Deep Learning models for experimental images segmentation

Francisco Santos de Vilhena Costa
Master in Engineering Physics
Department of Physics and Astronomy
Faculty of Sciences of University of Porto and Faculty of Engineering of University of Porto
2024/2025

Deep Learning models for experimental images segmentation

Francisco Vilhena Costa
Dissertation carried out as part of the Master in Engineering Physics
Department of Physics and Astronomy
2024/2025

Supervisor
Mónica Filgueiras, PhD, CEFT

Co-supervisor
Zafeiris Kokkinogenis, PhD, LIACC

Acknowledgements

I would like to express my deep appreciation to all those who helped me through this journey. To my supervisors Mónica Filgueiras and Zafeiris Kokkinogenis, I am grateful for the guidance and support given to me throughout this work. A special thanks to my family and friends, always present when times got tougher!

Francisco Vilhena Costa

Abstract

In this work, four distinct state of the art deep-learning architectures were compared and implemented for the task of multiclass image segmentation of two-phase bubbly flow experimental images. Moreover, the effect of image enhancing techniques was studied to assess whether this technique can improve the segmentation accuracy of the models used. This study begins with a comprehensive overview of existing image segmentation methods, tracing their evolution, and identifying the gap in the application of multiclass models to bubbly flow analysis. To enable consistent model comparison, an end-to-end and easily adaptable pipeline was developed with the stages: training, validation, inference, and characterization. Using these pipelines, we implement and benchmark four networks: YOLO, Detectron2, TransUNet, and SAM-LST, on a four-class bubble dataset (Bubble, Cut, Aggregate, and Spherical). Each model was tested using two parallel approaches: one based on the original images and another using images enhanced through Super-Resolution (SR) techniques. Both approaches produced reliable segmentation masks and enabled accurate extraction of bubble characterizing metrics. TransUNet achieved the highest performance, followed by SAM, Detectron2 and YOLO based models. The effect of applying Super-Resolution (SR) techniques was also assessed in terms of their impact on segmentation performance. The analysis showed that the benefits are not constant: in some cases, SR led to better segmentation results, while in others, the impact was neutral or slightly negative, depending on the model and class involved. To test generalization, the best models were applied to a maze-entrapment image-set, beyond the original training dataset, with low image quality. Finally, future directions were discussed, including further modularization of the developed pipelines for rapid model integration, exploring different architectures, and systematically identifying the model topologies that most benefit from synthetic data augmentation.

Keywords: Bubbly flow; Multiphase Flow; Super-Resolution; Multiclass Instance Segmentation; Deep-Learning

Resumo

Neste trabalho, quatro arquiteturas distintas de 'Aprendizagem Profunda' (YOLO, Detectron2, TransUNet e SAM-LST) foram comparadas e implementadas para a tarefa de segmentação multiclasse de imagens experimentais de escoamentos bifásicos com bolhas. Além disso, estudou-se o efeito de técnicas de melhoria de imagem para avaliar se estas podem melhorar a precisão de segmentação dos modelos utilizados. Para tal, foi desenvolvida uma estrutura de processamento integrada (com treino, validação, inferência e caracterização) que permite a comparação direta entre modelos. O conjunto de dados de treino dos modelos foi composto por imagens experimentais reais de escoamento com bolhas, com quatro classes (*Bubble*, *Cut*, *Aggregate* e *Spherical*). Cada modelo foi testado com duas abordagens paralelas: uma baseada nas imagens originais e outra utilizando imagens melhoradas por Super-Resolução. Foram obtidas máscaras de segmentação fiáveis, permitindo a extração precisa de caracteristicas. O modelo TransUNet obteve o melhor desempenho, seguido por SAM-LST, Detectron2 e YOLO. O efeito da aplicação de técnicas de Super-Resolução (SR) foi também avaliado na tarefa de segmentação revelando um efeito incosistente: em alguns casos, a SR conduziu a melhores resultados de segmentação, enquanto em outros, o impacto foi neutro ou ligeiramente negativo, variando conforme o modelo e a classe em questão. Para verificar a capacidade de generalização, os modelos com melhor desempenho foram aplicados com sucesso a imagens de qualidade muito baixa de uma experiência de labirinto. O trabalho termina com propostas de direções futuras, como a exploração de arquiteturas mais profundas e a continuacao do estudo do efeito de técnicas de SR. Outros desenvolvimentos podem incluir a melhoria da pipeline de forma a promover uma maior abrangência e integração mais rápida de novos modelos, assim como a exploração de arquiteturas e identificação sistemática das que mais se beneficiam da melhoria da qualidade das imagens através de técnicas de SR.

Palavras-chave: Escoamento de bolhas; Escoamento multifásico; Super-Resolução; Segmentação de instâncias multiclasse; Aprendizagem profunda

Contents

List of Tables

List of Figures

Acronyms

AI Artificial Intelligence

ANN Artificial Neural Network

CCA Concentric Circular Arrangements

CNN Convolutional Neural Network

CRF Conditional Random Field

CSP Cross Stage Partial

C2PSA Cross Stage Partial with Spatial Attention

DL Deep Learning

FAIR Facebook AI Research

FCN Fully Convolutional Network

FPN Feature Pyramid Network

IoU Intersection over Union

KNN K-Nearest Neighbour

LST Ladder-Side Tuning

MAE Masked Autoencoding

mAP50–95 Mean Average Precision (IoU = 0.50–0.95)

AP50 Average Precision at IoU = 0.50

mIoU Mean Intersection over Union

ML Machine Learning

MSE Mean Squared Error

NN Neural Network

RNN Recurrent Neural Network

RPN Region Proposal Network

RF Random Forest

RoI Region of Interest

SAM Segment Anything Model

SPPF Spatial Pyramid Pooling—Fast

SR Super-Resolution

SVM Support Vector Machine

TAL Task Alignment Learning

ViT Vision Transformer

YOLO You Only Look Once

Chapter 1

Introduction

In the present day and age, the automation of certain areas of science is becoming inevitable. Driven by rapid advances in algorithms, computational methods, and hardware capabilities, this transformation leverages the rise of machine learning and deep learning techniques that, by learning from datasets, are capable of revealing insights into intricate patterns that were previously beyond reach. These techniques fundamentally reshape scientific workflows, serving as a tool for scientists to be more efficient and productive, and producing results with levels of accuracy similar to human performance, sometimes even surpassing it [1].

In this thesis, we explore how such techniques can be applied to fluid mechanics, particularly to improve the analysis of two-phase flow systems starting, in this chapter, with a comprehensive overview of the fundamental concepts, objectives, and motivations underlying the research presented.

1.1 Context and Motivation

In fluid mechanics, various flow regimes can be observed, starting from simple single-phase systems such as water flowing through a pipe to more intricate multiphase scenarios. Multiphase flow systems consist of two or more distinct phases that co-exist and interact. These phases can be solid, liquid, or gas, and have different physical or chemical properties and can be found in various environments, both occurring in nature, like the water flowing in a river or man-made scenarios. Multiphase systems play a fundamental role in numerous industrial processes, including petrochemical refining, wastewater treatment, and nuclear reactor cooling [2] [3].

Understanding these systems is not merely an academic exercise, but a crucial factor in designing more efficient industrial equipment, enhancing safety measures, and improving predictive models used in engineering simulations. However, their inherent complexity poses significant challenges for accurate modeling.

The systems analyzed in this thesis are two-phase flows. These arise when two distinct phases, in this case gas and liquid, coexist and interact within the same flow domain. Depending on the distribution and interaction between these phases, several sub-regimes can be identified, such as slug flow, annular flow, or bubbly flow [4].

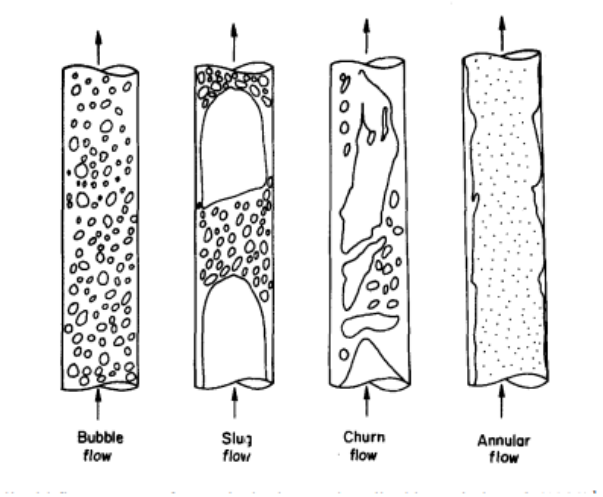

Figure 1.1: Gas-liquid flow patterns for vertical tubes as described by Taitel et al. (1980) [4].

Bubbly flow consists of a specific bi-phasic flow with a dispersed (gas) phase and a continuous (liquid) phase. Bubbly flow is characterized by the presence of small bubbles in a liquid phase [5]. This type of flow is complex due to the numerous interactions between the two phases that can greatly influence flow dynamics.

Analyzing bubble distribution and size is an important step in studying multiphase systems since controlling these parameters can dramatically impact coalescence and bubble breakup phenomena, which have a direct effect on heat and mass transfer efficiency [6]. For example, in bioreactors, it is used for oxygenation in cell cultures, allowing for precise control and ensuring optimal oxygen delivery to microorganisms or cells, enhancing growth rates and production yields [7]. In high-stakes environments, like nuclear reactors, it can play a vital role in order to monitor coolant flow, thus improving safety and operational reliability [8].

Several methods can be used for this bubble analysis. A more traditional approach involves

manual analysis of bubbly flow images; however, this is time-consuming, inconsistent, and prone to human error. As imaging equipment continues to improve, producing higher volumes of data, the need for automated, reliable, and fast image processing solutions becomes more pressing.

This thesis focuses on using deep learning (DL) models to tackle experimental image segmentation of bubbly flow. These methods allow for rapid and accurate analysis of large image datasets, enabling researchers to capture transient and complex phenomena with improved speed and precision.

Combined with a robust automated image analysis pipeline, this work gives a comprehensive overview of existing model architectures and the key aspects of advanced DL techniques for image segmentation, detection, and characterization, along with their evaluation, with a specific focus on their application to the experimental image analysis of two-phase flow.

It also addresses some of the challenges faced when using these models, such as variable experimental conditions and poor image quality, as well as solutions with a focus on the use of super-resolution (SR) as a way to tackle them.

With the ultimate goal being to compare various model architectures to assess which is more suited to the task of bubbly flow image segmentation, the development of a reproducible framework that allows for doing so efficiently becomes an important objective for this project. With this framework, researchers in different fields have an approachable way to use deep learning multiclass image segmentation technology and obtain more precise measurements, identify critical parameters, and optimize design and operation conditions. Such optimization can have wide-ranging benefits, from enhancing energy conservation and minimizing equipment fouling to preventing hazardous conditions associated with improper flow or phase distribution.

