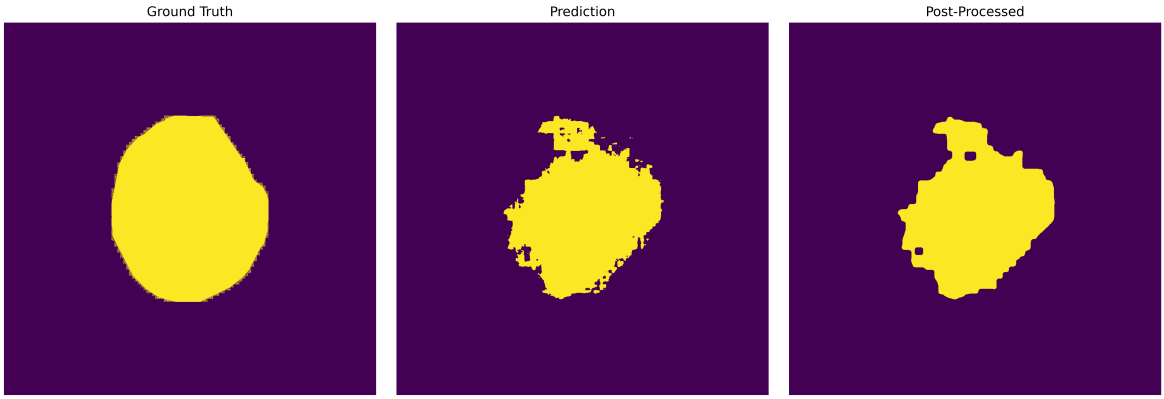
The strategy that I followed is simple. First, I applied a morphological opening operation to the predicted mask. Morphological opening consists of an erosion followed by a dilation. This helps to remove small noisy elements or small isolated parts of the mask. The erosion shrinks or eliminates the small regions, while the dilation restores the shape of the remaining parts (hopefully without reintroducing the noise).

After the opening step, I applied a morphological closing, which is the opposite: a dilation followed by an erosion. This operation tends to fill the small holes and connect parts of the mask that are close but not touching due to little segmentation errors. For example, in subfigure 2 of Figure 5.1, the predicted mask has a shape similar to an upside-down "C". The left and right parts at the bottom are disconnected, even though they should be connected according to the ground truth. The morphological closing helps address this kind of issue. In the worst case, it only slightly enlarges the mask, but at best, it reconnects parts that were supposed to be joined.

Finally, I filtered out any "too small" connected components that might still remain and might be due to noise. More specifically, I removed any component whose area was less than 10% of the largest connected component. This further cleans the mask and ensures that only the main, relevant parts are kept.

Figure 5.3 shows an example of this post-processing method. The first image is the ground truth mask, the second is the raw prediction from SAM2, and the third is the result after applying the post-processing. We can see that some of the holes have been filled, and the overall mask looks cleaner and more consistent.

Figure 5.3: Post-Processing Example with OpenCV



After visualizing the results of this post-processing method, I conducted more detailed experiments in order to evaluate its impact on the different metrics (including those focused on the qualitative aspects of mask quality, that I introduced in Chapter 3). The results for SAM and SAM2 are presented in Table 5.8 and Table 5.9, respectively.

Table 5.8: Results for Post-Processing using OpenCV (SAM)

Metric	**SAM**	**SAM (Post-pro)**
Dice	**0.897**	0.896
IoU	**0.824**	0.823
Precision	**0.900**	**0.900**
Recall	**0.903**	0.901
Compactness	0.362	**0.548**
Solidity	0.564	**0.891**
Perimeter Smoothness	0.854	**1.188**
Connected Component	5.399	**1.017**
Size Retention	/	**0.996**

Table 5.9: Results for Post-Processing using OpenCV (SAM2)

Metric	**SAM2**	**SAM2 (Post-pro)**
Dice	**0.906**	0.904
IoU	**0.837**	0.835
Precision	**0.904**	**0.904**
Recall	**0.915**	0.913
Compactness	0.323	**0.545**
Solidity	0.496	**0.887**
Perimeter Smoothness	0.795	**1.187**
Connected Component	7.196	**1.018**
Size Retention	/	**0.995**

From these two tables, several observations can be made. First, both the Dice score and the IoU slightly decrease after applying the post-processing. This is not surprising, since the method removes some parts of the predicted masks, more particularly the irregular or noisy regions, those parts might still have overlapped with the ground truth to some point. Thus, trimming them can slightly lower the scores. However, the decrease is extremely small (only 0.001 for both metrics), and therefore not really significant in practice. The precision remains unchanged after post-processing, which makes sense since we are only removing small, most likely incorrect parts from the masks, and not adding new ones. The recall drops slightly, as some of the removed pixels might have actually belonged to the ground truth (this is consistent with what we observe).

Now, if we look at the qualitative metrics: the compactness improves after post-processing, increasing from around 0.2. This indicates that the masks are more tightly shaped and closer to circular forms, which is expected since the edge noise has been reduced. Solidity also increases (from around 0.3), likely due to holes being filled during the morphological closing step, and some of the noise being removed. The perimeter smoothness improves significantly, meaning the contours are now smoother and less noisy. Initially, the smoothness values below 1 suggested jagged shapes (with a shorter actual perimeter than the one of the approximating polygon); after post-processing, the values rise above 1, showing that the masks have become more regular. For the number of connected components, it decreases to nearly 1. This is highly important, indeed, fewer disconnected parts make the masks easier for users to work with. Finally, the size retention metric remains high, around 0.995, indicating that only a very small portion of the mask was removed, yet this small change led to significant improvements in the qualitative metrics.

To sum up, this post-processing method is very effective. It has virtually no negative impact on the main performance metrics, while greatly enhancing all qualitative metrics. By trimming only a tiny portion of the mask, it produces cleaner, more regular, and more user-friendly segmentations.

5.7.2 Post-Processing using CascadePSP

CascadePSP, introduced in [Cheng et al., 2020], is a deep learning model which is designed to refine high-resolution segmentation masks. Its goal is to generate more accurate masks from the masks it receives as input (which in this case are the predicted masks from SAM and SAM2) by improving the boundaries. The model follows an encoder-decoder architecture based on PSPNet (using ResNet-50 as backbone) for the feature extraction. After extracting multi-scale features, a sequence of refinement steps (a cascade) is applied and progressively improves the segmentation quality. To achieve this, the model uses upsampling layers with some skip connections, which help to preserve and also enhance the mask details.

An example of the application of CascadePSP to a natural image is presented in Figure 5.4 (taken from the original code repository). As we can see, the model significantly enhances the boundaries of the predicted mask, making it more consistent with the target object, the hand, and thus improves the overall segmentation performance. Therefore, it seems interesting to consider CascadePSP for post-processing. However, an important concern is whether it will work as well on histopathology images.

Figure 5.4: Example with CascadePSP on a natural image. Source: GitHub

Figure 5.5 shows an example of post-processing using CascadePSP on a histopathology image. The first image represents the ground truth, the second is the original predicted mask, and the third one is the result after applying CascadePSP. We can observe that CascadePSP has indeed modified the boundaries of the mask. They appear slightly smoother, especially at the top of the object where the shape becomes less "rectangular", but the overall result remains noisy. In fact, some noise is still clearly visible, and in some areas, CascadePSP even removed pixels that should have been kept. The final mask is not significantly cleaner, and in some regions, it even seems more irregular or fragmented than before. Therefore, CascadePSP does not seem to be an effective solution for producing cleaner, user-friendly masks.

Figure 5.5: Post-Processing Example with CascadePSP

Table 5.10 and Table 5.11 present the results of the quantitative experiments that I conducted to evaluate the efficiency of CascadePSP compared to the previous post-processing method.

Table 5.10: Results for Post-Processing using CascadePSP (SAM)

Metric	SAM	SAM (Post-pro)
Dice	**0.897**	0.890
IoU	**0.824**	0.814
Precision	**0.900**	0.899
Recall	**0.903**	0.892
Compactness	0.362	**0.426**
Solidity	0.564	**0.658**
Perimeter Smoothness	0.854	**0.964**
Connected Component	5.399	**2.464**
Size Retention	/	**1.036**

Table 5.11: Results for Post-Processing using CascadePSP (SAM2)

Metric	SAM2	SAM2 (Post-pro)
Dice	**0.906**	0.898
IoU	**0.837**	0.825
Precision	**0.904**	0.903
Recall	**0.915**	0.901
Compactness	0.323	**0.426**
Solidity	0.496	**0.657**
Perimeter Smoothness	0.795	**0.95**
Connected Component	7.196	**2.512**
Size Retention	/	**1.031**

From these tables, we can see that the post-processing method using CascadePSP performs worse than the one using OpenCV. Indeed, it also causes a drop in Dice score and IoU, but this time the decrease is slightly more important (around 0.008). Moreover, both precision and recall are reduced here. One might argue that these differences are small and may not be very significant in practice, however, the more important differences are observed in the qualitative mask quality metrics.

For both SAM and SAM2, CascadePSP leads to worse values across nearly all qualitative metrics. The compactness drops to 0.426, compared to 0.54 with OpenCV, suggesting that the masks are less tight and more spread out. Solidity is also lower by around 0.2, which suggests more holes or irregularities in the masks. The perimeter smoothness is reduced as well, meaning that the contours are more jagged/noisy. Finally, the number of connected components is roughly doubled, this indicates that the masks are more fragmented, and thus, less user-friendly.

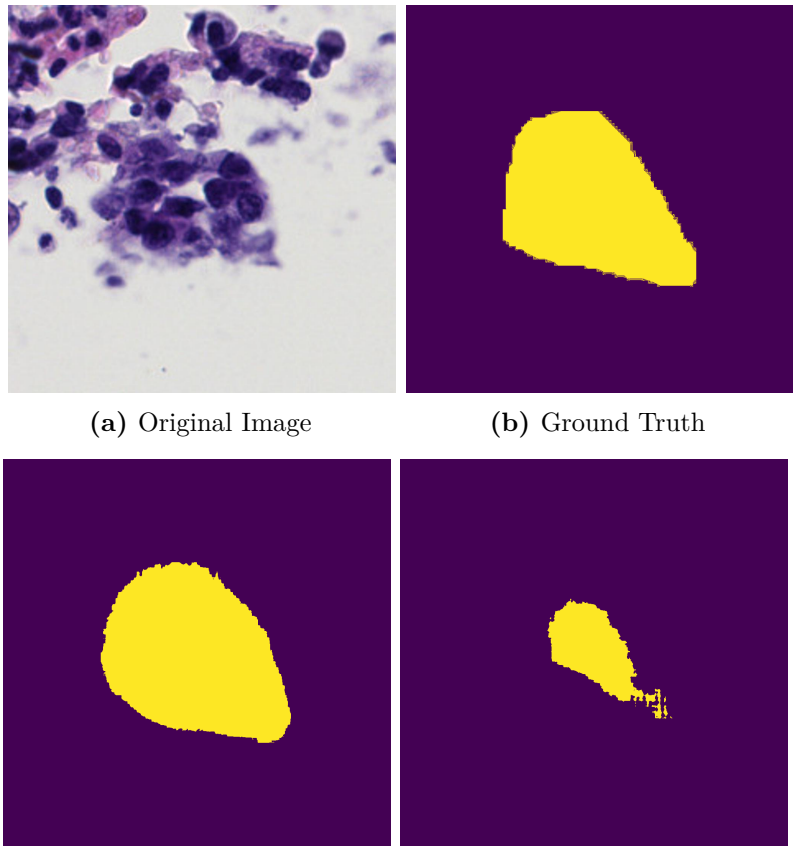
Moreover, the size retention is slightly greater than 1, which means that instead of removing noise, CascadePSP tends to add content to the mask (likely contributing to the observed decline in quality).

Overall, this method is clearly less effective than the OpenCV one. Not only does it slightly lower the performance metrics, but more importantly, it also significantly degrades the quality and usability of the masks.

5.7.3 Post-Processing using Recirculation

The idea behind this third post-processing method is to check whether feeding the predicted mask back into the model as an input prompt could be used as a post-processing technique and potentially improve the performance. This method is straightforward: first, the model is given the initial input prompts and returns a predicted mask. Then, this predicted mask is added to the set of original prompts and given back to the model to generate a second mask. This "recirculated" mask is the final segmentation result. In theory, I could repeat this process several times to progressively improve the mask quality.

Figure 5.6: Post-Processing Example with Recirculation



(a) Original Image **(b)** Ground Truth

(c) Predicted Mask **(d)** Recirculated Mask

Figure 5.6 shows an example of this method. The top-left image is the original image given as input, the top-right is the ground truth, and the two bottom images show the first predicted mask and the final recirculated mask. From this example, we can clearly see that the recirculated mask is much worse than the original prediction. Indeed, the first mask was quite close to the ground truth, but after the recirculation, the result is significantly smaller, noisier, and covers less of the true object. A potential explanation for this poor result is that the predicted mask does not follow the same distribution as the loose dilation masks the models were fine-tuned on. Therefore, the model is not able to correctly interpret the recirculated prompt, which highly degrades the output. Thus, the model does not seem to generalize well when a distribution shift occur regarding the masks.

Tables 5.12 and 5.13 show the quantitative results for this post-processing method. The numbers confirm what we observed visually. Except for the precision, all the other metrics drop significantly. The Dice score drops by 0.3 for SAM and 0.5 for SAM2, while recall is divided by two for SAM and divided by four for SAM2. The slight improvement in the precision is not surprising, indeed, the model outputs fewer pixels, which makes the predictions more "conservative", but at the cost of missing large parts of the actual mask. Due to these very poor results, I did not evaluate the qualitative shape metrics, as it is clear that the method is not suitable for post-processing.

Table 5.12: Results for Post-Processing using Recirculation (SAM)

Metric	**SAM**	**SAM (Post-pro)**
Dice	**0.897**	0.565
IoU	**0.824**	0.424
Precision	0.900	**0.972**
Recall	**0.903**	0.427

Table 5.13: Results for Post-Processing using Recirculation (SAM2)

Metric	**SAM2**	**SAM2 (Post-pro)**
Dice	**0.906**	0.399
IoU	**0.837**	0.275
Precision	0.904	**0.927**
Recall	**0.915**	0.277

In summary, this recirculation method is not a viable post-processing strategy. Among all the different techniques that I evaluated, the OpenCV-based method remains the most effective. Indeed, not only does it preserve the various performance metrics, but it also significantly improves the mask quality. Moreover, it is much faster, since it avoids calling some other neural network (compared to the second method) and only uses simple morphological operations. For these reasons, I select the OpenCV-based post-processing technique for the remainder of this master's thesis.

5.8 Generalization Under Distribution Shift for Mask Prompts

In this section, I evaluate the performance of the models when faced with a shift in the distribution of mask prompts. As previously seen in the post-processing section (Section 5.7), the models do not appear robust when such a shift happens. This section provides a more in-depth analysis of that behavior. To investigate, I ran two sets of experiments for both SAM and SAM2.

The first set examines how the performance changes when only a mask is used as input and its distribution is modified. The second set focuses on the same mask shift but evaluates the performance when the mask is combined with all other prompts. This allows us to verify whether the performance degradation occurs in both of these scenarios. To introduce a distribution shift, I used the "scribbled masks" that I previously analyzed in the zero-shot evaluation chapter. Since this mask generation process was already done by Mr. Noé Gille, it was a convenient and efficient choice for this experiment.

Tables 5.14 and 5.15 present the results for SAM. They show that the model is clearly not robust to a change in the input mask distribution. When we compare the performance between the loose dilation masks (used for fine-tuning) and the scribbled masks, we see a significant drop across all metrics, except for the precision, which actually increases. When using only the mask as input, the performance is nearly cut in half. When combining the mask with other prompts, the drop is even more important, with Dice and IoU scores reduced by up to a factor of four.

An explanation is that the loose dilation masks cover both the target object and a large surrounding area, whereas the scribbled masks are very tight and restricted to the annotation itself. If the model tends to segment subregions within the input mask, the scribbled masks may lead to under-segmentation. To verify this hypothesis, I plotted examples (see Appendix B). These show both the predicted masks and the inputs. As expected, the predicted masks for scribbled inputs are much smaller and more limited in scope, which reinforces the idea that the model segments internal regions. This also explains the higher precision: because the scribbled masks are so tightly centered around the object, the pixels predicted by the model are more likely to belong to the actual ground truth.

Table 5.14: Results for Mask Distribution Shift: Mask only (SAM)

Metric	Loose Dilation	Scribble
Dice	**0.881**	0.482
IoU	**0.801**	0.349
Precision	0.885	**0.962**
Recall	**0.886**	0.350

Table 5.15: Results for Mask Distribution Shift: All prompts (SAM)

Metric	Loose Dilation	Scribble
Dice	**0.893**	0.404
IoU	**0.820**	0.267
Precision	0.892	**0.967**
Recall	**0.901**	0.267

Tables 5.16 and 5.17 display the same experiments for SAM2. The conclusions are nearly identical, but the drop in performance is even more important than before. In some cases, the Dice and IoU scores drop by nearly a factor of four. The precision also decreases slightly this time. While the relative drop is more consistent across both experiments (the drops for both are nearly equal), the results confirm that SAM2 is just as sensitive to mask distribution shifts. The qualitative results are also worse for SAM2. For example, in Figure B.3, the model barely segments a few pixels from the scribbled mask, and in Figure B.4, it does not segment anything, the predicted mask is completely empty.

Table 5.16: Results for Mask Distribution Shift: Mask only (SAM2)

Metric	Loose Dilation	Scribble
Dice	**0.897**	0.274
IoU	**0.825**	0.181
Precision	**0.900**	0.887
Recall	**0.902**	0.181

Table 5.17: Results for Mask Distribution Shift: All prompts (SAM2)

Metric	Loose Dilation	Scribble
Dice	**0.902**	0.252
IoU	**0.833**	0.160
Precision	**0.894**	0.891
Recall	**0.918**	0.160

To summarize, we see that both Segment Anything models are highly sensitive to the mask distribution on which they have been fine-tuned. Indeed, changing that distribution during inference leads to an important performance drop. Therefore, mask prompts seem unsuitable for use in Cytomine. Indeed, users may provide masks with highly variable characteristics which will most likely not correspond to the ones used during training. On the other hand, using bounding boxes and points as prompts seem much more adapted, even thought the performance were slightly lower, they present much less variability than mask prompts which will result in better generalization performance.

5.9 Conclusion

In this chapter, I analyzed the performance of both Segment Anything models when fine-tuned on histopathology data. I conducted experiments with different loss functions (Section 5.2), tested various dataset configurations (Section 5.3), analyzed different prompt type combinations (Section 5.4), and investigated the impact of fine-tuning different parts of the models (Section 5.5). After that, I evaluated several post-processing methods (Section 5.7) and assessed how well the models generalize when there is a shift in the input mask distribution (Section 5.8).

Overall, the experiments confirmed the conclusions drawn from the zero-shot evaluation: SAM2 remains the most promising model for interactive segmentation. It provides high quality user-friendly masks when combined with the post-processing method based on OpenCV. Finally, I showed that the combination of bounding boxes and point prompts is the most suitable for use in Cytomine. This is because the models are highly sensitive to the mask prompt distribution, and in a real-world interactive setting, the consistency of such prompts cannot be guaranteed.

Chapter 6

Model Extension with Specialized Encoders

6.1 Methodology

In this chapter, I follow a similar approach to the one proposed in [J. Zhang et al., 2023] and [M. Zhang et al., 2024], which consists in combining a domain-specific encoder with either SAM or SAM2 to improve the segmentation results. Indeed, this is one of the two most common ways used in the literature to enhance the performance (the other being parameter-efficient fine-tuning).

To this end, I investigated three popular histopathology encoders: H-optimus-0, UNI, and UNI-2H. H-optimus-0, developed by the French startup Bioptimus [Saillard et al., 2024], was at its release the largest open-source foundation model for pathology, with over 1.1 billion parameters and fine-tuned on more than 500,000 histopathology slides. UNI, published by Mahmood Lab [R. J. Chen et al., 2024], is another encoder trained on over 100,000 WSIs with around 303 million parameters. UNI-2H, also from Mahmood Lab [R. J. Chen et al., 2024], is a follow-up of UNI, leveraging a larger architecture (681 million parameters) and trained on a broader dataset.

Since all three encoders are based on a standard ViT architecture and do not provide multi-scale or hierarchical features, I chose to examine their combination only with SAM, whose image encoder is also a standard ViT. Indeed, combining these encoders with SAM2 would have required more adaptation, as SAM2 expects hierarchical features.

Throughout this chapter, I analyze several different configurations to combine the domain-specific encoders with SAM. First, I evaluate the performance when simply replacing SAM's image encoder with a domain-specific one (Section 6.2), to see whether using an encoder with domain knowledge alone improves the performance. After that, I test a dual-encoder setup, in which both the original SAM encoder and a domain-specific encoder are used together, and their outputs are combined together via concatenation (Section 6.3). Then, I look at another way to combine the encoders outputs by incorporating attention mechanisms (Section 6.4). Finally, to conclude the series of experiments on the Segment Anything models, I perform a comparative

evaluation of the best-performing models from this chapter and the previous one, to determine which one is the most suitable for integration into Cytomine (Section 6.5).

6.2 Domain-Only Encoder Approach

As previously mentioned, in this section I investigate whether replacing the original SAM image encoder with a domain-specific encoder can improve the performance.

A potential challenge/issue with this experiment is the difference in input resolution. Indeed, the original SAM image encoder operates at a resolution of 1024×1024, whereas all three domain-specific encoders (H-Optimus-0, UNI, and UNI-2H) work with images at 224×224, which is approximately four times smaller. This discrepancy may impact the performance, since the domain-specific encoders, while potentially being better at recognizing the histopathological features, may not have enough spatial detail which are necessary to generate accurate masks.

Figure 6.1: Domain-Only Encoder Architecture

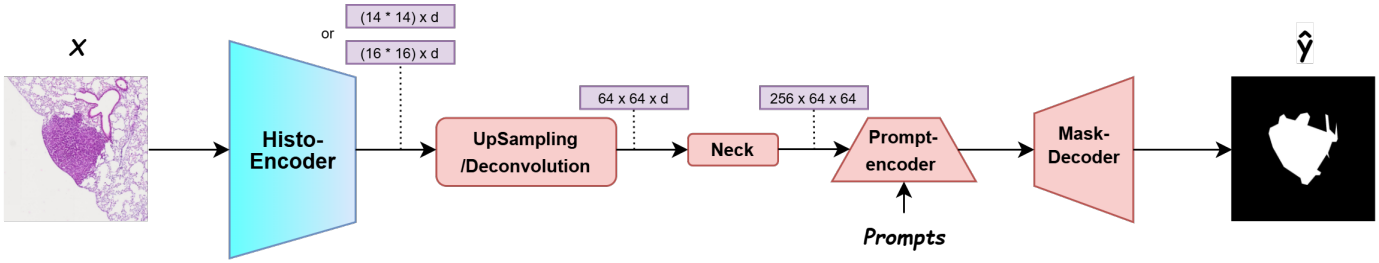


Figure 6.1 shows the architecture of the models that I fine-tuned using only the domain-specific encoders. In the figure, we can see that the input image (denoted by x) first passes through the domain-specific encoder, which can be either H-Optimus-0, UNI, or UNI-2H. For clarity, I refer to this module as the "Histo-encoder". After this step, the output goes through an additional module that I introduced, this module is needed because the resolution of the domain-specific encoders is four times lower than the one of SAM's original image encoder. Therefore, it is necessary to make the dimensions of the embeddings compatible with the rest of the SAM model.

Initially, SAM's image encoder produces a feature map of dimension [1] 256x64x64 when an image is passed through it. This is because the encoder takes images at a resolution of 1024x1024 and follows a ViT/16 architecture, which means that it works with 16x16 input patches. Thus, the image is divided into 64 patches along each of its dimension, which forms a 64x64 grid. For each patch, the encoder generates an embedding of size 768 (in the case of the ViT-B variant that I used). Therefore, this gives an initial representation of size 64x64x768. However, since the different encoder variants all have different embedding dimensions, SAM uses an extra module called the neck (that I also

[1] Dimensions are represented by purple boxes in the figure.

reused) to project those embeddings to size 256. This allows the rest of the network to stay the same, regardless of which encoder is used.

On the other hand, the Histo-encoders follow either a ViT/16 (UNI) or ViT/14 architecture (UNI-2H, and H-Optimus-0), all of them having a smaller input resolution of 224x224. Of course, this leads to smaller feature maps: either 16x16xd or 14x14xd, where d is the embedding dimension of the encoder [2]. To make those outputs compatible with the rest of the SAM architecture. I had to add an upsampling module that increases the spatial size to 64x64, and after that, the embedding dimension of those Histo-encoders is projected to 256 thanks to the neck module, so that the final representation also has shape 256x64x64.

I investigated two different upsampling modules. The first one uses transposed convolutions (thus, a deconvolution approach) to directly upsample the output of the Histo-encoder to the target dimension. This method is quite simple and direct but it tends to introduce a large number of parameters. Because of that, I also tested a second approach based on interpolation. In this method, I first resize the embeddings to the desired shape using standard bilinear interpolation. Then, I apply two convolutional layers: the first one performs a per-channel convolution, meaning that it processes each feature map independently to refine spatial details; the second applies a convolution across the channels for each pixel, to bring in some cross-channel context. Indeed, since the interpolation introduces many new values that were not part of the original embeddings, and that may not even carry some meaningful semantic information, these convolutions help to reorganize and refine the interpolated features to make them more useful and knowledgeable.

Thus, the main idea behind this second approach is to keep the upsampling process efficient by using interpolation, while still allowing the model to extract some useful representations from both the original and interpolated values through these two convolution steps. For example, in the case of H-Optimus-0, the deconvolution-based upsampling module had around 37 million parameters, while the interpolation-based module had only 2.38 million. This shows the advantage of the second method in terms of fine-tuning efficiency.

After the upsampling module, I kept the rest of the network identical to the original SAM architecture. Therefore, the next step is to pass through the neck, and then pass the prompts through the prompt encoder and finally, pass everything through the mask decoder, which produces the final mask prediction (denoted by $\hat{y}$). In the diagram, the modules shown in red are the ones that are fine-tuned during training, while those in light blue remain frozen.

Regarding the fine-tuning settings that I used in this experiment, they were mostly the same as those that gave the best results in the previous chapter. I used all prompt types, with the loose dilation mask type. To improve the training time efficiency, I also

[2]Note that the dimensions corresponding to the patches in the outputs of the encoders are flattened in Figure 6.1, and represented by two boxes since two different dimensions are possible.

precomputed all the image embeddings. For all the histopathology-specific encoders, I used a batch size of 32. The learning rate was kept at 1×10^{-5}, and I trained the models for 30 epochs, as before. For the loss function, we previously observed that the simpler BCE loss generally produced better results. However, since I changed the image encoder in this experiment, I could not assume this would still necessarily hold true. For that reason, in addition to testing the two upsampling modules, I also ran an experiment using the original SAM loss function, to verify whether the simpler loss still provided the best performance in this new setup.

6.2.1 Results

Tables 6.1, 6.2, and 6.3 present the results for H-Optimus-0, UNI, and UNI-2H, respectively. As before, Dice score, IoU, precision, and recall are displayed, and reported per dataset.

Table 6.1: Results for Encoder-Only (H-Optimus-0)

Setting	Metric	LBTD-AGDC10	LBTD-NEO04	Camelyon16	All
with Deconvolution	Dice	0.833	0.893	0.833	0.849
	IoU	0.739	0.825	0.731	0.757
	Precision	0.833	0.900	0.837	0.853
	Recall	0.853	0.905	0.850	0.865
with Interpolation	Dice	0.822	0.893	0.844	**0.852**
	IoU	0.728	0.823	0.746	**0.763**
	Precision	0.843	0.899	0.843	**0.858**
	Recall	0.832	0.903	0.860	**0.865**
with Loss (Interpolation)	Dice	0.802	0.875	0.809	0.825
	IoU	0.701	0.796	0.704	0.728
	Precision	0.875	0.931	0.900	0.903
	Recall	0.770	0.843	0.760	0.784

From Table 6.1, we can see that the upsampling module based on interpolation leads to the best performance. It achieves slightly higher Dice and IoU scores, by 0.003 and 0.006 respectively, compared to the deconvolution-based method. While this improvement is small, it still suggests a slight advantage, even though its consistency across the other encoders must be confirmed. We also observe that the simpler loss function once again performs better than the original SAM loss. To test this, I fine-tuned the model using the original loss in combination with the interpolation upsampling module (since it was the best of the two). The results show a performance drop, with Dice and IoU scores lower by approximately 0.2 to 0.3. This confirms the earlier result that the simpler loss is better, this time, even when the encoder is changed. Finally, we see that the overall performance with H-Optimus-0 is below the one of the original SAM image encoder. In Table 5.5, the best-performing SAM model achieved a Dice score of 0.897 and an IoU of 0.824. In comparison, the best configuration with H-Optimus-0 has a 0.045 lower Dice and a 0.061 lower IoU, which indicates a slight performance drop.

Table 6.2: Results for Encoder-Only (UNI)

Setting	Metric	LBTD-AGDC10	LBTD-NEO04	Camelyon16	All
with Deconvolution	Dice	0.805	0.879	0.826	0.835
	IoU	0.700	0.803	0.720	0.738
	Precision	0.826	0.893	0.827	0.844
	Recall	0.804	0.884	0.849	0.849
with Interpolation	Dice	0.819	0.881	0.836	**0.844**
	IoU	0.721	0.807	0.737	**0.752**
	Precision	0.837	0.900	0.853	**0.862**
	Recall	0.828	0.882	0.842	**0.850**
with Loss (Interpolation)	Dice	0.768	0.860	0.782	0.800
	IoU	0.661	0.779	0.672	0.698
	Precision	0.878	0.932	0.901	0.904
	Recall	0.712	0.820	0.722	0.746

From Table 6.2, we can draw similar conclusions. The interpolation-based upsampling module still gives the best results, and the performance gap compared to the deconvolution method is slightly larger than in the previous experiment. Once again, using the original SAM loss leads to lower performance, confirming the previous trend. Overall, the results obtained using the UNI encoder are lower than the ones achieved with H-Optimus-0. The best model using UNI shows a drop of about 0.1 in both Dice score and IoU compared to the best H-Optimus-0 configuration. This difference is likely due to the fact that H-Optimus-0 is a larger model and was trained on a wider dataset.

Table 6.3: Results for Encoder-Only (UNI-2H)

Setting	Metric	LBTD-AGDC10	LBTD-NEO04	Camelyon16	All
with Deconvolution	Dice	0.832	0.891	0.839	0.851
	IoU	0.738	0.822	0.738	0.760
	Precision	0.841	0.900	0.856	0.864
	Recall	0.845	0.903	0.844	0.859
with Interpolation	Dice	0.824	0.892	0.846	**0.853**
	IoU	0.727	0.820	0.748	**0.763**
	Precision	0.848	0.907	0.864	**0.872**
	Recall	0.826	0.891	0.846	**0.854**
with Loss (Interpolation)	Dice	0.788	0.871	0.803	0.818
	IoU	0.683	0.791	0.695	0.717
	Precision	0.893	0.941	0.912	0.916
	Recall	0.728	0.829	0.741	0.761

Finally, from Table 6.3, we can again see that the interpolation method outperforms the deconvolution-based upsampling module. As before, using the original loss function significantly reduces the performance. The results for UNI-2H are very similar to those of H-Optimus-0, indeed, the Dice score and the IoU are nearly identical. However, the configuration using UNI-2H has a higher precision and a lower recall compared to H-Optimus-0. This means that it predicts fewer pixels from the ground truth overall, but the ones it does predict are more accurate.