1.2 Objectives and Research Questions

The scope of this project focuses on the detection, segmentation, and classification of bubbles in experimental images. To address the challenge of selecting and fine-tuning deep-learning models for multiclass segmentation and classification of bubbly-flow images, despite only having a small training set with highly variable image conditions, the following objectives were defined: i) Research, implement and compare several state-of-the-art deep learning model architectures for multiclass image segmentation ii) Evaluate the influence of training data quality on model

performance, particularly by applying Super-Resolution (SR) pre-processing techniques to assess whether enhanced image quality improves segmentation results; iii) Assess the generalization capabilities of the models by testing them on image sets with different characteristics than those used for training; iv) Analyze the extracted bubble characteristics, including count, size, shape descriptors (e.g., circularity), and other relevant geometric features. Ultimately, this study aims to expand the understanding of two-phase flow dynamics in both academic and real-world settings by developing an approachable training and validation pipeline to compare different models that can be applicable in various growing sectors.

In order to guide the work developed, the following research questions will be addressed and revisited in the conclusion chapter:

- What are the most effective deep learning methods to perform multiclass bubbly-flow image segmentation?
- How does Super-Resolution pre-processing affect segmentation performance in bubbly flows, particularly under low-quality or low-resolution imaging conditions?
- Can deep learning models for multiclass segmentation be effectively used to extract reliable geometric and physical characteristics of bubbles in multiphase flow images?
- Does the proposed end-to-end segmentation pipeline offer a generalizable framework for multiclass bubble analysis in multiphase flow experiments?

The remainder of this document is organized as follows. **Chapter 2** introduces the foundational concepts necessary to understand the work developed, as well as a review of the state-of-the-art methods relevant to multiclass image segmentation and classification and the key challenges identified in the existing literature. **Chapter 3** provides a detailed description of the models implemented and the evaluation metrics employed. **Chapter 4** presents the dataset used in this study and outlines the proposed methodology, including a comprehensive description of the segmentation pipelines and the model comparison approach. In **Chapter 5**, the experimental setup is described and the results are presented and discussed in detail. Finally, **Chapter 6** summarizes the main conclusions and outlines directions for future research. This structured approach ensures a comprehensive understanding of the challenges and opportunities in bubbly flow analysis.

Chapter 2

Background

This chapter provides a comprehensive overview of the existing methods in the field of object detection and image segmentation. Techniques for image segmentation and bubble detection are explored, focusing on the improvement and progression of existing work, culminating with the most recent developments in the field, its principles, strengths, and limitations. Then, the models selected are described in greater detail, followed by the evaluation metrics used to assess the model's efficiency. The chapter ends with the challenges faced in bubbly flow analysis and evaluation metrics most suited for this type of work.

2.1 Models and Techniques Overview

Bubbles in a two-phase system can be evaluated using two main categories of measurement methods: those that are intrusive and those that are non-intrusive [9]. **Intrusive Measurements** include the use of various types of sensors such as conductivity probes, optical fiber probes, hot-film probes, and wire-mesh sensors. These techniques provide high spatial and temporal resolution of local parameters such as bubble characteristics however, these probes disturb the flow and bubble distribution, which is not optimal, especially in highly dynamic or small-scale systems.

In contrast, **Non-Intrusive Measurement** approaches, such as Doppler anemometry, X-ray computed tomography, Interferometric Particle Imaging and Digital Image Analysis, offer the advantage of measuring flow characteristics without physically interacting with the system. Doppler anemometry provides high temporal resolution but is limited to a single point. X-ray

allows for three-dimensional reconstruction, but its operation is costly. Interferometric particle imaging is highly accurate for small particle detection, but not as effective in environments with high particle overlap.

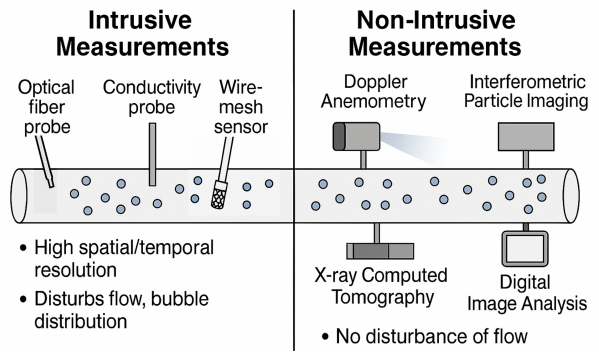

Figure 2.1: Intrusive sensors penetrate the flow and disturb bubble dynamics, while non-intrusive sensors allow accurate measurements preserving the flow.

Among these, Digital Imaging is a cost-effective alternative that plays a central role in experimental fluid mechanics, allowing for the conversion of physical phenomena into visual data that can be processed using different algorithms. From the acquired flow images, using Digital Imaging, we can extract quantitative and descriptive features such as shape, size, count, and distribution of bubbles with **Image Analysis**. Image analysis encompasses a wide range of operations needed to extract and interpret information from images. It usually involves multiple stages, including image pre-processing (e.g., denoising, contrast enhancement), feature extraction, object detection, and classification. The specific steps depend on both the imaging conditions and the goals of the analysis.

A fundamental component in Image Analysis is **Image segmentation**, the process in which the image is partitioned into meaningful regions corresponding to objects of interest, in this case, the bubbles within the fluid. Segmentation is essential so that further analysis, like the measurement of object characteristics (bubble size, shape, quantity, etc.), can be performed. It depends on several factors that influence the segmentation model's choice, which are discussed in more detail in Section 2.3. Image segmentation can be performed using traditional, machine learning (ML), or deep learning (DL) approaches.

Traditional approach relies on methods like thresholding, edge and region-based detection to distinguish objects based on intensity, color, or texture differences. It is mostly effective in images with good visibility and high contrast, low bubble overlap, and simple and consistent shapes. It requires few computational resources and can be deployed quickly with methods like

Otsu threshold [10], Hough transform [11], Watershed segmentation, Canny edge detection [12], and Concentric Circular Arrangements (CCA) [13]. When a purely geometric approach does not work and the complexity increases, a **machine learning approach** can be the most useful [14].

Machine learning enables computers to learn patterns from data using statistical models and make predictions or decisions without being explicitly programmed. This approach is best suited when there is an increase in the overlaps and scale variation of the target object. When there are data constraints, and a small labeled dataset is not enough to train a Deep Learning model, this can be the best option. However, it is still highly dependent on feature engineering and manually fine-tuning the models, relying on a fixed set of features for the ability to separate bubbles from the background. Combined with a feature extraction step, some commonly used Machine Learning models for image segmentation are Random Forest and Support Vector Machines [15].

A **deep learning approach** is used when the complexity of segmentation is the highest. DL is a subset of machine learning that uses multilayered neural networks, called deep neural networks. These neural networks are composed of many layers of neurons that process and transform data through weighted connections and activation functions and automatically extract features from an image. DL segmentation can be broadly separated into Semantic and Instance segmentation. *Semantic Segmentation* aims to assign a class label to every pixel and is often sufficient when trying to differentiate between bubble and background, for example. It has the downside of not being suited to distinguish between multiple instances of the same class and tasks like counting or tracking objects. Some common examples include Fully Convolutional Networks FCN-like architectures, encoder-decoder architectures such as U-Net [16] or SegNet [17] and Dilated Convolution-based architectures like Deep Lab [18]. *Instance segmentation* aims to identify and segment each object separately. This higher level in detail comes at a higher cost in computational efforts and often leverages object detection approaches with bounding-box proposals as opposed to the dense classification across the entire image done by semantic segmentation. Object detection can be further divided into one-stage or two-stage frameworks. The first prioritizes the computation speed, critical in real-time applications, and can be found in models like YOLO [19]. Two-stage, on the other hand, generates region proposals first and then refines classification/mask in a second stage. This framework achieves higher accuracies, being more common and present in models like Faster R-CNN and Mask R-CNN.

While traditional and classical machine learning can be viable for simpler problems, they are outperformed by deep learning models in both performance and applicability for more

complex task demands like multiclass segmentation. Deep learning models are at the forefront of technology and, due to their higher complexity, are capable of tackling more challenging situations and will therefore be studied in greater detail throughout this thesis [20].

2.2 State of the art - Image Segmentation using Deep Learning

The use of deep learning in image processing is closely tied to the development of Convolutional Neural Networks (CNNs), which were first introduced in the 20th century - Figure 2.2. CNN were popularized by Yann LeCun in the late 1980s for digit recognition with the LeNet [21]. While promising, the widespread adoption was limited by computational constraints and a lack of large training datasets. In the ImageNet Large Scale Visual Recognition Challenge (ILSVRC 2012) in 2012, a major breakthrough occurred with the introduction of AlexNet [22]. Leveraging large-scale datasets such as ImageNet and the computational power of GPUs, AlexNet achieved unparalleled performance accelarating adoption of deep learning architectures for computer vision tasks. This model reduced the top-5 error rate (percentage of test images for which the correct label is not among the model's top 5 predicted labels), outperforming other models by a significant margin.

Two major lines of research emerged following AlexNet's breakthrough. The first, focused on developing deeper neural network architectures to improve performance, leading to models like GoogLeNet (2014) [23], VGGNet (2014) [24], and ResNet (2015) [25]. The second aimed to maintain AlexNet-level performance while reducing computational costs, resulting in the development of lightweight architectures like SqueezeNet (2016) [26], MobileNet (2017) [27], EfficientNet (2019) [28].

Following the trend of developing deeper networks, Shelhamer et al. introduced Fully Convolutional Networks (FCN) for Semantic Segmentation [29]. This model replaced fully connected layers with convolutional layers, enabling end-to-end, pixel-level prediction from arbitrary-sized inputs. To further refine this approach, encoder - decoder architectures were introduced, leading to models like DeepLab, U-Net and Segnet. **DeepLab**, proposed by Chen et al. [18] improved the localization of segmentation boundaries without overly increasing computational complexity. This was achieved by combining the neural network's initial predictions with a fully connected Conditional Random Field (CRF).

Another model, **U-Net**, introduced by Ronneberger et al. [16], incorporated skip connections

to merge low-level details and high-level context, producing cleaner boundaries, and is more commonly used for biomedical imaging. Finally, **SegNet**, introduced in 2016 by Badrinarayanan et al. [17], and improved computational efficiency by storing pooling indices for more precise upsampling. However, while requiring fewer resources, it traded some detail quality for efficiency compared to U-Net. Further improvements on these models were made like U-Net++ by Zhou et al. (2018) [30] that managed to enhance the feature extraction process using densely connected nested decoder sub-networks.

DeepLabV2 achieved improved segmentation results through the use of parallel dilated convolutions, enabling it to capture multi-scale contextual information more effectively. DeepLabV3 by Chen et al. (2017) [31] further improved the model by incorporating global average pooling on the final feature map, followed by bilinear upsampling to restore the spatial resolution of the segmentation output, also largely moving away from Conditional Random Field (CFR) approach. Finally, DeepLabV3+ [32] introduced an encoder-decoder architecture, which significantly improved the recovery of fine spatial details and overall segmentation accuracy.

Beyond semantic segmentation, researchers expanded CNNs into instance segmentation, a task where individual object instances are separately identified and classified.

One of the earliest methods, R-CNN, proposed by Girshick et al. (2014) [33] introduced a two-stage approach. First, it used Selective Search to generate region proposals (bounding boxes) that likely contained objects. Then, a deep CNN was applied to extract features from each region independently for classification, enabling object-level recognition within an image.

To address the inefficiencies of R-CNN, Girshick expanded the previous work with Fast R-CNN (2015) [34], which improved both training and inference speed while enhancing detection accuracy. In this approach, it was first finetuned a ConvNet on object proposals using log loss, and the Support Vector Machine (SVM) were fitted to ConvNet features allowing for the model to be trained on the very deep VGG16 network 9x faster.