In conclusion, using only the domain-specific encoder results in lower performance compared to simply using the original SAM image encoder. For all three domain-specific encoders, we observed a performance drop of about 0.03 to 0.07 in both Dice score and IoU. This drop is likely due to the difference in resolution. Although the domain-specific encoders may better understand the content of the histopathology images, their lower spatial resolution leads to less detailed representations, which likely affects the segmentation quality.

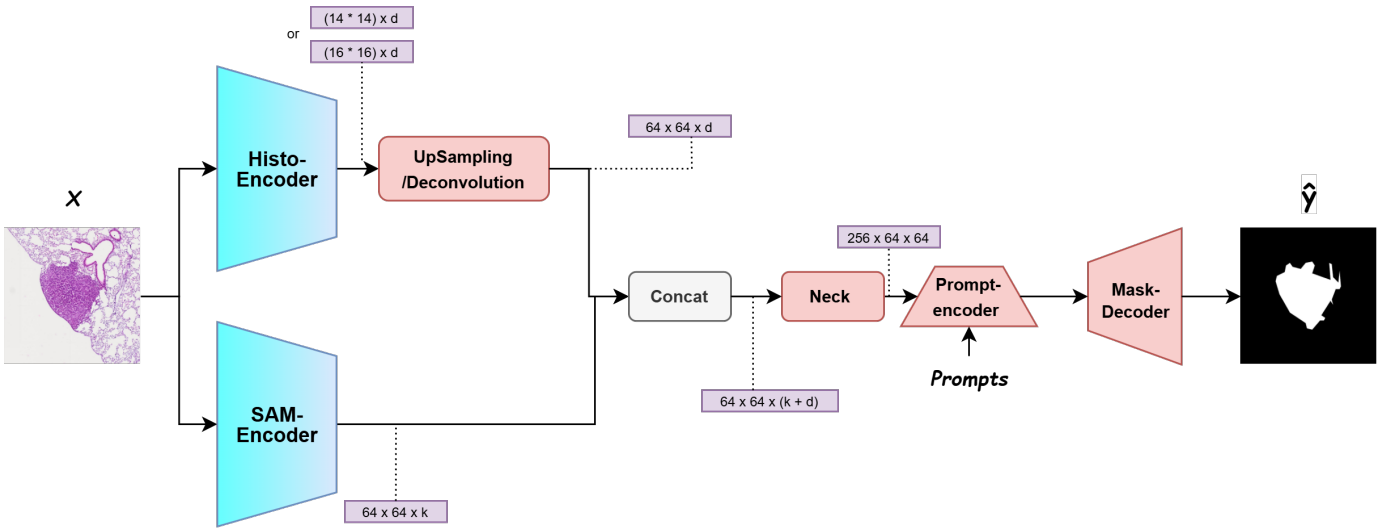
We also found that the interpolation-based upsampling module performed slightly better than the deconvolution-based one, while also having far fewer parameters to fine-tune. This may be because the deconvolution module, due to its larger number of parameters, is more prone to overfitting on small datasets. On the other hand, interpolation introduces no trainable parameters and reduces the risk of overfitting, while the two additional convolutional layers also help to refine the interpolated features.

Finally, we confirmed again the earlier result from Chapter 5 that the simpler loss function (BCEWithLogits) performs better than the original SAM loss. Among the three encoders tested, H-Optimus-0 and UNI-2H yielded better results than UNI, likely because they have more parameters and were trained on larger datasets.

6.3 Concatenation

In this section, I investigate whether combining the full SAM model with a domain-specific encoder can improve the segmentation performance compared to only using the domain-specific encoder alone.

Figure 6.2: Histo-Encoder with SAM Architecture



To do this, I follow the architecture shown in Figure 6.2. The input image (still denoted by x) is passed through both SAM's image encoder and the Histo-encoder in parallel. The resulting outputs from the Histo-encoder have shape (14*14)xd or (16*16)xd depending on the type of encoder, where d is the embedding dimension of that specific Histo-encoder. SAM's image encoder produces outputs of shape 64x64xk, with k being the embedding dimension of the SAM encoder variant that is used. As before, the output embeddings from the Histo-encoder are upsampled using the previously described upsampling modules. After this step, they have a shape 64x64xd, which matches the dimensions of the feature maps returned by SAM. I chose to test both methods again, since the deconvolution-based module, while more complex, might be better suited to this configuration because it could potentially learn richer features that complement the ones of the SAM encoder during concatenation.

Once the two sets of embeddings have compatible shapes, they are concatenated along the embedding dimension, resulting in embeddings of shape 64x64x(k + d), and passed through the neck module, which finally projects them to the final shape of 256x64x64 (the expected input dimensions of the rest of the model). The neck consists of convolutional and normalization layers. Here, I reused the exact same design and adapted it to match the dimensions of the combined embeddings.

After these few steps, the rest of the pipeline remains unchanged: the prompts are passed through the prompt encoder, and the outputs are then processed by the mask decoder. As shown in the figure, the light blue modules are frozen during training, while the red ones are fine-tuned. Note that the concatenation step itself does not involve any trainable parameters, so it is not fine-tuned.

6.3.1 Results

The fine-tuning settings for these experiments are identical to those that I used previously. However, unlike the previous section (Section 6.2), I did not repeat the comparison of loss functions here, as the previous results had already confirmed that the simpler BCE-based loss still performed better. The results of these experiments are presented in Tables 6.4, 6.5, and 6.6, using the same format as in the previous results.

Table 6.4: Results for Concatenation (H-Optimus-0)

Setting	Metric	LBTD-AGDC10	LBTD-NEO04	Camelyon16	All
with Deconvolution	Dice	0.844	0.899	0.853	**0.863**
	IoU	0.754	0.832	0.759	**0.777**
	Precision	0.826	0.900	0.854	**0.860**
	Recall	0.886	0.914	0.869	**0.884**
with Interpolation	Dice	0.837	0.889	0.842	0.853
	IoU	0.745	0.820	0.746	0.765
	Precision	0.851	0.905	0.855	0.867
	Recall	0.847	0.895	0.852	0.862

From Table 6.4, we see that, unlike in the previous section (Section 6.2), the upsampling module based on deconvolution gives the best results, and the difference is not negligible: around 0.01 higher for both the Dice score and the IoU. This is somewhat surprising but could potentially be explained by my earlier hypothesis that the deconvolution module, being more expressive, helps to better align the embeddings produced by the two different image encoders. Another observation is that the scores in this table are still lower than those obtained in Table 5.5. More particularly, the Dice score is about 0.03 lower, and the IoU is about 0.05 lower. This suggests that using only SAM's encoder gives better performance than combining it with a domain-specific encoder, which is a bit unexpected.

Table 6.5: Results for Concatenation (UNI)

Setting	Metric	LBTD-AGDC10	LBTD-NEO04	Camelyon16	All
with Deconvolution	Dice	0.831	0.893	0.844	0.854
	IoU	0.736	0.822	0.746	0.764
	Precision	0.825	0.891	0.835	0.847
	Recall	0.861	0.911	0.876	0.882
with Interpolation	Dice	0.841	0.897	0.845	**0.858**
	IoU	0.750	0.828	0.749	**0.770**
	Precision	0.832	0.895	0.844	**0.855**
	Recall	0.872	0.917	0.866	**0.880**

From Table 6.5, we see a different result: this time, the interpolation-based upsampling module performs better, though only by a small margin. This shows that the earlier result where deconvolution was better does not hold for all encoders. One possible reason is that the embeddings from UNI are smaller than those from H-Optimus-0, thus, the added complexity of the deconvolution module might not provide any extra benefit. However, the combined performance of UNI and SAM is still lower than the best SAM-only model, and even slightly worse than the results obtained with H-Optimus-0, confirming the trend observed in the previous section (Section 6.2).

Table 6.6: Results for Concatenation (UNI-2H)

Setting	Metric	LBTD-AGDC10	LBTD-NEO04	Camelyon16	All
with Deconvolution	Dice	0.839	0.894	0.847	**0.858**
	IoU	0.746	0.825	0.751	**0.770**
	Precision	0.849	0.898	0.861	**0.868**
	Recall	0.848	0.909	0.855	**0.868**
with Interpolation	Dice	0.837	0.889	0.842	0.853
	IoU	0.745	0.820	0.746	0.765
	Precision	0.851	0.905	0.855	0.867
	Recall	0.847	0.895	0.852	0.862

Lastly, in Table 6.6, we again see the same pattern as in Table 6.4, with the deconvolution-based upsampling module slightly outperforming the interpolation-based one. This suggests that when combining a domain-specific encoder with SAM's encoder, the deconvolution-based module tends to give slightly better results overall. However, compared to the previous section (Section 6.2), the performance of UNI-2H is slightly lower here, compared to H-Optimus-0, with a Dice score reduced by 0.005 and an IoU score reduced by 0.007.

To summarize, we see that among the three domain-specific encoders, H-Optimus-0 achieves the best performance, although only by a small margin. Moreover, the trend that we observed in the previous section (Section 6.2) regarding the upsampling modules does not hold here. In this setup, the deconvolution-based module generally performs better than the interpolation-based one. As previously mentioned, this might be linked to the larger number of parameters in the deconvolution module, which could give it greater expressiveness and thus a better ability to align the feature representations. More importantly, we observe that combining the SAM model with any of the three Histo-encoders through simple concatenation results in worse performance than fine-tuning the standard SAM model alone. This is somewhat surprising. Intuitively, one might expect that integrating domain-specific encoders, which are designed to bring additional knowledge about histopathological objects, would enhance the performance. This was, in fact, the case in studies such as [J. Zhang et al., 2023] and [M. Zhang et al., 2024]. There are a few potential explanations for this outcome. First, the spatial feature maps produced by the two encoders may not be perfectly aligned. Indeed, although the features from the Histo-encoder are upsampled to match the expected resolution, this transformation might not preserve the original spatial correspondence with the SAM encoder's features. Such a misalignment could worsen the model's ability

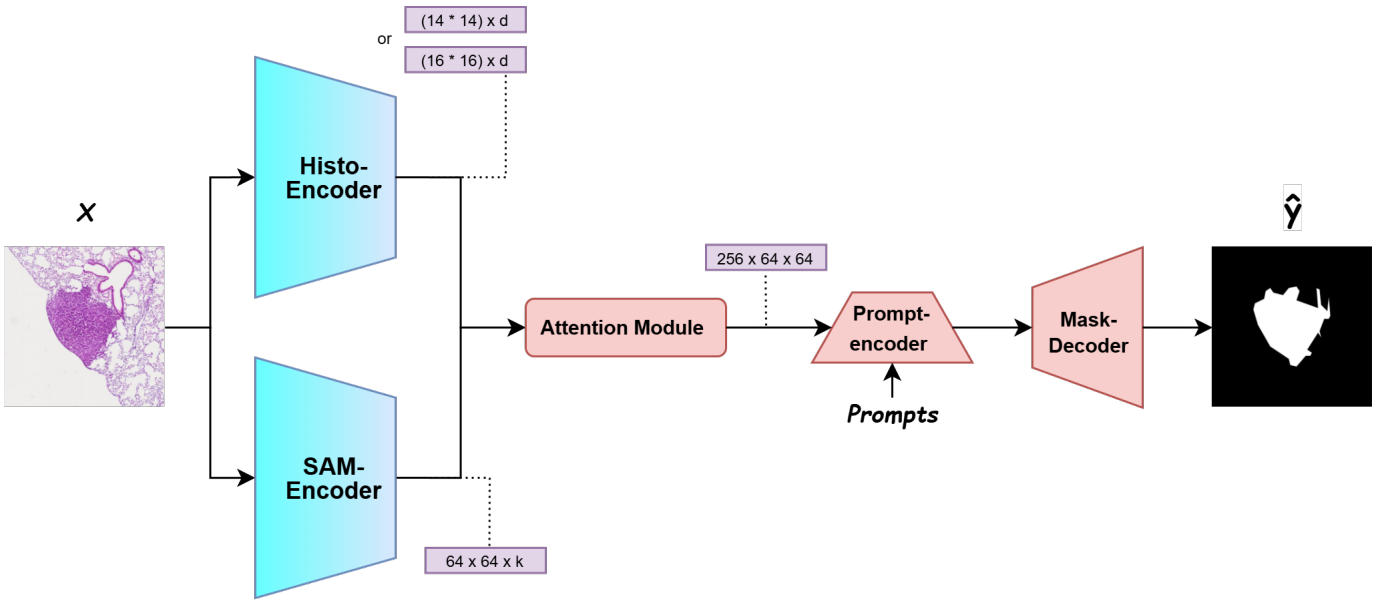
to combine the two representations effectively. Another possible explanation is that a simple concatenation followed by a processing through the neck module may not be sufficient to fuse the information meaningfully. In fact, mixing the interpolated and potentially noisy features with more reliable ones might introduce some inconsistencies, leading to a loss of useful signal and, thus, degraded performance.

6.4 Cross-Attention Fusion

Given the weaker performance of the architecture based on concatenation compared to the best-performing SAM model from Chapter 5, I decided to examine whether using attention mechanisms instead of simple concatenation could improve the overall performance of the combined SAM and Histo-encoder models.

To do so, I investigated two attention-based methods for combining the domain-specific encoders with the original SAM model. These two approaches are illustrated in Figure 6.3 and Figure 6.4.

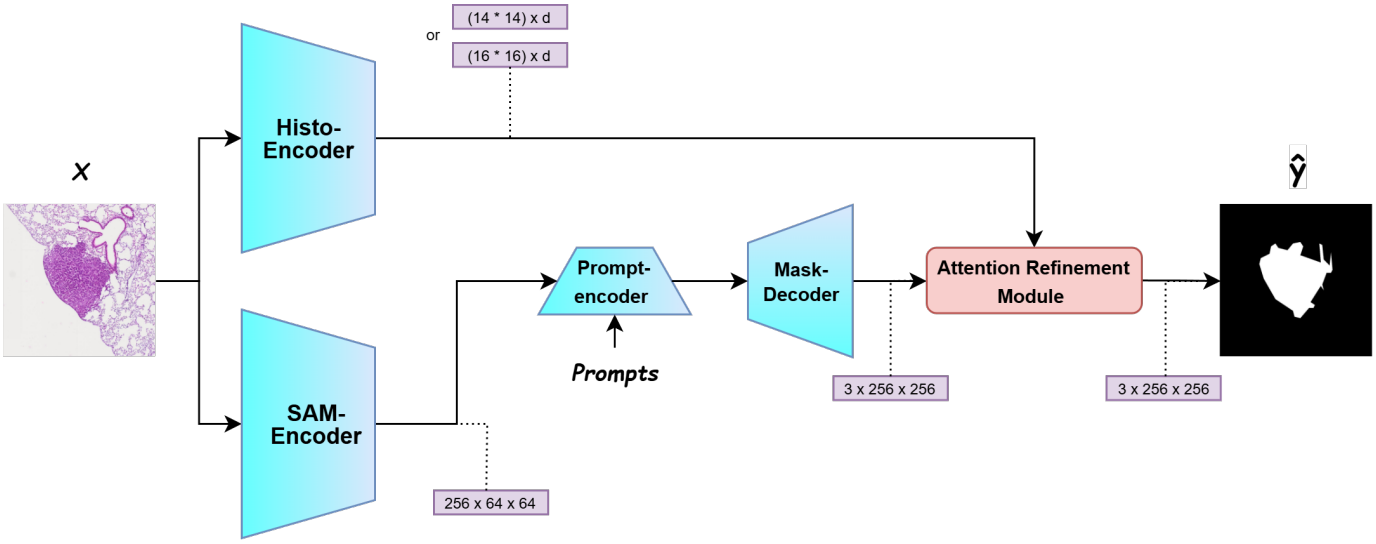
Figure 6.3: Histo-Encoder with SAM Architecture (Attention Module)



The first approach is quite straightforward. Instead of upsampling the features produced by the Histo-encoders to match the ones of the SAM embeddings, which could potentially introduce artifacts or values that are not semantically meaningful, I applied a cross-attention mechanism directly, as proposed in [Bahdanau et al., 2016]. In this setup, the feature maps from SAM are used as queries, while the features from the Histo-encoder are used as the context/values. In the attention module, the features returned by SAM's image encoder are first flattened across the first two dimensions to obtain a shape of (64*64)xk, matching the format of the Histo-encoder output. After that, they are both projected to an embedding dimension of 256 (to later have an output compatible with the rest of the network), resulting in (64*64)x256 and (14*14)x256 or (16*16)x256,

respectively for SAM and for the Histo-encoder. Then, all of these are passed through a multi-head attention module. This allows the SAM features to attend directly to those from the Histo-encoder, without introducing some artificial values through upsampling. In that aspect, this approach is more coherent and avoids the noise introduced by the interpolation or the deconvolution. Moreover, we know that attention mechanisms tend to help models capture relevant dependencies and enhance the performance by enabling more flexible feature processing. For the attention module, I used eight attention heads. The rest of the architecture remained unchanged. As before, only the attention module, the prompt encoder, and the mask decoder were fine-tuned, while the image encoders were both kept frozen.