Building on this, Faster R-CNN by Ren et al. (2016) [35], eliminated a computational bottleneck caused by Selective Search used in Fast R-CNN by incorporating a Region Proposal Network (RPN). This model follows a two-stage detection pipeline: first, the RPN, a deep fully convolutional network, generates object proposals directly from feature maps. Then, Fast R-CNN classifies and refines these proposals, achieving real-time object detection with higher accuracy than previous approaches.

Facebook AI Research (FAIR) has played a pivotal role in advancing this field: many of the improvements found in today's models originate from projects supported by this group. To enable rapid implementation and evaluation of novel computer-vision ideas, FAIR released Detectron [36], a high-quality, high-performance codebase for object-detection research. Its architecture paved the way for numerous follow-on projects among them Feature Pyramid Networks for Object Detection [37], RetinaNet [38], Mask R-CNN [39], and many others.

Most notably, Mask R-CNN introduced by He et al. (2018) [39] extended object detection frameworks allowing for the generation of high-resolution masks at the pixel level, further refining segmentation tasks. This method builds on Faster R-CNN by adding a segmentation branch to predict an object mask in parallel with the existing branch for bounding box recognition. By implementing the feature extraction module RoIAlign instead of RoIPooling, it improved spatial accuracy, leading to state-of-the-art performance on benchmarks like COCO (a large-scale dataset for object detection, segmentation and captioning), surpassing previous single-model approaches in both detection and segmentation tasks.

Detectron2 [40] improves its predecessor with new capabilities and faster training. This system supports the application of new capabilities like panoptic segmentation (combined semantic and instance segmentation), rotated bounding boxes, DeepLab, ViTDet and others, being a platform that allows new developments to built on top of it.

In contrast, one-stage detectors, notably YOLO (You Only Look Once), introduced by Redmon et al. (2016) [19], simplified object detection pipelines by predicting bounding boxes and class scores in a single pass. This approach improved inference speed, making real-time object detection more accessible. Subsequent versions of YOLO introduced several key improvements. YOLOv2 (2017) refined accuracy by incorporating batch normalization, anchor boxes, and multi-scale training. YOLOv3 (2018) added Feature Pyramid Networks (FPN) for better multi-scale detection and adopted Darknet-53, a deeper and more efficient backbone. Later versions, such as YOLOv5 (2020), YOLOv6 (2020), and beyond, optimized model architecture, training strategies, and computational efficiency, making YOLO one of the most widely used real-time object detection frameworks today. As of the time of writing this thesis, YOLO has continued evolving with the model YOLO v8 [41] and beyond the involvement of its original creator, with community contributors' versions extending up to YOLO v11 [42].

Taking a different approach, the transformer architecture substitutes convolution in favor of a pure self-attention mechanism. Transformers were originally devised by Vaswani et al.

in 2017 [43] for machine translation. In this network architecture, 'self-attention' is its core operation, where each input "token" is projected into query, key, and value vectors, computed attention weights via scaled dot-product, and aggregated information across the entire sequence in parallel. By stacking multi-head self-attention layers with feed-forward networks, residual connections, and layer normalization, these models excel at capturing long-range dependencies with impressive efficiency.

Leveraging this architecture's flexibility, Dosovitskiy et al. [44] developed Vision Transformer (ViT) to image segmentation. ViT splits an image into fixed-size patches, projects them into embeddings (with positional encoding and an added class token), and processes them through standard Transformer encoder layers whose multi-head self-attention captures global context. The final class representation feeds a simple MLP head for classification, demonstrating that pure attention mechanisms can rival convolutional networks when scaled and trained on large datasets.

Building on ViT self-attention mechanism Liu et al. with Swin Transformer V2 (2021) [45] enhances image segmentation performance by hierarchically processing images and using shifted window attention, making it computationally efficient for high-resolution inputs. Unlike traditional CNN-based architectures that use convolutional operations, Swin Transformer V2 leverages self-attention to model long-range dependencies more effectively. Compared to earlier ViT models, Swin Transformer V2 introduces log-spaced attention scaling and improved training stability, allowing it to handle extremely high-resolution images with reduced memory consumption. This makes it particularly useful in applications requiring detailed object segmentation and large-scale image processing, such as medical imaging and remote sensing.

As for one of the most recent developments in segmentation, the Segment Anything Model (SAM), introduced by Meta AI(2023) [46], represents a new paradigm in image segmentation. It functions as a universal, promptable segmentation model capable of segmenting objects without task-specific training. Unlike traditional deep learning-based segmentation models - such as U-Net, DeepLab, and Mask R-CNN, which require dataset-specific fine-tuning, SAM is trained on a massive dataset, SA-1B, of 1 billion real-world images and masks and generalizes to new images in a zero-shot manner (without being explicitly trained on that task or on labeled data from that domain). Its innovative prompt-based approach allows segmentation to be guided by user-provided inputs such as points, bounding boxes, or freehand selections, making it highly adaptable to different use cases. Additionally, SAM employs a transformer-based architecture,

specifically an image encoder with a lightweight mask decoder, enabling efficient segmentation while maintaining state-of-the-art accuracy. This model differs from existing semantic and instance segmentation methods by not being restricted to predefined categories and excelling in open-world, real-time applications where flexibility and generalization are essential.

To summarize the developments discussed, in the multiphase flow and bubble detection field, bubble segmentation and classification progressed from traditional computer vision methods to basic CNN and U-net variants around 2016. Then, with Mask R-CNN the accuracy and generalization was improved with works like bubble detection and mask extraction by Kim et al. (2021) [47].

Since then, several models have been adapted to address specific challenges. However, due to the niche nature of this application area, many of the latest segmentation models have not yet been widely adopted or tailored for bubbly flow image analysis. This represents an opportunity for future work to bridge the gap between general-purpose deep learning models and domain-specific segmentation tasks in fluid dynamics.

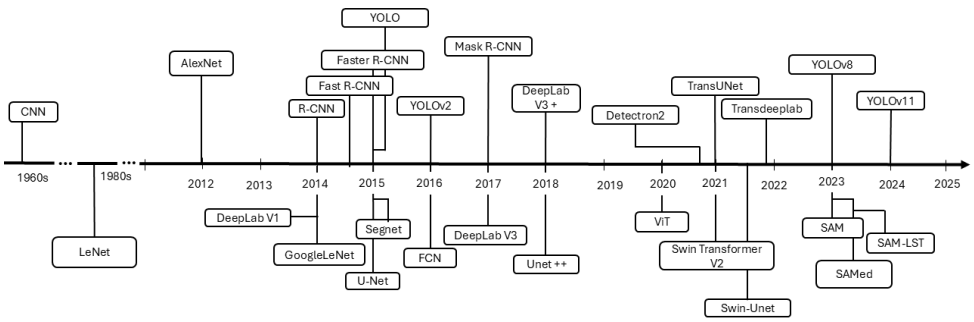

Figure 2.2: Timeline of major deep learning models in image processing and computer vision, highlighting key innovations

2.3 Challenges

2.3.1 Challenges in Two-Phase Flow Image Segmentation

There are several challenges that can greatly impact the type of model best suited for the task of image segmentation, as well as its quality and subsequent image analysis and measurements. Some of the biggest obstacles are High Gas Holdup and High-Void Fraction, Varying Transparency, Overlapping Bubbles, Bubble Size and Shape variability, High-Void Fraction scenarios, Lighting Variations, Optical Differences between phases, Noise and Sensor/Camera Artifacts. A lot of these are highly related to each other and therefore similar solutions are used for different problems.

Environments with **High Gas Holdup** are characterized by high concentrations of gas within the liquid phase, making it difficult to distinguish individual bubbles. Often related to this, in **High-Void Fraction** environments, most of the frame is occupied by bubbles, leading to saturation or reflection artifacts that can degrade segmentation accuracy. Both these situations commonly lead to **Overlapping Bubbles**, which complicates segmentation by merging or obscuring bubble boundaries and often leads to the occurrence of overlapping bubbles, meaning over-segmentation (splitting single bubbles incorrectly) or under-segmentation (merging multiple bubbles into one) are common problems. In these cases, models that allow the better retention of spatial details are more effective at separating densely packed objects. Cui et al. (2022) using a Mask R-CNN + shape reconstruction [48], provided an interesting contribution with the addition of a shape reconstruction module to restore overlapping geometries in submillimeter flows. Other examples of work done to mitigate these problems are Hessenkemper et al. [49] with a comparison of open-source U-Net's, StarDist to a Mask-RCNN CNN approach with focus on overlapping bubbles in air-water bubbly flows and Wen et al. [50] CNN Kalman Filter (KF) approach.

With obstacles like **Varying Transparency**, differences in fluid or bubble transparency, blur phase boundaries and hinder clear edge detection. Similarly with **Optical Differences** between phases, refractive index changes and reflections at the gas - liquid interface, introduce visual distortions and make accurate boundary detection more challenging. Another common problem, like **Lighting Variations**, results in uneven illumination, glare, or shadows that often distort bubble edges and reduce consistency in segmentation. Most of the works that tackle these problems do it in order to perform bubble dynamics tracking. Soibam et al. [51] deals

with the difficulties posed by shadows, reflections, and chaotic backgrounds, leveraging transfer learning to improve robustness under such conditions. Suh et al. [52] developed Vision-IT, a Mask-RCNN based model that presents a two-phase object tracking framework, enabling feature extraction from multi-phase video data. This work was utilized by Chang et al. [53] to predict critical heat flux through digital flow bubbles and Lee et al. [54] to study structure-enhanced boiling heat transfer performances. Despite being more focused on tracking and heat flux study, this approach utilizes a model that improves resilience to reflections, background noise, blurry and shadowed bubbles.

Bubble Size and Shape Variability presents a major challenge in multiphase flow analysis, where the presence of bubble sizes ranging from micro-scale to large, overlapping, and highly deformed forms impacts the effectiveness of the segmentation. To address this Haas et al. (2020) introduced *BubCNN*, combining Faster R-CNN with a regression network to estimate both bounding boxes and elliptical shape parameters, effectively handling complex bubble contours [55]. More recently, Nizovtseva et al. (2024) adapted YOLOv9 for real-time detection and segmentation of bubbles under high-speed flow, focusing on robust characterization across variable shapes and trajectories by integrating deep learning with physical modeling [56] Kang et al. (2025) proposed a multitask *Bubble Boundary R-CNN* that extends the Mask R-CNN framework with auxiliary semantic and boundary-preserving heads, enabling accurate reconstruction of irregular, overlapping bubbles [57].

Finally, **Noise and Sensor/Camera Artifacts** problems can introduce motion blur or pixelation that can further degrade image quality, making it difficult for the model to reliably differentiate between bubbles and background. Low-quality image input is a common occurrence that can greatly hinder models' training, affecting their performance. A way to address this is by adding a preprocessing step to images, improving their quality with Generative Adversarial Networks (GANs) before the training, as mentioned before. In this thesis, as stated in chapter 1, it's going to be assessed whether super-resolution (SR) preprocessing can enhance the segmentation quality of bubbly flow images.