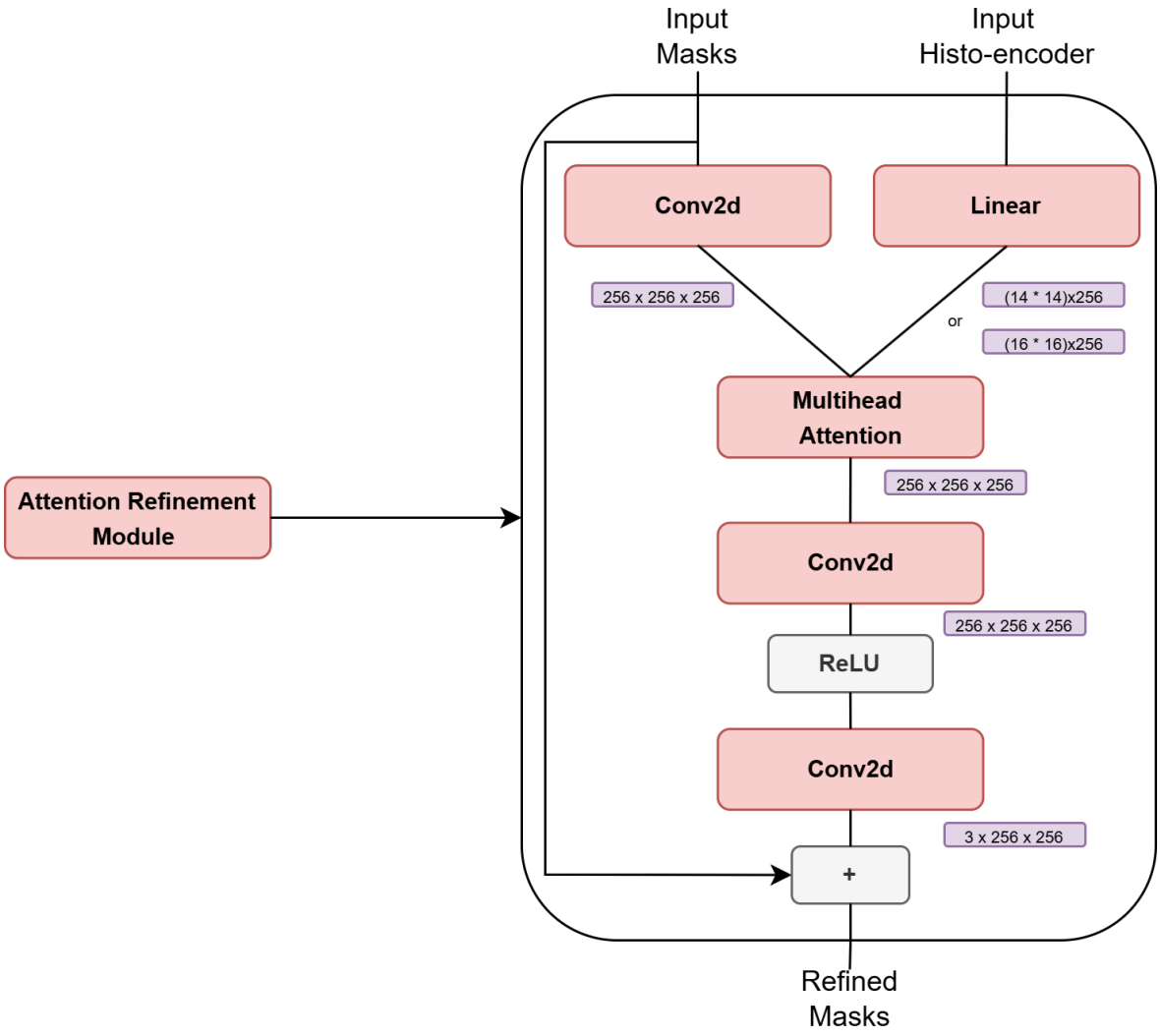
Figure 6.4: Histo-Encoder with SAM Architecture (Attention Refinement Module)



The second method follows a different philosophy, but is still simple. The idea is that we already have a well-performing configuration for SAM, as shown in Table 5.5, so retraining the entire model from scratch might not be necessary. Instead, we could leverage this already fine-tuned model, and analyze whether the Histo-encoders can be used to further refine its predictions.

In this setup, I thus reuse the weights of the best SAM model and keep them frozen, since this configuration already performs well. Then, the input image is processed normally through SAM, and the low-resolution mask predictions from the mask decoder are passed to an attention refinement module. This module also receives the image embeddings from the Histo-encoder. Basically, this approach acts like a post-processing step, where the initial SAM predictions are refined using the domain-specific features provided by the Histo-encoders. This process allows to inject some domain knowledge at the refinement stage, which likely mitigates the potential alignment issues that could be encountered, since it avoids directly mixing the internal features of the two encoders.

Figure 6.5: Attention Refinement Module Architecture



For the attention refinement module in itself: it takes the low-resolution masks as queries, and the Histo-encoder image embeddings as context/values. The low-resolution masks have an initial shape of 3x256x256, while the embeddings from the Histo-encoders still have the same shape as before. In the module, the masks are first passed through a convolutional layer to project them into a format compatible with the multi-head attention module; they are projected to a shape of 256x256x256. Simultaneously, the Histo-encoder embeddings are passed through a linear projection for the same reason, resulting in a shape of (14 * 14)x256 or (16 * 16)x256 (again depending on the specific Histo-encoder that is used). These projected inputs are then processed by the multi-head attention module that produces an output of shape 256x256x256. After the attention block, I included two convolutional layers that further process the output and produce a tensor of shape 3x256x256, compatible with the original low-resolution masks. This output is then added to the initial low-resolution masks through a skip connection, allowing for a light correction rather than a full replacement of the predictions. Indeed, the idea behind the whole module is to refine the masks, not to replace them

entirely. Finally, after the refinement module, the refined low-resolution masks are upsampled to the final resolution (1024×1024), similarly to the standard SAM processing.

6.4.1 Results

Regarding the experiments, I reused exactly the same fine-tuning settings as before. This means I kept the simpler loss function, the same batch size, learning rate, and all other training parameters unchanged. For each Histo-encoder, I fine-tuned a model using both attention-based methods, resulting in six fine-tuned models in total. The results of these experiments are presented in Table 6.7 and Table 6.8, which correspond respectively to the method using attention to fuse the two encoder embeddings, and the one using attention as a post-processing refinement step.

Table 6.7: Results for SAM with Attention Fusing

Model	Metric	LBTD-AGDC10	LBTD-NEO04	Camelyon16	All
H-Optimus-0	Dice	0.831	0.886	0.842	0.851
	IoU	0.735	0.813	0.742	0.759
	Precision	0.850	0.896	0.853	0.863
	Recall	0.833	0.898	0.850	0.859
UNI	Dice	0.836	0.890	0.842	0.853
	IoU	0.741	0.816	0.742	0.762
	Precision	0.832	0.896	0.852	0.859
	Recall	0.861	0.899	0.852	0.866
UNI-2H	Dice	0.846	0.890	0.846	**0.857**
	IoU	0.754	0.817	0.750	**0.769**
	Precision	0.844	0.893	0.857	**0.864**
	Recall	0.861	0.905	0.857	**0.870**

From Table 6.7, we can observe that the best performance is achieved when using UNI-2H as the Histo-encoder, although the margin is very small. Surprisingly, this time, the UNI encoder performs slightly better than H-Optimus-0, which was not the case in the previous experiments. However, even the best results in this table remain lower than the ones obtained with the best-performing SAM model from earlier experiments. This suggests that fusing the two encoder embeddings with attention did not bring any meaningful improvement. In fact, the results are even slightly worse than some obtained with the concatenation method. For example, the best model from Table 6.4 achieved both a higher Dice score and a better IoU.

Table 6.8: Results for SAM with Attention Refinement

Model	Metric	LBTD-AGDC10	LBTD-NEO04	Camelyon16	All
H-Optimus-0	Dice	0.864	0.924	0.898	0.898
	IoU	0.782	0.865	0.823	0.825
	Precision	0.868	0.926	0.903	0.901
	Recall	0.874	0.929	0.901	0.902
UNI	Dice	0.864	0.924	0.898	0.897
	IoU	0.781	0.865	0.823	0.825
	Precision	0.867	0.925	0.899	0.899
	Recall	0.874	0.930	0.904	0.904
UNI-2H	Dice	0.864	0.924	0.898	**0.898**
	IoU	0.781	0.865	0.823	**0.825**
	Precision	0.870	0.925	0.903	**0.901**
	Recall	0.872	0.930	0.901	**0.902**

Table 6.8 shows that the second method, using attention for mask refinement, leads to slightly better results. For all three Histo-encoders, this approach yields an improved performance compared to the best fine-tuned SAM model, even though by only a few thousandths. This result is not too surprising, since the method builds directly on the best-performing SAM model by keeping all its weights frozen and refining its output. In this setup, the attention has helped to slightly improve the performance. It is also the first method using a domain-specific encoder that actually improves the results.

In conclusion, these experiments showed that the first method, combining the image embeddings from the two encoders using an attention module, is not effective. In fact, it can perform worse than both the concatenation approach and when only the domain-specific encoder is used. This suggests that this method does not provide any real benefit, and in some cases, even removing SAM's image encoder could lead to better results. On the other hand, the second method, which uses attention as a refinement module, gives the best results among the four strategies that I tested. It is actually the only approach that slightly improves performance compared to the best fine-tuned SAM model, although the improvement remains very small.

Overall, this makes the refinement approach the most effective one, followed by the concatenation, then the first attention method, and finally the domain-specific encoder alone. However, another important point to consider, and that I will discuss further in the next section (Section 6.5), is the efficiency in terms of time and computational resources. Indeed, adding a second domain-specific encoder, as shown in the architecture in Figure 6.4, introduces a large number of additional parameters (for example, 1.1 billion in the case of H-Optimus-0), which increases the model complexity and slows down the predictions. This is a significant trade-off for what is in the end a negligible performance gain. Therefore, such a setup may not be really suitable, especially given the final goal of integrating a fast model within Cytomine. In fact, the post-processing method discussed in Chapter 5 would likely reduce or even eliminate the visible differences between this approach and the best-performing SAM model.

6.5 Final Evaluation

In the final section of this chapter, I perform one last evaluation of the best models that I have identified so far. Indeed, throughout this chapter and the previous one, I examined various configurations/settings that have given me strong models. This final experiment aims to compare them directly on a new dataset to assess their generalization capabilities and to choose the most suitable model for integration into Cytomine. For this, I used the CHU-ANAPATH dataset, that I previously introduced in Chapter 3, and evaluated all models using all prompt types.

I selected five models for this evaluation. The first two are the best-performing models from Chapter 5 for SAM and SAM2, both fine-tuned using all prompt types. After that, I included the model using the UNI-2H encoder from the "Domain-Only Encoder Approach" section of this chapter (Section 6.2). While this model was not the top performer overall, it tied with H-Optimus-0 in that section and was included to see if it might generalize better to unseen data. The fourth model is the combination of SAM with H-Optimus-0 using the deconvolution-based upsampling module. This was the best-performing model from the concatenation strategy and outperformed the attention-based fusion models (this is why no model from that approach was included in this evaluation). Finally, I selected the combination of SAM and UNI-2H using the attention refinement module. This was the overall best-performing model among those using domain-specific encoders, and even though its performance was extremely close to the best SAM model from Chapter 5, it still showed a slight improvement.

Table 6.9: Results for the final evaluation of the best models

Model	Metric	CHU-ANAPATH
SAM	Dice	0.875
	IoU	0.785
	Precision	0.851
	Recall	0.913
SAM2	Dice	**0.886**
	IoU	**0.803**
	Precision	**0.861**
	Recall	**0.925**
Uni-2H	Dice	0.825
	IoU	0.715
	Precision	0.787
	Recall	0.896
SAM with H-Optimus-0 (Concat)	Dice	0.835
	IoU	0.729
	Precision	0.799
	Recall	0.901
SAM with UNI-2H (Ref)	Dice	0.875
	IoU	0.785
	Precision	0.855
	Recall	0.910

Table 6.9 presents the results of this final experiment. From this table, we can see that the performance of all models has slightly decreased. For example, the SAM2 model originally had a Dice score of 0.906 and an IoU of 0.837, which have now dropped by 0.02 and 0.034, respectively. However, this drop remains very small, and all models still achieve high scores, which shows that they generalize well. Moreover, the performance ranking between models has remained consistent. SAM2 is still the best-performing model, followed by both the standard SAM and the SAM + UNI-2H (Ref) model. These two models were already very close in the previous experiments, and they remain so now. After them, we find the concatenation-based approach, followed by the domain-only UNI-2H model, which is last. Overall, these results are reassuring, as they confirm that all models generalize well to a new dataset and that the previous performance trends still hold.

To conclude this chapter and move forward with the integration into Cytomine, I chose the fine-tuned SAM2 model for the deployment. My decision is based on two main factors. Firstly, the fine-tuned SAM2 remains the top-performing model in terms of segmentation quality, making it the most logical choice from a performance perspective. Secondly, it is also the most efficient in terms of inference speed, being approximately twice as fast as the corresponding SAM model. This makes it particularly suitable for interactive use within Cytomine. Overall, the experiments in this chapter showed that adding an extra domain-specific encoder to SAM did not improve significantly the results. In fact, only one approach based on post-processing offered a very slight gain. This is somewhat surprising, but it can potentially be explained by the reasons and hypotheses that I discussed earlier. Since the performance gains are minimal and come at the cost of significantly higher computational requirements, using additional encoders does not seem wise. Indeed, these encoders add millions, or even billions of parameters which considerably increases the inference time. Therefore, given that they do not outperform the simpler and faster SAM2 model, I chose not to use these architectures.

Chapter 7

Integration into Cytomine

7.1 Methodology

In this chapter, I explain how I integrated SAM2 into a test version of the Cytomine platform.

I begin by describing the Cytomine architecture to give a better understanding of how the system works and why I had to modify certain parts of the code (Section 7.2). Then, I go into more detail about the integration process itself, describing the steps that I followed to make the new model functional within the platform (Section 7.3). Finally, I include a short tutorial in the Appendix that summarizes the key steps for integrating a new service or API into Cytomine. This tutorial is intended to help other developers/users to contribute more easily to the platform.