2.3.2 Super-Resolution (SR)

Traditional SR methods relied on interpolation and example-based learning, but recent developments in deep-learning-based SR particularly GAN-based models have demonstrated superior performance in restoring fine details [58]. In the work developed by Neves [59], Generative

Adversarial Networks (GANs) were applied to enhance experimental fluid dynamics images. This research showed that incorporating SRGAN into preprocessing pipelines can play a crucial role in noise reduction, artifact removal, optical distortion correction, and improving the differentiation between different phases, all of which are critical in high-speed imaging setups used for two-phase flow analysis.

SR aims to enhance image clarity, sharpness, and detail prior to segmentation, potentially improving model performance in terms of both accuracy and robustness. However, despite promising results in other domains, their application to experimental fluid mechanics, particularly for bubbly flow image segmentation, remains underexplored in the literature.

2.3.3 Multiclass Instance Segmentation

Many of the challenges discussed are caused by physical limitations, either due to the technology used to capture the images or the model's ability to learn complex features under those conditions, however complex tasks can also introduce additional layers of complexity. The task of Multiclass segmentation and classification requires the accurate segmentation of the bubbles' outline, as well as the correct classification of the instance in question, with the need of distinguish between different types, sizes, or behaviors, which may correspond to specific physical attributes or flow regimes. This demands that subtle visual differences be preserved and correctly interpreted by the model despite variability in lighting, image quality, and environmental noise.

Multiclass instance segmentation is therefore the task that combines both object detection and instance segmentation, to identify and classify each individual object in an image, while also assigning every pixel in that object to a specific class. However, in the context of multiphase flow imaging, multiclass segmentation remains a relatively underexplored area. Most studies focus on binary instance segmentation (bubble vs. background), not making a complete analysis that could offer deeper insights into flow characteristics.

The most commonly found applications are in the medical imaging field where complex anatomical structures require fine-grained class separation. Chen et al. (2024) introduced TransUNet, integrating transformer layers with the U-Net backbone to enhance contextual understanding in multiclass medical segmentation [60]. Zhang and Liu (2023) customized SAM specifically for medical segmentation tasks, tailoring its prompts and output layers to accommodate class-specific structures [61]. Based on those two works, Chai et al. (2023) proposed

a Ladder Fine-Tuning strategy for SAM (Segment Anything Model), enabling it to better handle multiple semantic classes by integrating a complementary network [62].

This thesis aims to address the gap found in the literature by creating a pipeline that can be easily altered to train and evaluate different models capable of robust bubble instance and class differentiation for the task of bubble analysis.

Using this pipeline, the ultimate objective would be to compare existing models, ultimately paving the way to the creation of a more tailored approach for bubble multiclass instance segmentation. This could be achieved either by incorporating solutions from the more abundant bubble instance segmentation literature to multiclass scenarios or adapting from other areas like medical imaging to a multiphase flow scenario.

Chapter 3

Models and Metrics

In this chapter the segmentation model architectures selected are described in greater detail as well as the evaluation metrics and validation methods used.

3.1 Segmentation Models

Selecting an appropriate model for the simultaneous tasks of segmentation and classification is crucial to achieving a balance between accuracy, speed, and generalization across varying bubble sizes and imaging conditions.

With the objective of finding architectures capable of the task of multiclass segmentation in different scenarios four model families were chosen based on characteristics like: Segmentation and Classification quality, ease of implementation, computation costs. To this end, four state-of-the-art deep learning model families and slight variations of each family (based on architecture configurations) were used to perform bubble multiclass segmentation:

(i) one-stage, real-time frameworks such as YOLO were chosen due to their quick inference times, community support and popularity. The last official model up to date capable of multiclass segmentation was YOLOv8m-seg with YOLO11m-seg still being officiated and having slight changes in the architecture possibly improving the segmentation outcome;

(ii) two-stage, high-precision detectors like Detectron2's Mask R-CNN with standard backbone ResNet-50 and deeper version ResNet-101. Leveraging a popular architecture Mask R-CNN with good proven results in similar tasks, the model Detectron2 was chosen with its ease of use being

a plus. Being a two stage architecture allowed for comparisons to YOLOs one-stage;

Searching for more precise alternatives transformer-enhanced methods were found. With a hybrid technology combining transformer with other proven architectures, despite requiring an increase in computational costs, the models chosen were:

(iii) TransUNet, combining a popular segmentation architecture U-net with Transformers, creating a segmentation mask with both good fine detail precision and overall image context retention;

(iv) SAM-LST, was chosen due to being based on the really powerful Vision Transformed based SAM model. Combining a ResNet-18 backbone with ViT encoders (ViT-Base and deeper ViT-Large) allowed for the usage of a transformer trained on a immense amount of data, using a clever strategy and fine tunning the model on the target segmentation data through a LST (Ladder-Side Tuning) technique.

By comparing architectures with different characteristics: one vs. two-staged approaches, convolutional vs. transformer backbones, and shallow vs. deep network capacities, this study aims to highlight the key trade-offs and assemble a diverse set of models.

At a high level we can think of these models as a pipeline with three main stages: backbone, neck and head. **The backbone** is the "feature extractor" of the network. It takes the raw image and produces high-level feature maps. Common examples include ResNet-50 / ResNet-101 (convolutional), ViT-Base / ViT-Large (transformer).

The neck is a "bridge" that refines and fuses features from multiple backbone stages. For example, FPN (Feature Pyramid Network) merges low- and high-resolution maps for multi-scale detection, and PANet (Path Aggregation Network) adds bottom-up paths for better localization of small objects.

The head takes fused features from feature maps generated by the Backbone and further processes them to produce the final output of the model in the form of bounding boxes and object classes.

Each of these modules is built by stacking one or more blocks. Blocks are self-contained units composed of several layers that implement a specific pattern of computation. Each layer consists of several neurons (or operations) that perform one transformation on their input before passing it on to the next layer. Layers can thus be seen as 'one' step in the data processing pipeline.

The following section provides a detailed discussion of each model and training decisions so that the future results can be better understood.

3.1.1 You Only Look Once (YOLO)

The two YOLO models used in this study are YOLOv8 and YOLOv11 [41]. Both models offer several specialized variants designed for different tasks, including detection, instance segmentation, pose estimation, oriented object detection, and classification. This wide range of variants highlights the versatility and adaptability of the YOLO architecture for diverse computer vision applications.

Given that the objective of this study is segmentation, the segmentation-specific variants (YOLO-seg) were selected. Each variant is available in five sizes n, s, m, l, and x, which increase in complexity, capacity, and computational demand. Due to hardware limitations, better described in 4.2, the "m" versions were chosen as the optimal balance between performance and resource efficiency. As a result, the models used in this study were YOLOv8m-seg and YOLOv11m-seg.

v8 YOLOv8 [72] builds on the previous versions, improving the backbone and neck structure. It also moves away from traditional anchor-based prediction that simplifies the detection pipeline, adopting an anchor-free “split” head. Anchor boxes are a set of pre-defined, fixed-size bounding-box “templates” tiled over each spatial location in the feature map, and moving away from these improves learning speed. Its architecture can be seen in the Figure A.3.

The backbone alternates between spatial downsampling convolutions and CSP-style C2f(Cross-Stage Partial with feature fusion) modules (combine high-level features with contextual information in order to improve detection accuracy), where each block splits features, sending part through a small bottleneck stack and part straight through, then reunites them to keep gradients flowing and cut extra work. At the deepest level, a fast pyramid-pooling SPPF layer runs several parallel pools, concatenates their outputs with the input, and fuses channels to capture rich, multi-scale context and increase the resolution of the feature maps.

The neck, which tightly fuses features across scales by combining top-down (FPN) and bottom-up (PAN) pathways. The highest-level features (P5) are upsampled and merged with lower features via concatenation and a C2f module several times, and the result of this operation does the sequence in reverse so that both semantic depth and fine spatial detail flow freely

through the network.

The head is decoupled, meaning that it processes objectness, classification, and regression tasks independently. It attaches a small detection subnet at each of the three scales (P3, P4, P5). Each of these fuses features via a few C2f+Conv layers, then splits into two 1×1 convs one predicting box offsets + objectness (anchor-free), the other class scores.

In training, to align the two separate heads so that they can work together during inference, Task Alignment Learning (TAL) is used. This learning strategy approach helps to improve the accuracy of YOLOv8 by aligning the classification and localization scores and optimizing shared representations. This is achieved using supervised learning with a weighted combination of the classification and localization losses. During inference, the network outputs the bounding box coordinates and associated class probabilities for each detected object.

The YOLOv8-Seg model is an extension of the YOLOv8 object detection architecture, adapted to perform semantic segmentation in addition to object detection. Its backbone, neck and detection head remain identical to that of YOLOv8m. YOLOv8-Seg subclasses the standard Detect head to inject two lightweight mask components: a small fully convolutional Proto module that generates a fixed set of low-resolution 'prototype' masks, and a cv4 ModuleList of compact conv blocks (two 3×3 Conv→BN→ReLU layers plus a 1×1 Conv) that predicts for each detected object, a vector of mask coefficients. During inference, the model runs Detect.forward() to get boxes and labels, computes mask coefficients via those cv4 blocks, and then linearly combines them with the shared prototypes to assemble high-quality per-instance masks no bulky transposed convolutions or softmax stages required. The result is a unified network that delivers both state-of-the-art object detection and precise instance segmentation in a single, real-time-capable pass.

v11 YOLOv11 [73] [42] [74] builds upon the YOLOv8 architecture, maintaining its core design principles while improving efficiency without discarding the strengths of its predecessor. One of the main contributions is the introduction of the C3k2 block, which replaces the C2f block used in YOLOv8 in all architecture components. The C3k2 block employs two smaller kernel-sized convolutions instead of one large convolution, contributing to faster processing and a more computationally efficient implementation.

In YOLO11 backbone and neck, the Cross Stage Partial (CSP) Bottleneck is built similarly to YOLOv8 but with C3k2. This component's structure also retains the Spatial Pyramid Pooling

- Fast (SPPF) block from YOLOv8 and introduces a new Cross Stage Partial with Spatial Attention (C2PSA) block after it. C2PSA is an attention mechanism that improves spatial attention in the feature maps, allowing the model to focus more effectively on important regions within the image and potentially improving detection accuracy for objects of varying sizes and positions, especially for smaller or partially occluded objects.

In the head, the C3k2 blocks function with flexibility to process multi-scale features at different depths. This block can behave similarly to the C2f block or be replaced by a C3 module, which allows for deeper and more complex feature extraction depending on the c3k parameter. Following the C3k2 blocks is a CBS (Convolution-BatchNorm-Silu) layer to refine the feature maps. This layer extracts the relevant features, stabilizes and normalizes data flow, and improves model performance with a Sigmoid Linear Unit (SiLU) activation function for non-linearity. The neck ends with a set of Conv2D layers after each detection branch.

These architectural refinements collectively optimize the model for both inference speed and representational power, enabling more efficient feature extraction without significantly increasing model complexity. For segmentation, YOLOv11-seg mirrors the structure of YOLOv8-seg while incorporating the enhancements introduced in YOLOv11.

In YOLOv11-seg, segmentation is performed using a dynamic mask generation approach inspired by YOLACT [77]. Building on the detection pipeline, the segmentation variant introduces a fully convolutional "ProtoNet" head that upsamples and processes the fused feature maps into a fixed set of k prototype masks. Simultaneously, the detection head outputs per-instance mask coefficients, alongside bounding box coordinates and class scores. The final instance masks are generated by linearly combining the prototype masks with each object's coefficients, followed by cropping to the predicted bounding box and applying a threshold. This strategy enables real-time instance segmentation while maintaining high mask quality [74].

3.1.2 Detectron2

Detectron2 is Facebook AI Research's library that provides state-of-the-art detection and segmentation algorithms [40].

It utilizes the Mask R-CNN architecture, a two-stage object detection and instance segmentation framework that builds upon Faster R-CNN. This model combines region proposal generation with classification, bounding box regression, and pixel-level mask prediction in a

modular architecture.

In this study, two variants of Mask R-CNN from the Detectron2 model zoo were used: mask_rcnn_R_50_FPN_3x and mask_rcnn_R_101_FPN_3x. Both models share the same overall architecture, consisting of a Region Proposal Network (RPN), Feature Pyramid Network (FPN), and parallel heads for bounding box regression, classification, and instance segmentation. The key difference is the depth of the backbone network used for feature extraction.

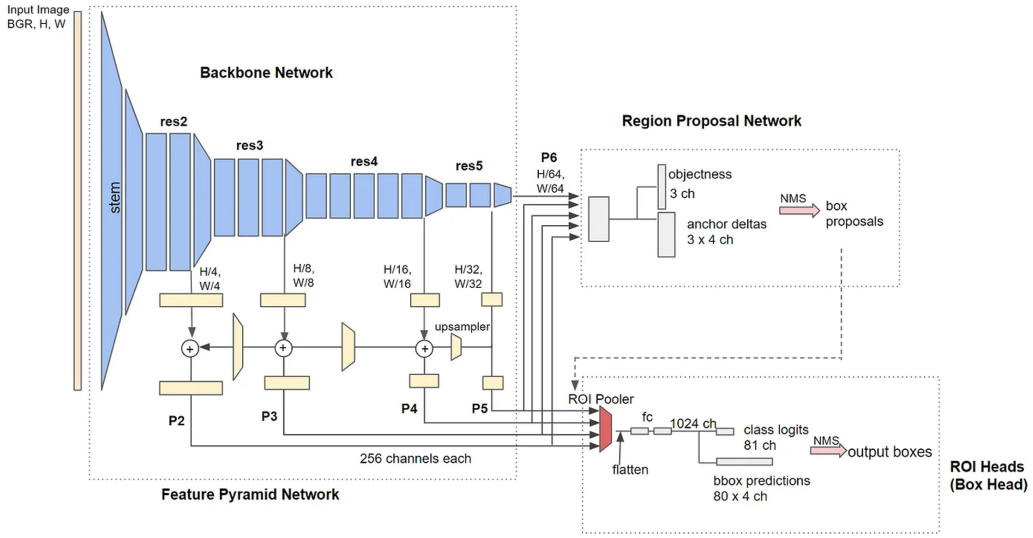

Figure 3.1: Detectron2 Base R-CNN FPN framework. [66]

ResNet, the Backbone of Detectron - Figure 3.1, is a deep convolutional neural network architecture originally introduced by He et al. in 2015 [25]. Compared to previous architectures, it introduced residual connections, allowing the network to learn identity mappings and preserve gradient flow during backpropagation, training much deeper networks without performance degradation or vanishing gradients. ResNet is composed of a series of residual blocks, each containing multiple convolutional layers and a skip connection that adds the input of the block to its output. The number in the model name refers to the total number of layers: ResNet-50 has 50 layers, while ResNet-101 has 101 layers, making it substantially deeper. This added depth allows ResNet-101 to extract more abstract and detailed features, which can be beneficial in tasks involving fine object boundaries, small structures, or complex textures.

The output from various ResNet layers is fed into the neck component, a Feature Pyramid Network (FPN) which constructs a top-down feature hierarchy with lateral connections by combining high-level semantic maps with lower-level, higher-resolution maps via lateral 1×1 convolutions and upsampling with element-wise fusion. The FPN generates a set of multi-scale

feature maps (typically 5 scales: P2 to P6), each of which is used for region proposal and further downstream processing, providing a rich representation that allows the model to detect small and large objects with improved accuracy. This design enables the model to leverage both high-resolution spatial features and deep semantic information, facilitating robust object detection across scales.

The Head of the architecture is composed of three parallel branches: The first is the Region Proposal Network (RPN), which is built on top of the FPN outputs and attached at each pyramid level. It scans each level with a small convolutional head to predict objectness scores and bounding box deltas for a set of predefined anchors at every spatial location. Objectness scores are scalar values that estimate how likely it is that a given proposal or anchor contains an object. The second is the Detection Head, which refines the candidate regions proposed by the RPN. To achieve this, it uses RoIAlign, a technique introduced in Mask R-CNN to extract fixed-size features from the FPN feature maps for each proposal (Region of Interest, or RoI). Then, fully connected layers are applied to perform object classification and bounding box regression. Proposals with the highest scores are selected using non-maximum suppression and passed to the ROI heads. The third and final branch is the Segmentation Head (Mask Head), which predicts a binary mask for each Region of Interest (RoI). Unlike semantic segmentation, this head outputs a distinct mask for each object instance, enabling instance-level segmentation.

For the segmentation task, Mask R-CNN operates in a region-based manner: after detecting the bounding boxes, each RoI is aligned using RoIAlign, which preserves spatial alignment of features. These aligned features are passed through a small fully convolutional network (FCN) that outputs a pixel-wise mask (typically 28×28) for each object instance. The final instance mask is then resized and positioned according to its associated bounding box.

This architecture enables precise instance-level segmentation, with strong performance on benchmarks like COCO and Cityscapes. Compared to single-stage models like YOLOv8, Mask R-CNN is generally slower but more accurate, particularly for tasks requiring detailed boundary delineation or multiple overlapping objects.

3.1.3 TransUNet

TransUNet [63] model is a hybrid vision architecture that combines the widely adopted U-net architecture with a transformer-based encoder into its framework.

The U-Net [16] is an architecture originally proposed for biomedical image segmentation with an encoder - decoder structure, combined with skip connections that fuse low-level spatial details with high-level semantic features, making it particularly effective for dense prediction tasks. U-Net's simplicity, efficiency, and strong performance have led to numerous adaptations and extensions in various segmentation challenges.

Despite being highly effective at learning visual features, CNN-based approaches like U-Net have the drawback of being limited in their ability to capture long-range dependencies within an image due to the local nature of convolution operations. This often results in a struggle to accurately segment structures that vary significantly between different instances in terms of shape, size, or texture. To address this, some studies have introduced self-attention mechanisms like Transformers on top of CNN features. These allow a model to focus on different regions of an input simultaneously, enabling it to capture both local and global dependencies.

Transformers, originally developed for natural language processing tasks by [43], have recently gained significant attention in computer vision by eliminating the need for convolutional operations and relying solely on attention mechanisms to model global dependencies. Their capacity to capture long-range contextual relationships, combined with the advantages of large-scale pretraining, has enabled Transformers to achieve strong performance and generalization across a wide range of vision tasks, often surpassing traditional CNN-based architectures. However, these benefits come with notable drawbacks: Transformers are computationally intensive, require large annotated datasets for effective training, and may struggle to preserve fine-grained spatial details, which are critical in tasks such as medical image segmentation.

To address the limitations of CNNs, TransUNet introduces a ViT-based encoder into the U-Net framework, leveraging both global context modeling and precise spatial localization -Figure 3.2. Much like a U-Net, its architecture can be divided into an encoder-decoder structure with the following specifications:

The **encoder** begins with a standard convolutional stem that downsamples the input image and extracts low-level features. These features are flattened and embedded into a sequence of tokens, which are then passed to a Vision Transformer (ViT) module. The transformer processes the token sequence through multi-head self-attention and feed-forward layers, enabling the model to capture long-range dependencies and contextual information across the entire image. To retain spatial information typically lost during token flattening, position embeddings are added to each token before transformer processing.

The **decoder** follows a U-Net-style upsampling path. It consists of a series of convolutional and upsampling blocks that reconstruct the spatial resolution of the feature maps. Critically, skip connections from the early convolutional layers (before tokenization) are used to inject fine-grained spatial details into the decoder. These skip connections help mitigate the loss of resolution caused by the flattening operation in the transformer and restore detailed anatomical structures.

To produce the final segmentation mask, the decoder outputs are passed through a final convolutional layer followed by a softmax activation. This allows TransUNet to generate pixel-wise class probabilities. By combining the global modeling power of transformers with the local precision of CNNs, TransUNet achieves high performance in medical image segmentation tasks, particularly in organ and lesion segmentation, where both fine structure and contextual awareness are crucial. [43]

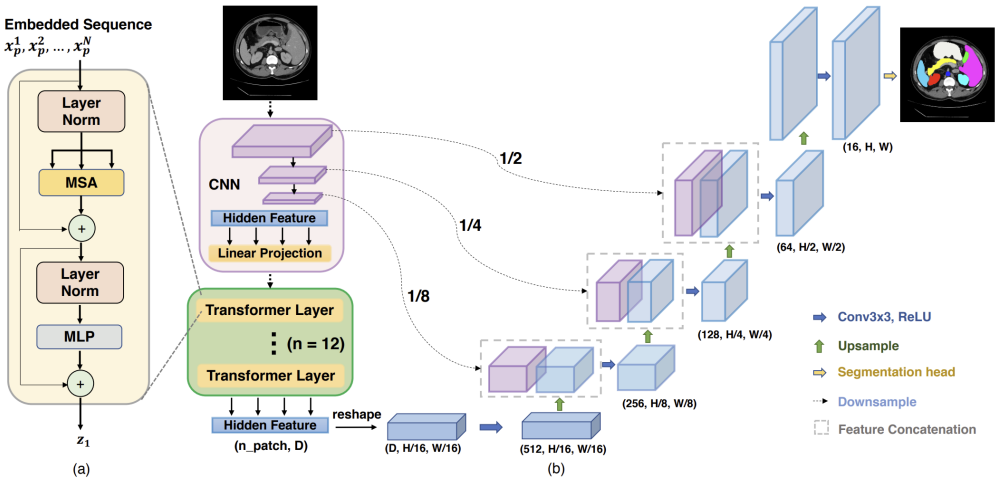

Figure 3.2: TransUNet framework. (a) schematic of the Transformer layer; (b) architecture of the proposed TransUNet. [63]

The vit backbone implemented was R50+ViT-B_16. It is a two-step feature extractor that fuses the first four stages of a ResNet-50 convolutional network with a 12-layer Vision Transformer (ViT-B_16) encoder. The ResNet-50 uses convolutional layers to shrink the image down and extract features that are broken into a sequence of tokens and sent through a 12-layer Vision Transformer (ViT-B_16), whose output tokens are reshaped back into a feature map. The global features, together with the ResNet maps saved along the way, go into the U-Net decoder to reconstruct a detailed segmentation. A deeper implementation was attempted, but without R50+ViT-L_16 pretrained weights, this task was made more difficult. Merging pretrained data

from an R50 and ViT-L_16 was attempted, but not successful.

3.1.4 Segment Anything Model Ladder-Side Tuning (SAM-LST)

To understand the SAM model variant used a comprehensive understanding of how SAM works is needed. It follows a three-module design with an Image Encoder, a Prompt Encoder, and a Mask Decoder, each designed to operate independently, allowing the model to be scalable and prompt adaptive - Figure 3.3.