7.2 Platform Description

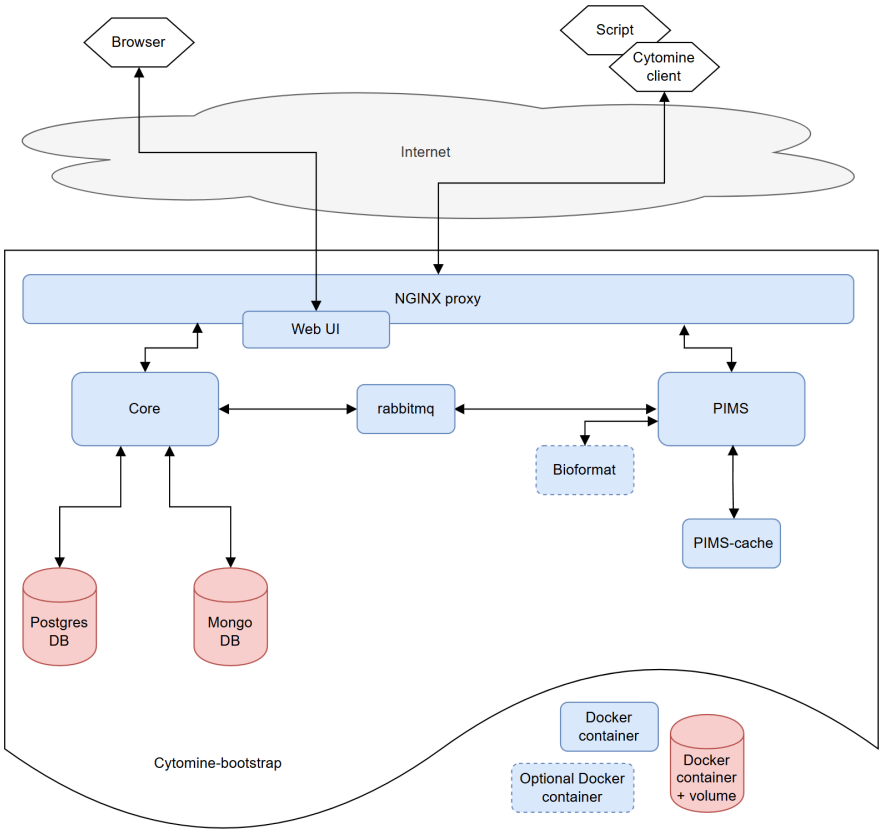
The Cytomine platform includes several editions, such as legacy versions, the BigPicture edition, and others. The integration that I carried out targeted the most recent version of the Community Edition, which is the main and standard version of Cytomine.

Regarding the system's architecture, Cytomine is built as a collection of Docker containers. Each container corresponds to a specific service within the platform, and these services communicate with one another. The overall architecture of the platform is illustrated in Figure 7.1 (taken from [Cytomine ULiège R&D Team, 2023]). Figure 7.2, also from [Cytomine ULiège R&D Team, 2023], gives a more detailed view of the different Cytomine services and their roles within the system.

We can see that the architecture follows a fairly standard structure for applications of this kind. Indeed, Cytomine adopts a modular, microservice-oriented approach. It includes a front-end/web UI, which is the interface that users interact with directly through their browser. Moreover, the platform offers several client libraries in different programming languages such as Python and JavaScript, which allow users to interact with the system through scripts.

A Nginx HTTP proxy is used to correctly route the incoming requests to the appropriate services. The Core service is the central component of the application, it handles all the API requests and delegates tasks to the appropriate services. It acts like the "brain" of the system. Even though the other parts of the architecture are not directly involved in the integration of SAM2, I briefly describe them here for completeness. Cytomine relies on two databases: a PostgreSQL database and a MongoDB instance. It uses PIMS as the main image server. Communication between the Core and the image server is handled via RabbitMQ.

Figure 7.1: Cytomine Community Edition (CE) Architecture



To decide how I would structure the integration of SAM2 into Cytomine, I took inspiration from a previously integrated service also based on deep learning. More specifically, a content-based image retrieval (CBIR) algorithm had already been successfully incorporated into Cytomine [Cytomine ULiège R&D Team, 2024]. I chose to follow a similar approach. The idea was to remain consistent with Cytomine's microservice architecture by creating a new microservice, implemented as a separate Docker container, to handle the new functionality/API.

For the CBIR algorithm, the developer, Mr. Ba Thien Le, has used Python, which is well suited for deep learning due to the availability of libraries such as Pytorch, and Tensorflow. He also used FastAPI to build the API, as it can efficiently handles large volumes of HTTP requests. I thus decided to adopt the same stack for my integration.

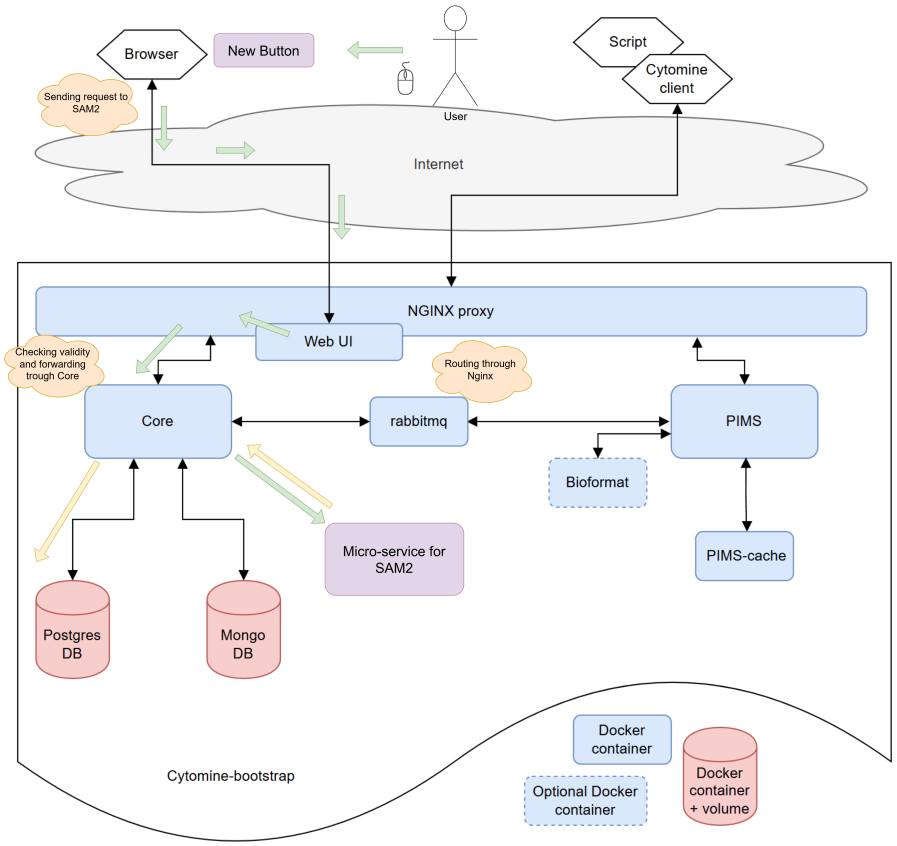
More practically, the integration that I had to perform involved modifying four main components. First, I needed to make it possible to call my API/service from the web-UI, allowing the users to invoke the model. Then, I had to properly route this request through Nginx so that it reached the Core container. Inside the Core, I had to handle the request and forward it to the appropriate service. Finally, I developed the new service as a standalone API, implemented in Python using FastAPI, and containerized with Docker. Thus, to carry this out, I relied on several technologies: Docker, Nginx, Python with FastAPI, Vue.js, and Spring Boot. Indeed, the Core of Cytomine is written in Java using Spring Boot, while the web-UI is built with Vue.js.

Figure 7.2: Cytomine Services

Name	Required	Goal
Nginx	Yes	Main HTTP(S) proxy dispatching incoming requests.
Core	Yes	Main Cytomine server. Provide the REST API.
Postgres	Yes	Main database. Store most of data.
MongoDB	Yes	Secondary database. Store activity data.
PIMS	Yes	Main image server.
PIMS-cache	Yes	Fast cache for images.
Web UI	No	Web graphical user interface.
App Engine	No	Service to execute containerised apps.

In the next section, I describe the integration process in more detail and present the final result. Nevertheless, to give a more intuitive understanding of how the integration fits within the current Cytomine architecture, I include Figure 7.3, which illustrates roughly what the modified Cytomine architecture will look like once the API is fully integrated. This figure gives a general overview of how the new API will be used after integration. First, the user will click on a new button in the webUI (new elements in the architecture are shown in purple) from their browser, to trigger the model. This sends a request to the API running the model. The forwarding of this request is represented by green arrows in the figure, while the associated step descriptions are shown in the orange clouds. The request is then routed to the Core service through Nginx. In the Core service, several validity checks are performed before eventually forwarding the request to the API. Finally, the API processes the request and updates the relevant data in the database via the Core service.

Figure 7.3: Cytomine Community Edition (CE) Architecture with SAM2 API



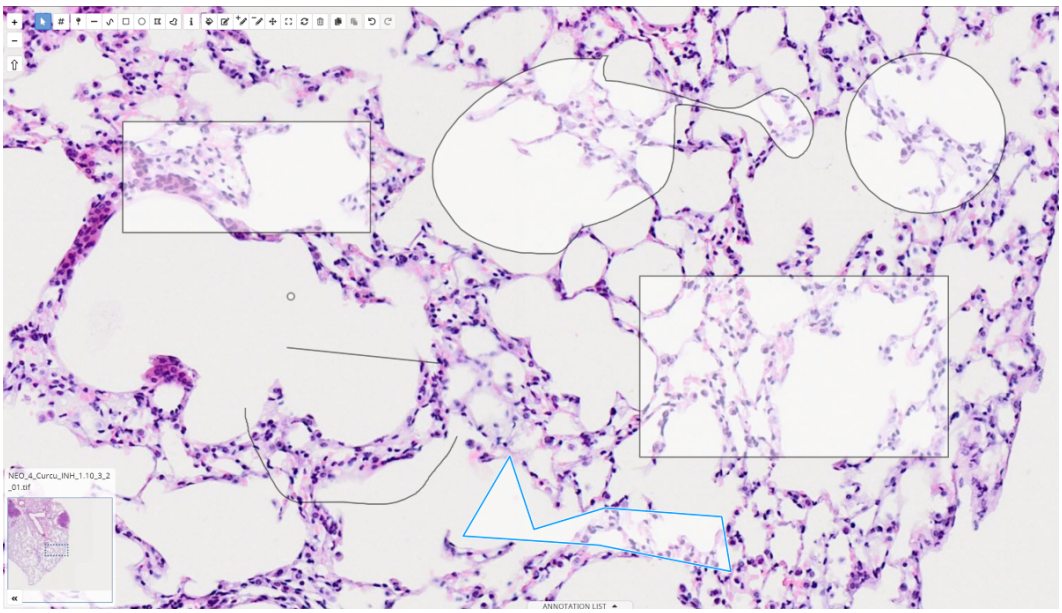
7.3 Integration Process

Before describing the step-by-step process that I followed to integrate SAM2 into Cytomine, I first describe what I actually integrated into the application. As mentioned, in the conclusion of the previous chapter, I selected the best-performing SAM2 model that I fine-tuned in Chapter 5, which uses the Hiera-Base-plus encoder. This model was trained on all three datasets used in that chapter, along with all prompt types (see Table 5.6). Alongside this model, I also integrated the post-processing method based on OpenCV, as it was also the best-performing post-processing technique.

I will now explain the underlying idea behind this integration, specifically regarding the prompts. As we know, SAM2 relies on prompts to guide the segmentation. However, Cytomine does not natively support the concept of prompts in the same way. Instead, all user interactions in Cytomine, such as adding a point, drawing a bounding box, or creating a mask are used to generate annotations. All these tools are designed to annotate, not to provide prompts in a deep learning context. Figure 7.4 shows annotations generated using the different tools; each interaction has created a new annotation.

With that in mind, my approach was the following: rather than asking users to give prompts explicitly, I would let them create annotations as usual using the existing tools, and then make it possible for them to refine these annotations using SAM2. For example, a user can draw a bounding box, then click on a "Refine" button, which will trigger the model to process this box as a prompt and to return a refined mask (as represented in Figure 7.3). In this setting, the annotation becomes the prompt.

Figure 7.4: Cytomine Annotation Examples



However, as discussed earlier in this thesis, more specifically in the Chapter 5, not all types of prompts are appropriate for SAM2. We saw that mask-based prompts performed poorly when their distribution was shifted from the one used during the training. Therefore, I chose to use the combination of bounding boxes and point prompts, which proved to be more reliable.

Apart from this decision, several other technical constraints had to be taken into account. Firstly, the model needs to always have a bounding box to correctly extract an image window around the region of interest. This is because, as explained in Section 3.1.3, the training and inference were always done by selecting image windows around the annotations, using a dezooming factor to ensure that the region was large enough. Thus, without a bounding box, we would not be able to extract such windows properly (since points alone do no provide any information about the size of the region to segment), and we might alter the input distribution which could significantly degrade the model performance. Therefore, a bounding box is always required for the model to work properly.

This constraint, always requiring a bounding box, might seem restrictive if we implement it naively by forcing the user to use the "Rectangle" tool. Indeed, a user may prefer using other tools such as polygons or freehand shapes, or may already have annotations created in the past using those shapes. Thus, forcing the user to redraw everything as bounding boxes would be impractical. To address this issue, if a user selects an annotation that is not a point and has a valid shape (i.e., a polygon with non-zero area), I extract the bounding box of that shape and use it as the model prompt. I do not use the detailed shape itself as input, as this corresponds to using mask prompts.

In short, the refinement using SAM2 is possible for all types of annotations except for the points, lines, and freehand lines. When necessary, a bounding box is extracted internally and used as the prompt.

I will now describe step by step the process that I followed to integrate SAM2 into Cytomine.

The first step was to modify the web-UI to include a new button allowing users to call the SAM2 API. I kept this implementation simple. I just added the button to the annotation detail menu, which already includes several other buttons (such as the one for centering the view on the selected annotation). I duplicated this centering button and updated it so that it sends an HTTP request to the URL of the new API. I also added basic notifications to inform the user whether the refinement using SAM2 was successful or not. This button is available only for annotations that are either bounding boxes or valid polygons, but not for points or lines. As explained earlier, those types do not provide enough information about the size of the object to be segmented. Figure 7.5 shows the updated annotation detail menu.

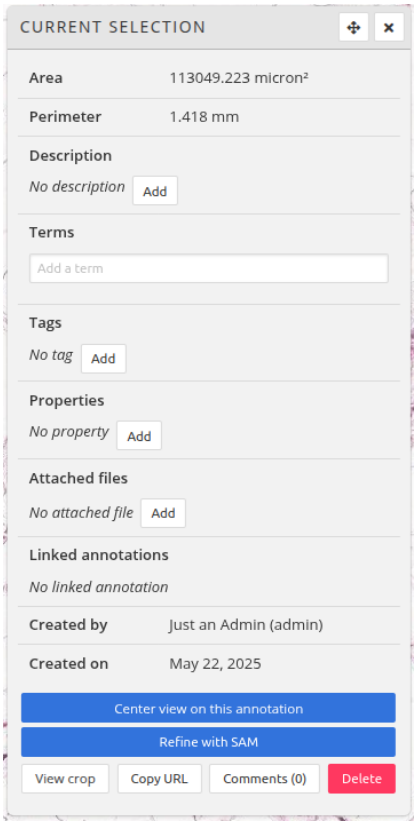
After that, I modified the Nginx configuration to add a routing rule for the new API. This ensures that the request is correctly directed to the Docker container running the API, and that the container can be reached by the rest of the application.

Regarding the Core service, I did minimal changes. The Core simply checks whether the user requesting the refinement has the appropriate authorization, and then forwards the request to the API, which handles the rest.

Now, regarding the main part of the code, the API itself, I developed a complete pipeline that handles the full process and directly modifies the annotation in the database. The API takes the ID of the annotation as input, since this is sufficient to identify any annotation. Based on this ID, it retrieves the complete annotation from the database to gather all the necessary information. It then checks whether the annotation is valid, meaning that it is not a point, a line, or a freehand line.

After that, it extracts a bounding box from the annotation using the Shapely library [Gillies et al., 2025]. If the annotation is already a box, the result is the same. If it is any other valid shape, the API computes its bounding box.

Figure 7.5: Annotation Detail Menu



Then, I needed a way to allow users to provide additional point prompts alongside the bounding box. To do this, I used a straightforward approach. I focused only on the combination of bounding boxes and positive point prompts. Indeed, in Chapters 4 and 5, we saw that negative point prompts did not really improve the performance when used in addition to bounding boxes and positive points, as shown in Table 5.5. Moreover, integrating negative prompts in Cytomine is not straightforward, as the platform does not natively distinguish between positive and negative points. One possible solution could be to use terms (which can be seen as keywords associated with annotations, directly supported by Cytomine) and to ask the users to assign a placeholder term (such as "positive", or "negative") to differentiate between the two types of point prompts. Nevertheless, I considered this approach to be impractical and not particularly user-friendly. Indeed, asking users to assign a term to each of their point prompts introduces some extra interaction, at each step, without really offering a meaningful gain in performance. In fact, the performance difference is just of a few thousandths (even one thousandth for the Dice score), as shown in Table 5.5, and would likely be imperceptible to users. In addition, this small difference would probably be completely eliminated after applying the post-processing. So, sticking with only positive prompts made the most sense. However, because of how Cytomine works, when a user adds a point, even if it lies inside a bounding box, it creates a separate annotation. To support the prompting combination, I chose to interpret all point annotations created by the same user inside the bounding box as additional positive point prompts. This provides a simple and intuitive way for users to add prompts using

the existing Cytomine interface. Therefore, once the bounding box is extracted, the API checks if the user has also created any point annotations inside that box. If so, these points are included as additional positive prompts. I also decided to automatically delete those point annotations after the mask has been refined, so that the users do not need to remove them manually. However, I left the option to keep them if desired in the API.

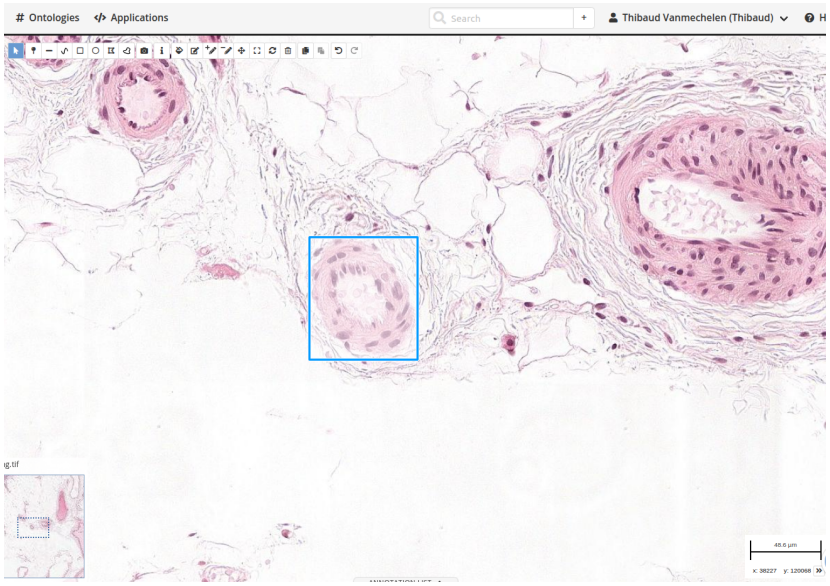
At this stage, the API has all the prompts it needs: a bounding box and possibly some positive point prompts. It then uses the bounding box to extract a region of interest from the original WSI. If this region is larger than the model's maximum input size, it is resized (so that its largest dimension becomes 1024), and the scaling factor is stored to later resize the output mask. After extracting the patch, I realign the prompts to match the coordinate system of the extracted region. Indeed, since whole slide images are typically very large, and the patch is much smaller, this transformation is clearly necessary to keep the prompts consistent. Once everything is ready, I pass the image patch and the prompts to the model to get the prediction. After receiving the output mask, I apply the post-processing method described in Chapter 5, and if the image was resized earlier, I rescale the mask back to the original size of the image. Finally, I update the corresponding annotation in the database with the new refined mask geometry. This completes the request processing. Of course, throughout this pipeline, I also included multiple checks to ensure that the process runs smoothly. If an error occurs at any point, an appropriate error message is returned.

To summarize, my integration of SAM2 into Cytomine, first, I modified the web-UI by adding a new button to the annotation detail menu. This allows the user to trigger the model when the selected annotation is of a valid type. Once clicked, the request is sent to the Core and routed to the new API via the mapping that I added to Nginx. The API then handles the request appropriately: it processes the annotation, generates the refined mask using SAM2, and directly updates the annotation in the database. The refined annotation is then automatically updated in the user's browser view.

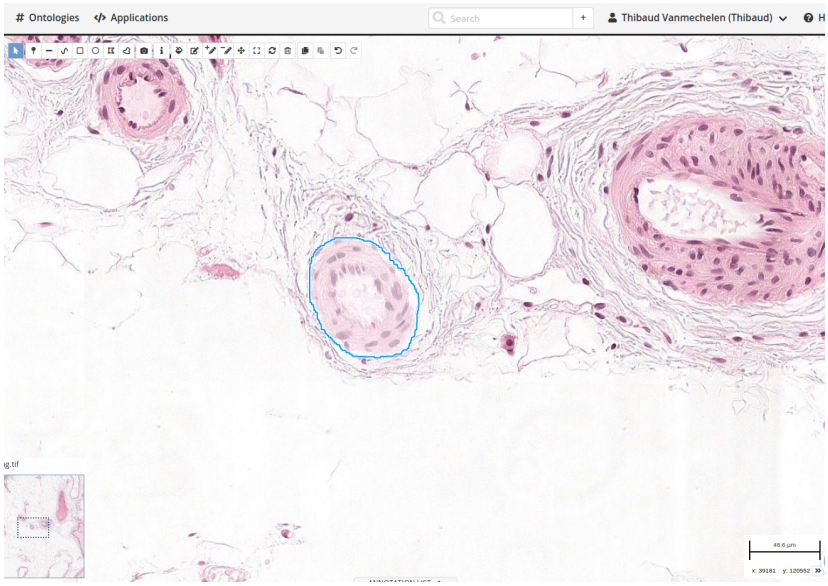
An example produced using this integration is shown in Figure 7.6 and Figure 7.7. Figure 7.6 presents the original annotation (before the processing with SAM2) alongside the final result. As we can see, the original bounding box has been refined and is now much closer to the target object. Figure 7.7 shows the same annotation with the corresponding menu, and provides a link to a full video demonstration in the caption. In the video, the end of the processing is marked by a success notification (in green), after that there is short delay before the annotation is refreshed in the browser.

Figure 7.6: Cytomine API Example (Before vs After Processing)

(a) Before Processing



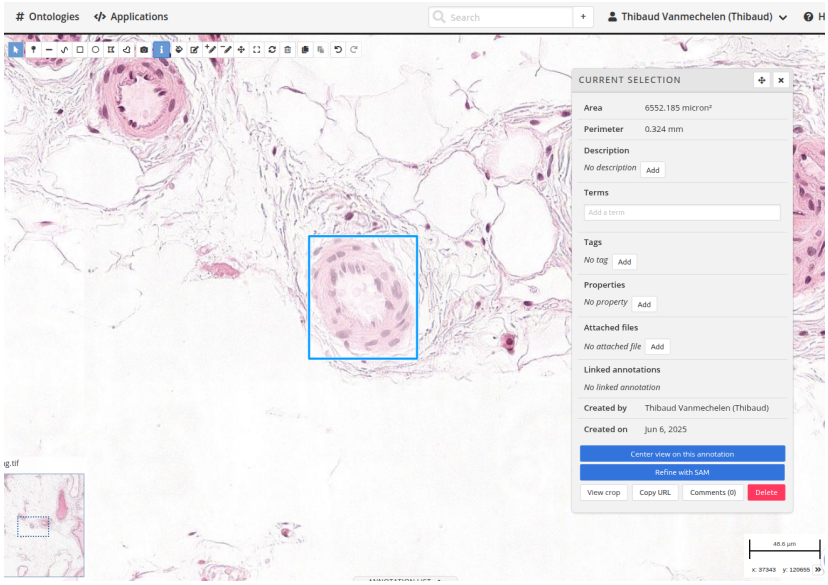
(b) After Processing



After finalizing the integration into Cytomine, I conducted a last analysis. As mentioned several times before, the processing speed is an important factor to ensure that the model is responsive. For this reason, I monitored each step within my API container and within the Core service to assess their performance (the request transmission times were negligible, as they were already extremely fast). I ran my API a total of 10 times using annotations of varying size: ranging from very small (a few dozen of pixels) to very large (several tens of thousands of pixels). After that, I averaged the results. Unfortunately, I did not have access to a computer with a GPU for the deployment, so all the operations [1], including running the model, were performed on CPU.

[1]This is also the case in the video demonstration, which was entirely executed on CPU.

Figure 7.7: Cytomine API Example Video [Link]



This significantly reduced the speed performance compared to the results presented in Table 4.7. The results of this analysis are shown in Table 7.1. The first column describes each step of the API process, and the second shows the corresponding execution time (in seconds).

Table 7.1: Execution Time Analysis of Cytomine API

Step	**Time (s)**
Core	0.002
Checking Annotation Validity	0.053
Computing Bounding Box	0.001
Retrieving Point Prompts	0.116
Formatting the Prompts	0.000
Obtaining the ROI	0.626
Inference with SAM2	5.540
Post-Processing	0.091
Updating	0.142
Total	6.571

From Table 7.1, we can see that the total execution time is not optimal. Indeed, a little over six seconds is not extremely responsive. However, if we look more closely at the time taken by each individual step, we see that most of them are actually quite fast, which is encouraging. There are mainly four steps that slow down the overall processing.

The first one is the retrieval of the point prompts. This is logical, as the API must check all the annotations (on that image) created by the user making the request to identify any point annotations inside the bounding box. This requires retrieving several elements from the database, which causes this step to be slower.

The second slowest step is the retrieval of the region of interest from the original WSI. This too is understandable, indeed, the image server is not mounted in a way that allows the API to access the images directly, so a transfer of the image patch is needed, which adds some overhead (just over half a second). Nevertheless, this delay could potentially be eliminated in the future if Cytomine is updated in a way that allows the API to access images directly from the disk.

The slowest step overall, and the main contributor to this high execution time, is the inference using SAM2. In fact, running the model on CPU is significantly slower than on GPU (approximately 44 times slower). However, this issue should be resolved once the API is deployed on a machine that has a GPU. In that case, the inference times should be much closer to those shown in Table 4.7, although some additional time will still be needed to transfer the image to the GPU. Therefore, this overhead does not seem to be a major obstacle.

Finally, the last step that takes a bit more time is the update of the annotation in the database, likely for the same reason as the point prompt retrieval (due to interactions with the database).

To sum up, while the API is not really fast when running on CPU, the main bottleneck is the SAM2 inference, which should be significantly improved when using a GPU. If we replace the CPU inference time with the corresponding one from Table 4.7 (this is slightly optimistic, as it does not take into account the GPU transfer time), the total execution time would drop to around 1.155 seconds, which is quite fast. This time could potentially be further reduced if the image-handling between the database and the API is also optimized.

This chapter concludes the work I carried out during this master's thesis. The next chapter provides a conclusion and some reflections on the overall work. Moreover, a tutorial is included in Appendix C, describing step by step how to integrate a new micro-service into Cytomine. This aims to help both users and developers who may want to extend the platform.

Chapter 8

Conclusion

In this final chapter, I summarize the main aspects of my master's thesis. I begin by describing my contributions (Section 8.1). After that, I discuss the limitations of my study, more particularly concerning the Segment Anything models and the integration of SAM2 into Cytomine, and I present a few suggestions for future work that could address these limitations (Section 8.2). Finally, I acknowledge the tools and the resources that I used throughout my work (Section 8.3).

8.1 Contributions

During my thesis, I made several contributions that were aimed at introducing an interactive segmentation tool based on Segment Anything within the Cytomine platform.

To begin with, I evaluated the zero-shot performance of SAM on histopathology images and compared it to SAM2. I carried out these experiments under varying conditions (regarding the datasets and prompting strategies). Overall, I saw that the best zero-shot performance for both models was achieved when using a combination of bounding box prompts with positive point prompts, and that, globally, SAM2 consistently outperformed SAM.

After that, I studied the performance of both models after fine-tuning and compared the results. Here again, I analyzed multiple configurations by varying several parameters such as the loss functions, dataset combinations, and prompt types. These experiments showed that the best results were achieved when training on all datasets and using all prompt types. They also further confirmed the superiority of SAM2 compared to SAM.

I then experimented with several post-processing techniques to ensure that the predicted masks were user-friendly and could be easily edited or used by end-users. Moreover, I analyzed the robustness of the models when the distribution of mask prompts was changed. Overall, the results showed that the simplest post-processing method, based on OpenCV, was the most effective to produce user-friendly masks. In addition, I observed that both Segment Anything models were not robust to changes in the mask prompt distribution after fine-tuning.

I also looked at the combination of SAM with several state-of-the-art encoders specialized in histopathology, to try to further improve the performance. To my knowledge, some of these combinations, especially with those specific encoders, have not been previously investigated in the literature. Unfortunately, all these different combinations led to worse performance compared to the best-performing SAM2 model obtained through standard fine-tuning.

Finally, I integrated the best-performing model into Cytomine by developing a new API and verified its functioning within a test version of the platform. I also conducted a speed performance analysis which showed that the performance was not optimal but could likely be significantly improved.

8.2 Limitations & Future Work

There are several limitations that can be identified in the work that I carried out during my thesis, and most of them offer potential directions for future research and development.

Firstly, I formulated several hypotheses to explain some of the surprising results, but I did not conduct further investigations to validate them (even thought I based myself on the literature). For example, in Chapter 6, we observed that using an additional domain-specific encoder did not improve the results. I proposed some hypotheses to explain this, but exploring them in more detail was outside the scope of my thesis. Indeed, my end goal was to find a practical solution for integration into Cytomine. Similarly, another example is the hypothesis regarding why fine-tuning the entire model resulted in worse performance than fine-tuning only some specific components. All these points could be explored further in future work.

Secondly, during the full model fine-tuning, I had to use small batch sizes due to hardware limitations. This might have affected the training and the final results. It would therefore be interesting to repeat these experiments on more powerful hardware to determine whether larger batch sizes would lead to better performance.

Thirdly, while I focused on combining SAM with domain-specific encoders to improve the segmentation performance, another direction which is highly popular in the literature is parameter-efficient fine-tuning, which I did not investigate due to time constraints. Techniques such as Low-Rank Adaptation (LoRA [Hu et al., 2021]) have shown strong potential in the literature and could be worth investigating in future work.

Fourthly, when combining a Segment Anything model with domain-specific encoders, I focused only on SAM, mainly because its image encoder shares the same architecture (ViT) as the domain-specific ones. However, it would be interesting to examine ways to integrate such encoders with SAM2, which uses a hierarchical encoder. This may lead to better results, although it requires more complex adaptations.

Finally, while I successfully integrated SAM2 into a test version of Cytomine and ensured that the API functioned properly, I did not consider aspects related to load balancing or scalability. Indeed, in its current form, the API may not handle many multiple concurrent users efficiently. Moreover, as shown in my final analysis from Chapter 7, the overall speed of the API is not optimal due to several performance bottlenecks, such as image transfer, and more importantly, the inference time of SAM2 on CPU. Future improvements could include the introduction of caching mechanisms, performance optimizations to make the API more robust and efficient, and more critically, the deployment on a machine equipped with a GPU.

8.3 Tools and Platforms

In this final section, I would like to acknowledge the various tools and code repositories that I used during my master's thesis.

To run my experiments, I used both Google Colaboratory and the Alan Cluster from Montefiore [Montefiore, 2018]. Indeed, having access to multiple environments in parallel was particularly helpful given the large number of models that I needed to fine-tune.

Regarding the reused code repositories, I first relied on the one developed by Mr. Noé Gille: **https://github.com/NoeGille/SAMSAM**. As explained in Chapter 3), I adapted this code for my own use.

After that, I also worked extensively with several repositories from the Cytomine platform. Namely, the ones for the web-UI, the Core service, the Nginx configuration, and the global Community Edition. All of these are publicly available at the following links: **https://github.com/Cytomine-ULiege/Cytomine-web-ui**, **https://github.com/Cytomine-ULiege/Cytomine-core**, **https://github.com/Cytomine-ULiege/bigpicture-cytomine-nginx**, and **https://github.com/cytomine/Cytomine-community-edition**. For all these repositories, I modified specific parts to integrate the new API, as explained earlier.