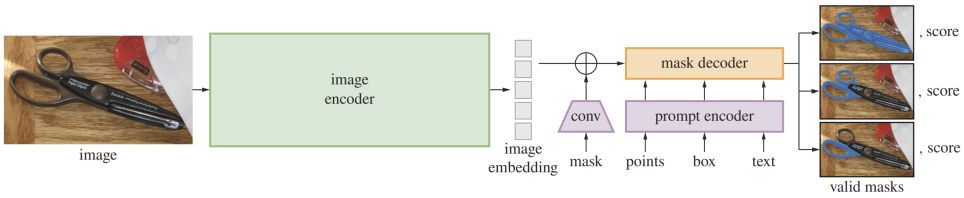

Figure 3.3: SAM architecture overview. [46]

The **image encoder** is a high-capacity Vision Transformer (ViT) pretrained with Masked Autoencoding (MAE), designed to extract rich and scalable features from input images. It processes the image once per inference cycle, producing a 64×64 spatial embedding with 256 channels. This embedding is obtained by passing the 1024×1024 input through windowed self-attention and global attention blocks, followed by convolutional layers with layer normalization to reduce dimensionality while maintaining spatial fidelity.

The **prompt encoder** transforms user inputs such as points, boxes, or masks into 256-dimensional vector embeddings. The sparse geometric prompts are positionally encoded and combined with learned tokens to indicate foreground/background or corner positions. Dense prompts (e.g., masks) are downsampled through convolutional layers and mapped to the same 256-dimensional space to align with the image embeddings. These prompt embeddings serve as queries that condition the mask generation, allowing SAM to flexibly respond to a wide variety of prompt types during segmentation tasks.

The **mask decoder** is a lightweight Transformer-based module that fuses the image embedding with prompt embeddings to produce binary segmentation masks. It consists of two Transformer layers that perform self-attention among tokens, cross-attention between prompts and image features, and feed-forward updates with residual connections and layer normalization. Positional encodings are re-injected at each attention step to preserve spatial context. After

decoding, the image embedding is upsampled using transposed convolutions, and the final segmentation mask is computed by a spatial dot product between the upsampled image features and the MLP-transformed output token. This design enables fast, prompt-adaptive segmentation suitable for real-time interactive applications.

Despite being trained with a huge array of images, applying the SAM model directly does not result in satisfactory performance. Inspired by the work developed by Sung et al with a ladder side tuning network for Transformers [75] and using SAM as a base, Ladder-Side Tuning (LST)-SAM was created [62] - Figure 3.4.

Ladder-Side Tuning (LST) is a lightweight fine-tuning strategy designed to adapt large pretrained models to downstream tasks without updating the entire network. Instead of modifying the main backbone, LST uses an auxiliary network alongside the frozen pretrained model. This side network learns task-specific representations and feeds them into later layers of the main architecture, being an efficient adaptation while preserving the general-purpose features learned by the original model. This approach significantly reduces training time and computational cost, making it ideal for scenarios with limited data or resources, as it enables the flexible integration of additional networks while avoiding backpropagation on the entire model. A pretrained ResNet18 [76] was added as an additional network and was slightly modified for compatibility to output feature maps with the same spatial dimensions as the encoder.

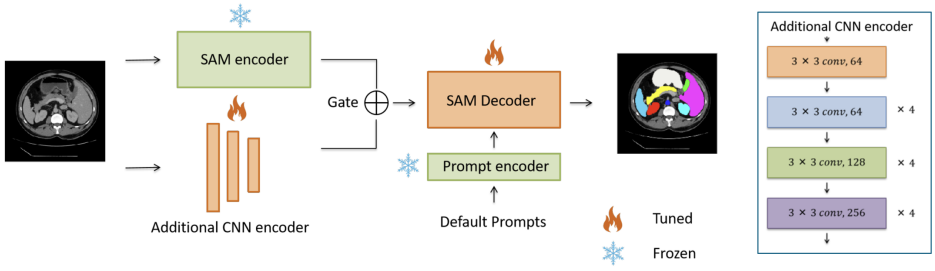

Figure 3.4: SAM LST architecture overview and additional CNN encoder with activation functions, batch normalisation layers, and residual connections omitted for simplicity [62]

As seen in the architecture scheme, the input image is first fed into the SAM image encoder and CNN encoder simultaneously. During training, only the parameters of the additional CNN and SAM Decoder parts are fine-tuned while keeping the original SAM encoder parameters frozen.

Summing up the main characteristics previously mentioned of the models used in this work, the following table 3.1 was compiled.

Table 3.1: Summary of the Implemented Models and Their Characteristics

Model	**Positive Characteristics**	**Negative Characteristics**
YOLO-Seg	- One-stage architecture - Fast inference times - Straightforward use	- Less accurate on fine boundaries - Performance degrades with small dense objects
Detectron2	- Two-stage architecture good accuracy. - Straightforward use - Good general object localization	- Slower inference times than YOLO than one-stage models - Region-based method can struggle with extremely small
TransUnet	- Combines Unet CNN + ViT transformer - Efficient encoder–decoder structure - Best spatial detail preservation	- High memory usage - Transformers require extensive computation
SAM-LST	- Leverages SAMs' pretrained ViT encoder - LST enables adaptation with fine-tuning - Good performance despite limited training data	- Limited generalization without fine-tuning - High memory usage - Complexity increases due to dual-encoder design

3.2 Evaluation Metrics and Validation Methods

Evaluation metrics play a vital role in assessing the performance of segmentation and detection techniques. With the objective being the segmentation and classification of an image with bubbles of different classes as the target, the metrics chosen to evaluate the performance of the model used can be separated into two categories: Global metrics and Class metrics. The global metrics provide a general overview of the model's performance across the entire dataset and class metrics help to understand and reveal the model's performance for each category, so a deeper analysis can be performed, for example, checking if the model is biased toward dominant or easily detectable classes.

In order to assess how well each model captures the boundaries and maintains the label consistency at pixel scale, several pixel-level metrics were calculated including: Accuracy (proportion of correctly labeled pixels); Precision (Fraction of correctly predicted pixels among all

predicted positive pixels); Recall (Fraction of correctly predicted pixels among all actual positive pixels); F1 Score (Harmonic mean of precision and recall, suitable for imbalanced datasets); IoU; Mean Squared Error (MSE); and Mean Intersection over Union (mIoU); Dice Coefficient (similar to IoU, focusing on overlap between predicted and true masks) [70].

Instance-level metrics were used to ensure a complete picture of the segmentation and classification results.

A custom implementation of Average Precision (AP) at IoU=0.50 and mean Average Precision (mAP) across IoU thresholds (0.50 - 0.95) is applied by matching predicted instances to ground truth on a per-image basis.

For this purpose, a custom implementation of Average Precision at IoU 0.5 (AP50) was used, matching predicted instances to ground truth on a per-image basis by counting a detection as a true positive only if its Intersection-over-Union (IoU) with a ground-truth box is at least 0.50. This metric therefore measures how well the model finds objects when allowing a relatively loose overlap threshold. To provide a more robust evaluation, mean Average Precision (mAP) across IoU thresholds (0.50 - 0.95) was also calculated. The added IoU thresholds result in a greater penalization of models with imprecise masks [71]. All these metrics' functions are described in more detail in the Appendix A.1.

In many two-phase flow images, the number of background pixels (non-bubble) can vastly outnumber the bubble pixels, creating class imbalance. In such cases, **accuracy** alone can be misleading as it simply predicts "no bubble" most of the time. Despite appearing to have a high accuracy, it can fail to detect important bubble regions. Therefore, **precision**, **recall**, and especially their harmonic mean **F1 Score** are highly relevant. High precision indicates that when the model predicts a bubble pixel, it is usually correct, which is critical for avoiding spurious segmentation. High recall suggests that the model successfully captures most of the actual bubble regions, reducing the likelihood of missing bubbles. Since segmenting bubbles accurately is important for subsequent calculations (such as bubble count, size distribution, or shape factors), the **F1 score** offers a balanced way to evaluate both aspects (precision and recall), which is especially useful for imbalanced datasets.

In addition, **Intersection over Union (IoU)** and **Dice Coefficient** measure how well the predicted and true masks overlap, offering more nuanced insights into the quality of the segmentation than pixel classification metrics alone. For bubble images, small misalignment or

boundary errors can significantly affect derived measurements like bubble diameter or centroid position.

Lastly, when quantitative metrics indicate high performance,**visual inspection**, overlaying predicted masks on the original images, can reveal edge cases where the model fails. This step can also help identify problems with contrast variations, illumination, or optical distortions commonly seen in experimental two-phase flow setups.

Chapter 4

Methodology

This chapter presents a detailed description of the dataset and tools used in the study and outlines the training, inference, validation, and bubble-characterization pipelines developed.

4.1 Dataset

The dataset used in this study comprises 213 experimental bubbly flow images acquired in a controlled laboratory setup using high-speed imaging equipment, with a resolution of 640 × 480 pixels, stored in .jpg files and corresponding hand-annotated label masks done by 3 people in YOLO .txt format. These images capture gas-liquid interactions under varying flow conditions. In this work, four bubble classes were considered: Bubbles, which correspond to larger, well-defined structures; Spherical, which are smaller in size and with a nearly spherical shape; Agglomerated, representing groups of connected or partially merged bubbles; and Cut, which are only partially captured at the edges of the image - Figure 4.1. The classes are moderately balanced with 3020 bubbles, 2277 spherical bubbles, 1636 cut bubbles, and 1403 aggregated bubbles.

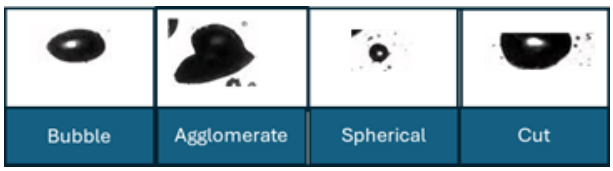

Figure 4.1: Example of the four bubble classes bubbles, spherical bubbles, agglomerated bubbles, and cut bubbles at the image borders

As mentioned in the previous chapters, a common challenge that can affect the effectiveness of the dataset is image quality, with low-resolution input images having a significant impact. Regardless of a model's inherent precision, poor-quality training data, where phase boundaries are blurred and critical features for classification are lost, inevitably leads to suboptimal training outcomes. As a continuation of the work developed by [59], the effect of SR GAN techniques on training performance was assessed by duplicating the original dataset and enhancing one of the copies with a BSRGAN. This setup allowed training the same model under identical conditions, in the same pipeline, using both the original dataset and the enhanced version. Figure 4.2 provides an example of the application of SR.

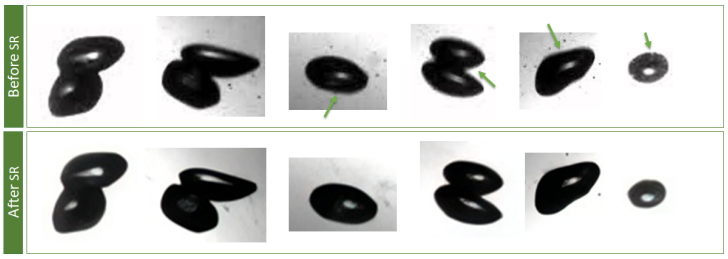

Figure 4.2: Comparison of bubble images before and after super-resolution (SR) enhancement

As observed in the enhanced version, SR sharpens the overall image and effectively reduces blur and noise, resulting in more defined edges of the bubbles and improved contrast across the scene. However, it is important to note that while SR performs well in improving the delineation of medium and large bubbles, in some cases, the enhancement process may introduce slight distortions or over-smooth fine structures, potentially affecting the precise identification of smaller-scale features.

4.1.1 Dataset train-test split

The dataset was randomly split into training (80%) and test (evaluation) (20%) subsets to support model training and performance evaluation. The same split was applied to the original and SR enhanced datasets, ensuring that the corresponding images in each version shared identical indices. In other words, the training sets contained the same images, original in one case and enhanced in the other, allowing for a fair comparison under identical conditions.

4.2 Tools, Libraries & Computational Resources

The development and evaluation of deep learning models for image segmentation in this thesis was done using the programming language **Python**. Its versatility, scientific computing ecosystem and widespread adoption in the machine learning and computer vision communities made it the obvious choice.

At the foundation of the model implementations PyTorch was used due to the dynamic computation graph of this framework, greater flexibility in model customization, widespread use in academic research, and compatibility with the external libraries required for this project. For data manipulation and numerical operations, NumPy and Pandas were used extensively, while visualizations of both data and results were created using Matplotlib and Plotly.

Different models required specific libraries. For the **YOLO** models, training and inference were performed using the Ultralytics library, which offers a streamlined interface for YOLOv8 and its variants [41]. For the **Detectron2** Mask R-CNN architectures, the official Detectron2 framework from Facebook AI Research (FAIR) was used [40]. For the **TransUNet** model, the architecture was built upon a Vision Transformer (ViT) backbone with custom configuration dictionaries and helper utilities such as np2th for weight conversion based on Chen et al. implementation [63]. For the **Segment Anything Model** variant SAM-LST, similarly to TransUNet, a custom dataset class was used largely inspired by the work in [62].

Finally, several Python libraries also played a vital role in the project. These included TQDM for progress monitoring, **os**, **glob**, and **pathlib** for file management, **importlib** for dynamic imports and specific utilities from **Google Colab** given where the code was run mainly. All these tools and libraries enabled the efficient creation and implementation of complex pipelines used to support the thesis objectives.

As for the computational resources, all the models were trained using NVIDIA Tesla T4 provided by Google Colab. Despite the strict running time limitations, not being optimal due to frequent interruptions, this environment provided enough computational power necessary to run all the models while being responsive without the need for local high-performance hardware.

4.3 Pipelines Developed

The pipelines developed were built on a common logic to ensure consistency and reproducibility across the experimentation process, simplifying the orchestration of training workflows. A common theme is that all pipelines begin with a clear and consistent directory structure, rooted in a central base directory located on Google Drive. This promotes systematic experimentation and ensures compatibility with the subsequent validation, visualization and post-processing tools.

4.3.1 Training

A consistent naming convention is set on the training procedure and carried throughout the other pipelines, composed, for example, of the model name and backbone type, the datasets trained on, and loss functions. This allows for the specific model Results directories to be created and seamless switching between experimental conditions. In this naming convention, all variables are defined allowing one to choose, in the beginning, the model version one wants to train and simply run the cells bellow while waiting for the output in the Results directory.

The environment setup starts with Google Drive mounting using the google.colab interface to access shared resources. Then comes the verification and installation of the required libraries such as PyTorch, and model-specific dependencies, ensuring that each model runs in its optimal configuration. In some pipelines, version-specific installation commands are included (e.g., fixed pyyaml or h5py versions) to avoid compatibility issues. Even though different libraries and frameworks are used (e.g., Ultralytics for YOLO, Facebook's Detectron2, or custom PyTorch classes for SAM-LST and TransUNet), all scripts follow the same structure of isolating the installation block, checking versions, and dynamically adjusting system paths to load local modules (especially in the TransUNet and SAM pipelines).

As for the dataset metadata types and its handling, in YOLO and Detectron2 pipelines, the number of segmentation classes is retrieved from a data.yaml file. For SAM and TransUNet, dataset paths and parameters are derived from .npz files or explicitly set variables. This approach is necessary since different models require specific data parsing configurations, ensuring that all pipelines are capable of dynamically adapting to different datasets, class labels, and preprocessing settings without manual rewriting of code blocks.

Next in the pipeline is Data Loading. Data loading is implemented slightly differently depend-

ing on the framework: YOLO using built-in Ultralytics logic, Detectron2 using DatasetCatalog, SAM and TransUNet using custom PyTorch Dataset classes for .npz files. However, all pipelines share a systematic approach starting with defining the training and validation loader as well as the appropriate batch_size, shuffle, and num_workers settings.

Some models require a preprocessing step to optimize the models capabilities and ensure that input shapes and formats are consistent with model expectations (e.g., resizing or padding images if needed). Notably, several pipelines perform dynamic image resizing and label conversion to standardize dimensions (such as 1024×1024 pixels) across all experiments.

At this point, based on the naming convention set in the beginning and other key hyperparameters, the run_name is defined making the pipeline ready to enter the training phase, where the model is initialized and optimized over successive epochs.

Each model variant was trained for the equivalent of 75 epochs with an appropriate learning rate and loss function. The number of epochs was chosen based on the training/validation loss epoch progression, with this value being a compromise between convergence and stabilization of the loss and running time/memory usage. As for the learning rate through an iteration process this value was optimized as used for all models. Finally, regarding the loss function, SAM-LST and TransUnet used the same loss function. As for YOLO and Detectron2, their native loss functions were already optimized for each model architecture, and changing them to a common loss function was not possible, as it negatively affected the results due to incompatibility issues with the architecture. Since the 4 architecture types capture the details differently, they require slightly different hyperparameter settings for each. The training was performed using the same labeled mask once with the original images input and a second time with the enhanced images, finalizing with the weights for each training configuration being saved. In every case, results such as loss curves, accuracy metrics, and trained weights are saved to structured folders under a Results root directory. This is crucial for maintaining traceability of results and enabling later comparison between models.

4.3.2 Testing

The same naming convention was used allowing for the selection of the model to be tested in the first cell and simply running the testing pipeline. All metrics and inference examples are saved automatically to the output Testing Results folder. Another key feature is that the metrics

calculation and exportation step is independent of the model being tested allowing for a fair comparison between models. To achieve this the run_name set in the training pipeline is defined, the directories are loaded and the required libraries are imported, followed by the configuration of data loading routines. Then, the models are built mirroring the training pipeline, followed by an inference pass per image is done across the validation dataset. The result is a dictionary with all instance masks, classes, confidence scores which can be combined in a semantic mask for each image in the validation folder. The output at this stage is model-independent, meaning that all subsequent processing steps are applied uniformly, regardless of the chosen architecture.

Due to some of the model's architecture features, all generated segmentation masks pass through a post-processing function. This function's objective is to make sure the mask is correct by checking if there is more than one pixel classification within the same bubble instance. In SAM-LST this step was essential to correct the masks. The function ultimately returns a list of instance masks, their associated classes, confidence scores, and pixel composition properties.

The inference masks are checked side by side with the original and the validation loop is initiated. The loop begins by initializing containers to collect pixel-level and instance-level metrics. For each image in the validation set, the ground truth mask and the predicted segmentation output are loaded and compared. After all images are processed, global and per-class metrics are computed and printed. Results are also exported to an Excel file with separate sheets for global summaries and per-class performance, facilitating downstream analysis and documentation.

In summary, while each model-specific pipeline incorporates necessary adaptations to accommodate the nuances of its respective architecture and framework, the underlying structure, data handling, evaluation methodology, and logging practices remain intentionally uniform. This unified approach was essential to achieve a rigorous and fair evaluation of segmentation performance across diverse deep learning models applied to experimental image data.

4.3.3 Model specific

YOLO based installation and setup are minimal, and once the environment is prepared, training can be launched setting the hyperparameters. While this facilitates rapid deployment and ease of use, it can limit customization at lower levels. The dataset was organized in a specific folder structure required for the YOLO model to work. The training was performed with YOLO's native Adam optimizer and loss function and other metrics computed by the ultralytics framework.

The loss function was composed of four terms, one for each output of the model predictions (bounding box regression, classification confidence, refined localization, and instance segmentation mask): $\mathcal{L}_{\text{box}} = \text{CIoU}(b, b^*)$, $\mathcal{L}_{\text{cls/obj}} = \text{BCE}(\hat{p}, p^*)$, $\mathcal{L}_{\text{dfl}} = \text{Distribution Focal Loss}$, $\mathcal{L}_{\text{mask}} = \text{BCE}(M, M^*) + \text{Dice}(M, M^*)$. The total loss is a weighted sum of these terms:

$$\mathcal{L}_{\text{YOLO}} = 7.5\,\mathcal{L}_{\text{box}} + 0.5\,\mathcal{L}_{\text{cls/obj}} + 1.5\,\mathcal{L}_{\text{dfl}} + 4\,\mathcal{L}_{\text{mask}}. \tag{4.1}$$

where b and b^* are the predicted and ground-truth bounding boxes; $\hat{p}$ and p^* are the predicted and true class probabilities; M and M^* are the predicted and ground-truth instance masks.

As for validation, it begins with the definition of run-specific parameters in order to access the weights of the model corresponding to those parameters. The environment is configured with the libraries and model building, and then, using the run name, the best model checkpoint weights are loaded based on the training run identifier. The test images and corresponding ground truth labels are retrieved from the predefined dataset directory, with labels formatted according to the YOLO segmentation standard. The next step is the inference phase, followed by a visualization step, culminating with the test pipeline described previously.

Detectron2 implementation was similar to YOLO in procedure and also in the sense that it was not too complex but lacked some flexibility. The necessary libraries are imported, the directories and model parameters are defined and data was loaded from the output_coco_train.json file made from the YOLO train dataset folder. The model is built and pretrained weights available in the repository are loaded, and finally, the model is trained.

The loss function predetermined by Detectron2 libraries was composed of 5 loss terms (RPN objectness, RPN box regression, RoI classification, RoI box regression, instance segmentation): $\mathcal{L}_{\text{rpn\_cls}} = \text{CE}(p_{\text{rpn}}, p^*_{\text{rpn}})$, $\mathcal{L}_{\text{rpn\_loc}} = \text{SmoothL}_1(\Delta_{\text{rpn}}, \Delta^*_{\text{rpn}})$, $\mathcal{L}_{\text{roi\_cls}} = \text{CE}(\hat{c}, c^*)$, $\mathcal{L}_{\text{roi\_box}} = \text{SmoothL}_1(\Delta_{\text{roi}}, \Delta^*_{\text{roi}})$, $\mathcal{L}_{\text{mask}} = \text{BCE}(M, M^*)$ and the total training loss

$$\mathcal{L}_{\text{Detectron}} = \mathcal{L}_{\text{rpn\_cls}} + \mathcal{L}_{\text{rpn\_loc}} + \mathcal{L}_{\text{roi\_cls}} + \mathcal{L}_{\text{roi\_box}} + \mathcal{L}_{\text{mask}}. \tag{4.2}$$

where p_{obj} and p^*_{obj} are the predicted and anchor-level ground-truth objectness scores; t and t^* are the RPN's predicted and target box deltas; $\hat{p}$ and p^* are the RoI classification logits and labels; d and d^* are the Fast R-CNN box-regression deltas; and M and M^* are the predicted and ground-truth masks.