For the API itself, I took inspiration from a previously developed one by Mr. Ba Thien Lee for the CBIR algorithm, available here: **https://github.com/Cytomine-ULiege/Cytomine-cbir**.

In terms of additional tools, and more particularly generative artificial intelligence tools, I occasionally used ChatGPT throughout my thesis. Indeed, it helped me for some of my researches, and assisted me with debugging (particularly during the integration, when modifying some configurations). I also used it for correcting the grammar and to improve phrasing in some parts of my report.

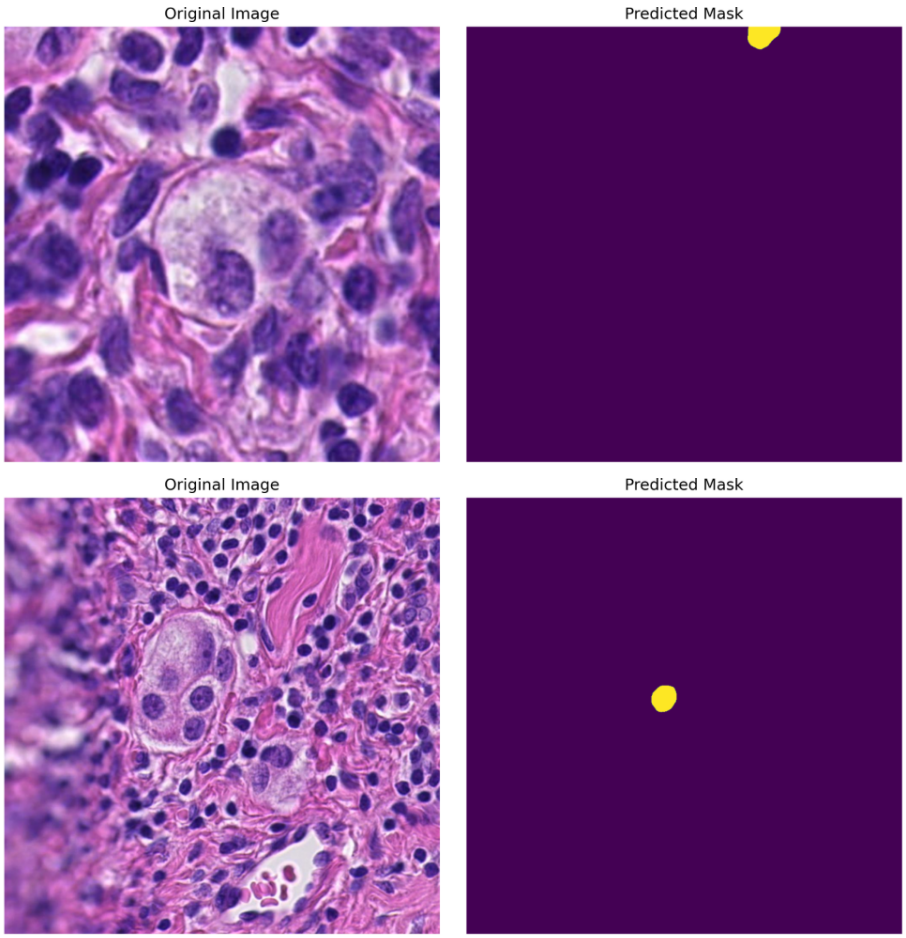
Finally, the code that I developed during this thesis is available in public repositories. The repository containing all my experiments from Chapters 4 to 6 is available here: **https://github.com/ThibaudVanmechelen/HistoSAM**. The API is located here: **https://github.com/ThibaudVanmechelen/Cytomine-sam**. And, the modifications that I made to the web interface, the Core service, the Community Edition, and the Nginx configuration can be found here: **https://github.com/ThibaudVanmechelen/Cytomine-web-ui**, **https://github.com/ThibaudVanmechelen/Cytomine-core**, **https://github.com/ThibaudVanmechelen/Cytomine-community-edition**, and **https://github.com/ThibaudVanmechelen/bigpicture-cytomine-nginx**.

Appendix A

Automatic Mode Segmentation

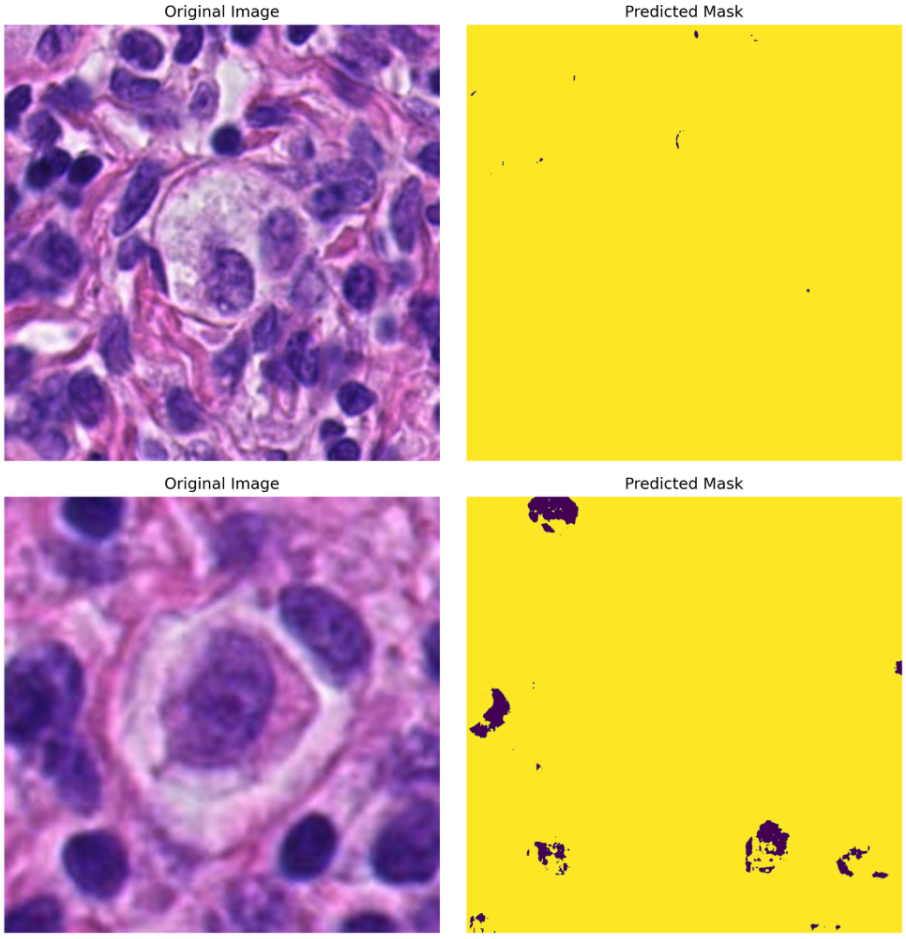
A.1 Additional Examples for SAM

Figure A.1: Additional Automatic Mode Segmentation Examples for SAM



A.2 Additional Examples for SAM2

Figure A.2: Additional Automatic Mode Segmentation Examples for SAM2

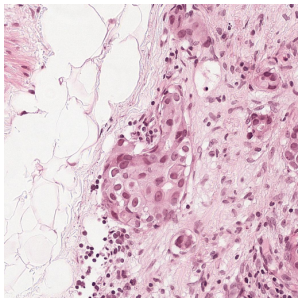


Appendix B

Mask Distribution Shift

B.1 Examples for SAM

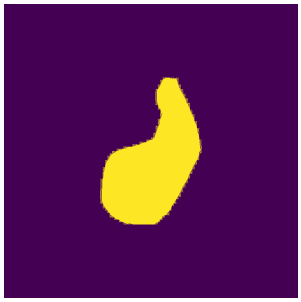
Figure B.1: Mask Distribution: Example 1 (SAM)



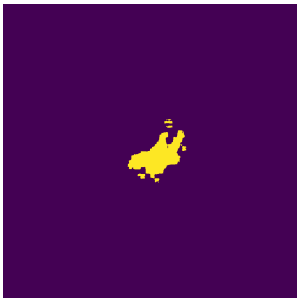
(a) Original Image

(b) Input Scribble

(c) Input Loose dilation



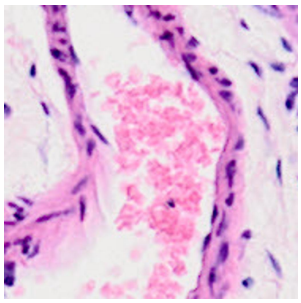
(d) Ground truth



(e) Prediction Scribble

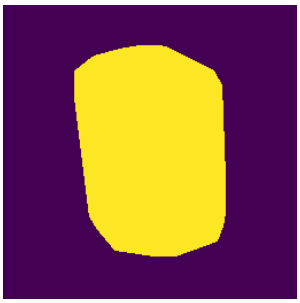
(f) Prediction Ld

Figure B.2: Mask Distribution: Example 2 (SAM)

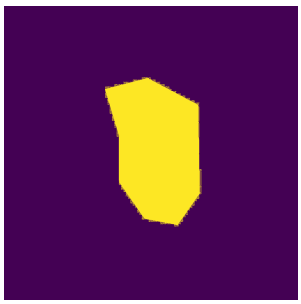


(a) Original Image

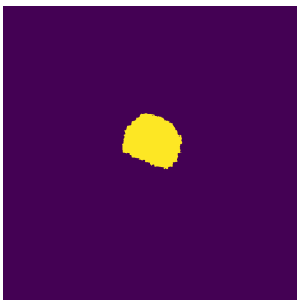
(b) Input Scribble



(c) Input Loose dilation



(d) Ground truth

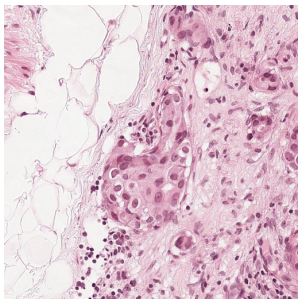


(e) Prediction Scribble

(f) Prediction Ld

B.2 Examples for SAM2

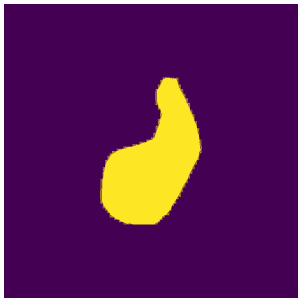
Figure B.3: Mask Distribution: Example 1 (SAM2)



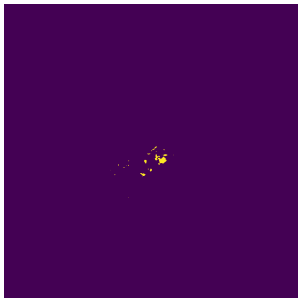
(a) Original Image

(b) Input Scribble

(c) Input Loose dilation



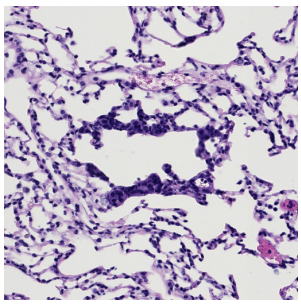
(d) Ground truth



(e) Prediction Scribble

(f) Prediction Ld

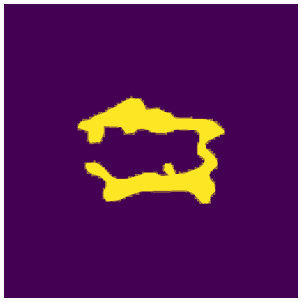
Figure B.4: Mask Distribution: Example 2 (SAM2)



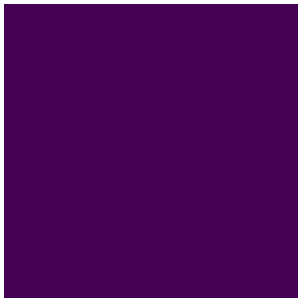
(a) Original Image

(b) Input Scribble

(c) Input Loose dilation



(d) Ground truth



(e) Prediction Scribble

(f) Prediction Ld

Appendix C

Cytomine - API Tutorial

C.1 Developing Additional Micro-Services in Community Edition

This guide describes the steps required to integrate a new micro-service into `Cytomine Community Edition`. It ensures a smooth setup and integration with the existing platform. Please note that depending on the nature and purpose of your custom micro-service, some steps, such as Web-UI modifications, may not be necessary.

C.1.1 Prerequisites

A cluster must be installed and running before proceeding with the installation. See **Clusters**.

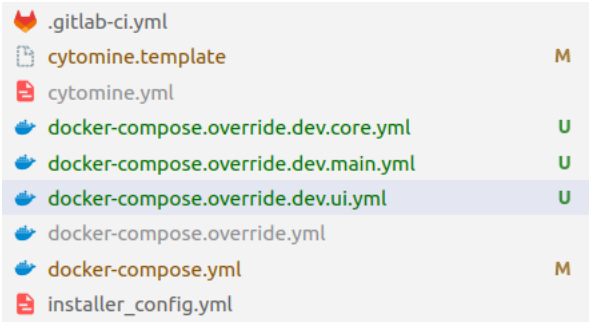
C.1.2 Installation

If you have not already installed Cytomine Community Edition, follow the official **installation instructions**.

Additionally, ensure that the following files are present in your repository:

- `docker-compose.override.dev.main.yml`: Overrides main service with its development version.
- `docker-compose.override.dev.ui.yml`: Overrides UI service with its development version.
- `docker-compose.override.dev.core.yml`: Overrides Core service with its development version.

Place these 3 files in the root directory of the Community Edition, at the same level as the `docker-compose.yml` file.

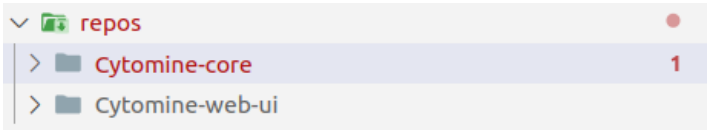


C.1.3 Web-UI

If you plan to modify the Web-UI, follow these steps:

1. If it does not already exist, create a `repos` directory at the root of the Community Edition.
2. Clone your customized version of the Web-UI into the `repos` directory:

```
cd repos
git clone <your-forked-web-ui-repo>
```



Regardless of the nature of your modifications, it is highly recommended to use the JavaScript Cytomine Client to make the API requests from the Web-UI to the Core service.

Example usage:

```
await Cytomine.instance.api.post('your_url')
```

C.1.4 Nginx

To make your API accessible from the other Docker containers, you need to configure a new route in Nginx, by following these steps:

1. Clone the **Cytomine Nginx** repository in a separate directory:

```
git clone <nginx-repo>
```

The Nginx repository can either be the original Cytomine Nginx repository or the BigPicture Nginx repository.

2. In the `Dockerfile`, define a default internal URL for your API using an environment variable:

```
ENV INTERNAL_URLS_your_api_name=your_api_url
```

```
ENV INTERNAL_URLS_IMS3=pims:5000
ENV INTERNAL_URLS_WEB_UI=web_ui
ENV INTERNAL_URLS_APPENGINE=appengine
ENV INTERNAL_URLS_CBIR=cbir
ENV RESOLVER="127.0.0.11"
ENV URLS_INTERNAL_PROXY=nginx
ENV URLS_SCHEME=http
ENV VERSIONS_CYTOMINE_COMMERCIAL="not provided"
```

3. In `nginx.conf.sample`, add a new location to proxy traffic to your API:

```
location /your_api_name {
   set $tmp_your_api_name_host "$INTERNAL_URLS_your_api_name";
   proxy_pass http://$tmp_your_api_name_host;
}
```

```

        # Internal communication to App Engine from other internal micro-services
        location /app-engine {
            set $tmpappenginehost "$INTERNAL_URLS_APPENGINE";
            proxy_pass http://$tmpappenginehost;
        }

        # Internal communication to CBIR from other internal micro-services
        location /cbir {
            set $tmpcbirhost "$INTERNAL_URLS_CBIR";
            proxy_pass http://$tmpcbirhost;
        }

        # Internal communication to CORE from other internal micro-services.
        # Base path convention: "/api" + special endpoints that should be moved under "/api" base path in the future.
        location ~ /(api|ws|login|logout|saml|saml2|server|session|custom-ui|static) {
            set $tmpcorehost "$INTERNAL_URLS_CORE";
            proxy_pass http://$tmpcorehost;
        }
```

This configuration creates a new route that matches all HTTP requests starting with `/your_api_name` and proxies them to the internal URL defined by $INTERNAL_URLS_your_api_name.

4. Create a new Docker image for the updated Nginx service:

```
sudo docker build -t my_new_nginx/nginx:0.1.0 .
```

C.1.5 Core

If you plan to modify the Core, follow these steps:

1. If it does not already exist, create a `repos` directory at the root of the Community Edition.

2. Clone your customized version of the Core into the `repos` directory.

```
cd repos
git clone <your-forked-core-repo>
```

To contact your API from the Core service, you can use the application properties to retrieve the internal proxy URL, then append your API path to it (as defined in the Nginx location). Here is an example:

```
URI url = UriComponentsBuilder
      .fromHttpUrl(applicationProperties.getInternalProxyURL())
      .path("/your_api_name/your_endpoint")
      .build()
      .toUri();
```

C.1.6 API

You are free to design your API and define its endpoints as needed. However, if you are using Python, it is highly recommended to follow the structure and the best practices (regarding Cytomine) demonstrated in the Cytomine **CBIR API** example.

Your API should be set up in order to handle requests directed to the path:

```
/your_api_name/your_endpoint
```

This path must correspond to the route defined in the Nginx configuration to ensure proper request forwarding.

Moreover, the Docker container corresponding to your API should also be build beforehand, similarly to the Docker container corresponding to the Nginx service.

C.1.7 Cytomine Community Edition

The final step is to modify the Cytomine Community Edition configuration to include your new micro-service. Follow the steps below:

In `cytomine.template`

1. Around line 30, inside the `aaa_services_uri` section, add a new entry for your micro-service:

```
YOUR_API_NAME: your_api_name:port_number
```

```
32  aaa_services_uri:
33    constant:
34      WEB_UI: web_ui
35      CORE: core:8080
36      PIMS: pims:5000
37      MONGO: mongo
38      NGINX: nginx
39      PIMS_CACHE: pims-cache:6379
40      POSTGIS: postgis
41      APPENGINE: app-engine:8080
42      REGISTRY: registry
43      your_api_name: your_api_name:6000
```

2. Around line 60, update the Nginx image and add your micro-service image to the list of constants.

```
images:
  constant:
    your_api_name: my_new_api/your_api_name:0.1.0
    CORE: cytomine/core:5.1.0
    MONGO: cytomine/mongo:1.0.0
    NGINX: my_new_nginx/nginx:0.1.0
    PIMS_CACHE: redis:7.2
    PIMS: cytomine/pims-ce-package:ce-1.0.0
    POSTGIS: cytomine/postgis:1.5.1
    WEB_UI: cytomine/web-ui:3.1.0
    APPENGINE: cytomine/app-engine:0.1.0
    REGISTRY: registry:2.8.3
```

3. Around line 210, add your micro-service URL to the list of internal service URLs.

```
nginx:
  global:
    INTERNAL_URLS_WEB_UI: aaa_services_uri.WEB_UI
    INTERNAL_URLS_CORE: aaa_services_uri.CORE
    INTERNAL_URLS_IMS: aaa_services_uri.PIMS
    INTERNAL_URLS_IMS2: aaa_services_uri.PIMS
    INTERNAL_URLS_IMS3: aaa_services_uri.PIMS
    INTERNAL_URLS_APPENGINE: aaa_services_uri.APPENGINE
    INTERNAL_URLS_your_api_name: aaa_services_uri.your_api_name
    VERSIONS_CYTOMINE_COMMERCIAL: versions.CYTOMINE_COMMERCIAL
    IMAGES_CORE: images.CORE
    IMAGES_MONGO: images.MONGO
    IMAGES_NGINX: images.NGINX
    IMAGES_PIMS_CACHE: images.PIMS_CACHE
    IMAGES_PIMS: images.PIMS
    IMAGES_POSTGIS: images.POSTGIS
    IMAGES_WEB_UI: images.WEB_UI
    URLS_SCHEME: aaa_config.SCHEME
```

4. (Optional but recommended) At the end of the file, include an environment variable definition for `API_BASE_PATH`. This step is optional if your API uses the hardcoded paths, but required if your service follows the CBIR approach which relies on environment variables to override the base path.

```
app-engine:
  constant:
    API_PREFIX: /app-engine/
  global:
    DB_HOST: aaa_services_uri.POSTGIS
    DB_PORT: postgres.PORT
    DB_NAME: postgres-appengine.DB_NAME
    DB_USERNAME: postgres-appengine.USER
    DB_PASSWORD: postgres-appengine.PASS
    REGISTRY_HOST: aaa_services_uri.REGISTRY
    STORAGE_BASE_PATH: container_paths.APPENGINE_FILE_STORAGE
    SCHEDULER_MASTER_URL: app-engine.SCHEDULER_MASTER_URL
    SCHEDULER_OAUTH_TOKEN: app-engine.SCHEDULER_OAUTH_TOKEN
    SCHEDULER_USERNAME: app-engine.SCHEDULER_USERNAME

your_api_name:
  constant:
    API_BASE_PATH: /your_api_name
```

In `docker-compose.yml`

5. Define your micro-service as a new service within `docker-compose.yml`, so that it gets built and launched alongside the other Cytomine services.

```
93    your_api_name:
94      image: my_new_api/your_api_name:0.1.0
95      restart: unless-stopped
96      networks:
97        cytomine-network:
98          ipv4_address: 172.20.0.11
```

C.1.8 Launching your Dev Version of Cytomine

To launch a working version of your customized Cytomine, follow the steps below.

Run the Installer Container

This command run the Cytomine installer container, using the current configuration files:

```
sudo docker run -v $(pwd):/install --user "$(id -u):$(id -g)" --rm -it
    cytomine/installer:latest deploy -s /install
```

Start Cytomine in Development Mode

Start Cytomine with all the necessary containers using:

```
sudo docker compose -f docker-compose.yml -f docker-compose.override.yml
    \
-f docker-compose.override.dev.main.yml -f docker-compose.override.dev.ui.
    yml \
-f docker-compose.override.dev.core.yml --env-file .env \
--profile dev-core --profile dev-ims --profile dev-ui --profile dev-ae --
    profile dev up -d
```

This launches Cytomine with all services needed.

Start the Core Application

While the Core container is running, the core application itself is not yet launched. You need to manually start it inside the container.

1. Identify the Core container:

```
sudo docker ps
```

It should be `cytomine-ce-core-1`.

2. Open a terminal in the container:

```
sudo docker exec -it cytomine-ce-core-1 /bin/bash
```

3. Launch the Core application:

```
gradle bootRun
```

Once everything is running, open your browser and go to `http://localhost/`.

C.1.9 Apply Code Modifications

Web-UI

To update the Web-UI:

- Make changes directly in the `repos` directory where you cloned your modified Web-UI.
- Refresh your browser to see the updates.

Core

To update Core code:

- After making changes, stop the application inside the Core container using `CTRL + C`.
- Then restart it by running:

```
gradle bootRun
```

This ensures that your code changes are applied.

Bibliography

Bahdanau, D., Cho, K., & Bengio, Y. (2016). Neural machine translation by jointly learning to align and translate. https://arxiv.org/abs/1409.0473

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Chen, R. J., Ding, T., Lu, M. Y., Williamson, D. F., Jaume, G., Chen, B., Zhang, A., Shao, D., Song, A. H., Shaban, M., et al. (2024). Towards a general-purpose foundation model for computational pathology. *Nature Medicine*.

Chen, T., Lu, A., Zhu, L., Ding, C., Yu, C., Ji, D., Li, Z., Sun, L., Mao, P., & Zang, Y. (2024). Sam2-adapter: Evaluating adapting segment anything 2 in downstream tasks: Camouflage, shadow, medical image segmentation, and more. https://arxiv.org/abs/2408.04579

Chen, X., Zhao, Z., Zhang, Y., Duan, M., Qi, D., & Zhao, H. (2022). Focalclick: Towards practical interactive image segmentation. https://arxiv.org/abs/2204.02574

Cheng, H. K., Chung, J., Tai, Y.-W., & Tang, C.-K. (2020). Cascadepsp: Toward class-agnostic and very high-resolution segmentation via global and local refinement. https://arxiv.org/abs/2005.02551

Cytomine ULiège R&D Team. (2023). *Cytomine documentation: Community edition*. University of Liège. https://doc.uliege.cytomine.org/admin-guide/ce/

Cytomine ULiège R&D Team. (2024). Cytomine cbir: Content-based image retrieval server [Version 0.5.0].

Deng, R., Cui, C., Liu, Q., Yao, T., Remedios, L. W., Bao, S., Landman, B. A., Wheless, L. E., Coburn, L. A., Wilson, K. T., Wang, Y., Zhao, S., Fogo, A. B., Yang, H., Tang, Y., & Huo, Y. (2023). Segment anything model (sam) for digital pathology: Assess zero-shot segmentation on whole slide imaging. https://arxiv.org/abs/2304.04155

Ehteshami Bejnordi, B., Veta, M., van Diest, P. J., van Ginneken, B., Karssemeijer, N., Litjens, G., van der Laak, J. A., & Consortium, C. (2017). Diagnostic assessment of deep learning algorithms for detection of lymph node metastases in women with breast cancer. *JAMA*, *318*(22), 2199–2210.

Ghosh, A. (2025). Finetuning sam2 for leaf disease segmentation - step-by-step tutorial.

Gille, N. (n.d.). Noé gille's github profile [Accessed: 2025-06-03].

Gillies, S., van der Wel, C., Van den Bossche, J., Taves, M. W., Arnott, J., Ward, B. C., et al. (2025, May). *Shapely* (Version 2.1.1). https://doi.org/10.5281/zenodo.5597138

Gu, H., Dong, H., Yang, J., & Mazurowski, M. A. (2024). How to build the best medical image segmentation algorithm using foundation models: A comprehensive empirical study with segment anything model. https://arxiv.org/abs/2404.09957

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021). Lora: Low-rank adaptation of large language models. https://arxiv.org/abs/2106.09685

Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W.-Y., Dollár, P., & Girshick, R. (2023). Segment anything. *arXiv:2304.02643*.

Lee, H. H., Gu, Y., Zhao, T., Xu, Y., Yang, J., Usuyama, N., Wong, C., Wei, M., Landman, B. A., Huo, Y., Santamaria-Pang, A., & Poon, H. (2024). Foundation models for biomedical image segmentation: A survey. https://arxiv.org/abs/2401.07654

Lindvall, M., Sanner, A., Petré, F., Lindman, K., Treanor, D., Lundström, C., & Löwgren, J. (2020). Tissuewand, a rapid histopathology annotation tool. *Journal of Pathology Informatics*, *11*, 27. https://doi.org/10.4103/jpi.jpi_5_20

Liu, Q., Xu, Z., Bertasius, G., & Niethammer, M. (2023a). Simpleclick: Interactive image segmentation with simple vision transformers. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 22290–22300.

Liu, Q., Xu, Z., Bertasius, G., & Niethammer, M. (2023b). Simpleclick: Interactive image segmentation with simple vision transformers. https://arxiv.org/abs/2210.11006

Liu, Z., Wang, Y., Vaidya, S., Ruehle, F., Halverson, J., Soljačić, M., Hou, T. Y., & Tegmark, M. (2025). Kan: Kolmogorov-arnold networks. https://arxiv.org/abs/2404.19756

Ma, J., He, Y., Li, F., Han, L., You, C., & Wang, B. (2024). Segment anything in medical images. *Nature Communications*, *15*(1). https://doi.org/10.1038/s41467-024-44824-z

Marée, R., Rollus, L., Stévens, B., Hoyoux, R., Louppe, G., Vandaele, R., Begon, J.-M., Kainz, P., Geurts, P., & Wehenkel, L. (2016). Collaborative analysis of multi-gigapixel imaging data using cytomine. *Bioinformatics*, *32*(9), 1395–1401. https://doi.org/10.1093/bioinformatics/btw013

Mazurowski, M. A., Dong, H., Gu, H., Yang, J., Konz, N., & Zhang, Y. (2023). Segment anything model for medical image analysis: An experimental study. *Medical Image Analysis*, *89*, 102918. https://doi.org/10.1016/j.media.2023.102918

Montefiore, I. (2018). Alan cluster - montefiore institute.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., & Sutskever, I. (2021). Learning transferable visual models from natural language supervision. https://arxiv.org/abs/2103.00020

Ravi, N., Gabeur, V., Hu, Y.-T., Hu, R., Ryali, C., Ma, T., Khedr, H., Rädle, R., Rolland, C., Gustafson, L., Mintun, E., Pan, J., Alwala, K. V., Carion, N., Wu, C.-Y., Girshick, R., Dollár, P., & Feichtenhofer, C. (2024). Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*. https://arxiv.org/abs/2408.00714

Ren, S., Luzi, F., Lahrichi, S., Kassaw, K., Collins, L. M., Bradbury, K., & Malof, J. M. (2023). Segment anything, from space? https://arxiv.org/abs/2304.13000

Ryali, C., Hu, Y.-T., Bolya, D., Wei, C., Fan, H., Huang, P.-Y., Aggarwal, V., Chowdhury, A., Poursaeed, O., Hoffman, J., Malik, J., Li, Y., & Feichtenhofer, C. (2023).

Hiera: A hierarchical vision transformer without the bells-and-whistles. https://arxiv.org/abs/2306.00989

Saillard, C., Jenatton, R., Llinares-López, F., Mariet, Z., Cahané, D., Durand, E., & Vert, J.-P. (2024). *H-optimus-0*. https://github.com/bioptimus/releases/tree/main/models/h-optimus/v0

Sengupta, S., Chakrabarty, S., & Soni, R. (2025, April). Is sam 2 better than sam in medical image segmentation? In O. Colliot & J. Mitra (Eds.), *Medical imaging 2025: Image processing* (p. 97). SPIE. https://doi.org/10.1117/12.3047370

Sofiiuk, K., Petrov, I. A., & Konushin, A. (2021). Reviving iterative training with mask guidance for interactive segmentation. https://arxiv.org/abs/2102.06583

Song, Y., Zhou, Q., Lu, X., Shao, Z., & Ma, L. (2024). Su-sam: A simple unified framework for adapting segment anything model in underperformed scenes. https://arxiv.org/abs/2401.17803

Teuber, C., Archit, A., & Pape, C. (2025). Parameter efficient fine-tuning of segment anything model. https://arxiv.org/abs/2502.00418

Wang, C., Li, D., Wang, S., Zhang, C., Wang, Y., Liu, Y., & Yang, G. (2023). SAM$^{\text{Med}}$: A medical image annotation framework based on large vision model. https://arxiv.org/abs/2307.05617

Yan, Z., Sun, W., Zhou, R., Yuan, Z., Zhang, K., Li, Y., Liu, T., Li, Q., Li, X., He, L., & Sun, L. (2024). Biomedical sam 2: Segment anything in biomedical images and videos. https://arxiv.org/abs/2408.03286

Zhang, J., Ma, K., Kapse, S., Saltz, J., Vakalopoulou, M., Prasanna, P., & Samaras, D. (2023). Sam-path: A segment anything model for semantic segmentation in digital pathology. https://arxiv.org/abs/2307.09570

Zhang, L., Liang, Y., Zhang, R., Javadi, A., & Xie, P. (2024). Blo-sam: Bi-level optimization based overfitting-preventing finetuning of sam. https://arxiv.org/abs/2402.16338

Zhang, M., Wang, L., Chen, Z., Ge, Y., & Tao, X. (2024). Path-sam2: Transfer sam2 for digital pathology semantic segmentation. https://arxiv.org/abs/2408.03651

Zhang, Y., Shen, Z., & Jiao, R. (2024). Segment anything model for medical image segmentation: Current applications and future directions. https://arxiv.org/abs/2401.03495

Zhu, J., Hamdi, A., Qi, Y., Jin, Y., & Wu, J. (2024). Medical sam 2: Segment medical images as video via segment anything model 2.