Each full training run comprised of 6374 iterations. A batch training implementation was required to surpass computational limitations, and the number of iterations was calculated based on (number of images over batch size) times desired epochs in order to correspond to the 75 epochs used in the other models. The model weights of the trained model were saved, as well as the training metrics progression for further analysis. As for inference and validation, the steps were the same as previously described, with the data being loaded from output_coco_validation.json instead, and in a similar fashion, inference and its metrics were obtained.

TransUNet The training pipeline implemented for the TransUNet [60] architecture was designed to integrate Vision Transformer-based encoders into a semantic segmentation framework while maintaining compatibility with structured datasets and experiment management conventions. To enable modular reuse of code, the core scripts from the GitHub [60] were loaded dynamically into the runtime environment using Python's importlib. This method bypasses the need for command-line execution, enabling flexible experimentation directly from notebook environments.

The dataset consisted of .npz files created from the YOLO dataset containing RGB image arrays and corresponding label masks. Through a custom PyTorch Dataset class, the files are loaded, normalization is applied, and joint image-mask transformations performed. All images were resized to a uniform resolution.

Due to limitations in GPU memory, the DataLoader instances were initialized with four workers for parallel loading, and a batch size of 4. Before training R50+ViT-B_16 pretrained weights were fetched (which automatically rescaled the transformer's 14×14 positional grid to our 32×32 feature map).

Training ran for 75 epochs employing a composite hybrid loss function with Cross-Entropy (CE), ensuring each pixel's predicted class probability aligns with the ground truth by penalizing wrong confidences, Dice (D) used to directly maximize overlap between predicted and true masks mitigating class imbalance, and Tversky (T) to extend from Dice with weights α, β on false positives vs. negatives to control over vs under-segmentation

$$\begin{aligned}\mathcal{L} = \lambda_{\text{edge}}\Big(\Big(-\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\,\log p_{i,c}\Big)_{\text{CE}} + \Big(1 - \frac{2\sum_{i=1}^{N} p_i\,y_i + \varepsilon}{\sum_{i=1}^{N} p_i + \sum_{i=1}^{N} y_i + \varepsilon}\Big)_{\text{Dice}}\Big) \\ + \lambda_{\text{class}}\Big(1 - \frac{\sum_{i=1}^{N} p_i\,y_i + \varepsilon}{\sum_{i=1}^{N} p_i\,y_i + \alpha\sum_{i=1}^{N} p_i(1-y_i) + \beta\sum_{i=1}^{N}(1-p_i)\,y_i + \varepsilon}\Big)_{\text{Tver}}\end{aligned} \tag{4.3}$$

A study of the best loss function weight combination of the three components was performed with the best results occurring for the weight values $\lambda_{\text{edge}} = 0.75, \lambda_{\text{class}} = 0.25$. Training was done with AdamW optimizer (lr $= 1 \times 10$, weight decay $= 1 \times 10^2$) via a trainer_synapse routine present in the GitHub repository and adapted to this case.

The model was periodically evaluated using validation loaders, and checkpoints were saved within a structured results directory under Results/Training, saving both the final and best checkpoints and a training log. This pipeline, while based on externally sourced implementations, was heavily customized to integrate into a reproducible, scalable experimentation framework, capable of handling multiple dataset variants and architectural configurations.

SAM-LST was trained with two variations ViT-B (Base) with fewer parameters and layers and ViT-H (Huge), with a deeper architecture. The pipeline required certain adaptations to enable successful training, particularly due to its architectural complexity and compatibility limitations when used in custom environments such as Google Colab.

The light configuration relied on the ViT-B backbone. This is natively supported by the SAM architecture, thereby avoiding the need for modifications. This version emphasized simplicity and robustness: model instantiation used the default registry mechanisms, and no manual patching of the model source code was required.

In contrast, the deeper SAM-LST pipeline used a more complex encoder-ViT-L (Vision Transformer - Large). Since this backbone is not directly supported in the training context of the original SAM implementation, substantial code-level interventions were necessary. The internal model registry was patched using regular expression-based substitutions in the deeper_sam_LST.py file to force the architecture to recognize and initialize ViT-L. A compatible pretrained checkpoint (sam_vit_l_0b3195.pth) was manually downloaded and injected into the pipeline outside of the standard model loading workflow. Some compatibility issues appeared with dependencies such as h5py having to be reinstalled with specific flags to prevent package conflicts within the Colab environment to allow successful execution and model loading of the ViT-L backbone.

ViT-B was trained in the image resolutions 512×512 pixels and 1024×1024 pixels. This second resolution was needed to enable fair comparisons with ViT-L since this larger model was fixed to the resolution of 1024×1024 pixels. The lighter model configuration served as a reliable and adaptable baseline for development, while the deeper pipeline was reserved for high-resolution, performance-focused training runs that required greater computational resources.

Besides model loading, despite the differences both model configurations followed the same logic for data preparation, training routines, and loss evaluation, employing a composite hybrid loss the same as used in TransUNet 4.3 with the same weights.

4.3.4 Bubble Analysis Pipeline

Based on the raw inference results, a bubble analysis pipeline was developed. It calculates bubble analytical metrics that better describe the mask allowing for consistent and clear comparison of how different classes are being segmented and how models affect mask generation in images. This workflow was applied systematically to the best segmentation model variants revealing strengths and weaknesses, evaluating their ability to preserve physically relevant features and in the future potential allowing for correlation with heat/mass transfer metrics.

The first step is a normalization of all segmentation masks to a common resolution. Because different models may take different input sizes, and thus produce masks at varying resolutions, we resize every output mask to a common dimension in this case back to the original image dimensions (480×640 pixels). By keeping the masks in integer format and using nearest-neighbor sampling, we preserve each class label exactly, avoiding any blending or interpolation artifacts.

Then, an instance extraction and feature computation metric is performed. For each resized mask, the script isolates the pixels belonging to one class at a time, labels connected components, and computes the geometric descriptors: **Area** (Pixel count within each labeled region.); **Equivalent Diameter** (Diameter of a circle with the same area.); **Circularity** (A normalized shape metric: $4\pi \times area/perimeter^2$, indicating shape compactness.); and **Bounding Box** and **Centroid** (Used for spatial analysis and tracking.) These descriptors enable both individual bubble analysis and distributional statistics over each image or dataset. Each region's metrics are then assembled in a dataframe capturing these geometric describing values for every instance (bubble occurrence) of every image in the test dataset.

The pipeline ends with a visualization and analysis stage where complementary plots based on the metrics calculated previously are plotted so that a deeper discussion can be performed. The most relevant graphs included in this part are boxplots, cumulative distributions and density estimates. **Boxplots** were used to display the spread and outliers of each shape feature, helping to assess if the model is consistently underestimating or overestimating bubble size in edge cases. They were used to compare each metric's central tendency and outlier spread, highlighting shifts

in median values and extreme observations. **Empirical Cumulative Distribution Function (ECDF)** plots provide a non-parametric view of the full distribution, with each curve representing either a model or the ground truth obtained from the original masks metrics. By plotting the cumulative distribution functions, questions like “what proportion of bubbles are smaller than x px^2?" can be answered with precise percentiles. When a model’s ECDF lies to the left of the GT curve, it underestimates that metric; when it lies to the right, it overestimates. Moreover, a shallower ECDF indicates greater variability in the model’s predictions compared to the GT distribution. Finally, **Kernel-Density Estimates (KDEs)** are smoothed density curves for each metric/class that allow the comparison of the full shape of distributions are they skewed?", not just quartiles. They plot highlights modes and sub-peaks in the data that boxplots cannot reveal, transforming raw measurements into a smooth, continuous “density hill” along the x-axis (e.g., equivalent diameter). The summit of the curve indicates the most frequent value for a given metric, while the width reflects its variability. As with the other visualizations, the closer the model’s KDE curve aligns with the ground truth curve, the closer the metric to the real value and the better the segmentation model’s performance.

Chapter 5

Results and Discussion

This chapter starts with the presentation and discussion of training and its validation losses, testing and inference results of each model. Then a broader discussion on the models' global performance, Class-level trends, and effects of SR enhancing is done, followed by an assessment of some model performance on additional unseen images. Finally, the bubble characterization analysis results are presented.

5.1 YOLO

5.1.1 Training metrics results

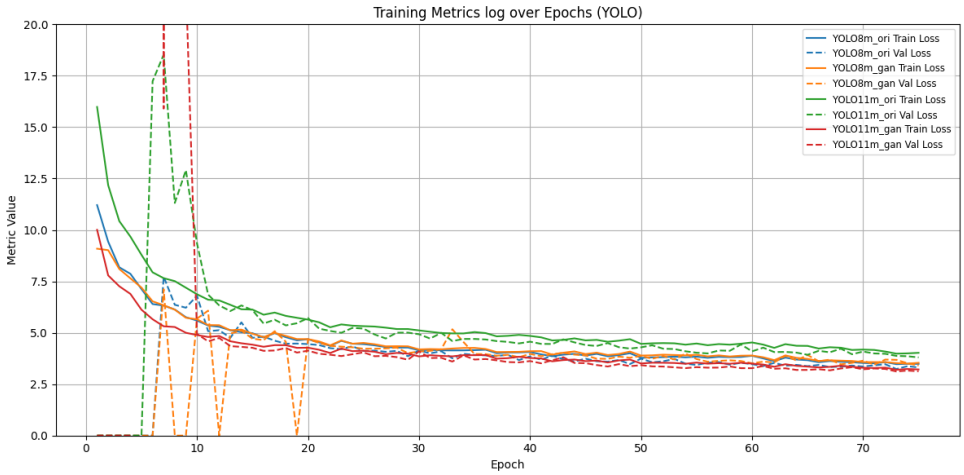

Figure 5.1: Training loss across epochs for YOLO models, comparing original (ori) and GAN-augmented (gan) datasets. Both training and validation losses are shown for YOLOv8m and YOLOv11m configurations, illustrating convergence behavior and differences in loss trajectories.

Figure 5.1 presents the Training and Validation loss log, obtained during the training process for the four YOLO variants. Regarding the Training Loss Convergence, all four models show a rapid initial drop in losses over the first 10 to 15 epochs, then a slower but consistent decline for the remainder of the training process with a constant slope. The Validation loss starts with highly dispersed values, but after epoch 15 it converges closely following the Training loss, indicating no signs of overfitting. YOLOv11m-seg, trained with original images, has the highest loss across all categories. This contrasts with the values obtained with YOLOv11m-seg, trained with SR-enhanced images, as this model achieves the lowest loss scores.

The early-epoch jump confirms that the detector rapidly learns to find most bubble instances, but the slow tail suggests remaining false negatives, likely small or irregularly shaped bubbles that the model still misses. The close clustering of all four curves indicates that, despite architecture or data differences, the models converge to a similar ability to catch several masks per image across all classes.

5.1.2 Test metrics results

Table 5.1: Global metrics for the models YOLOv8m-seg and YOLO11m-seg

Dice	Accuracy	Precision	Recall	F1 Score	MSE	AP50	AP50-95	Mean IoU	Model Name	Data Type
0.706	0.940	0.725	0.790	0.746	0.202	0.359	0.233	0.595	yolov8m-seg	original
0.708	0.942	0.744	0.792	0.740	0.205	0.401	0.272	0.597	yolov8m-seg	SR enhanced
0.739	**0.949**	**0.770**	**0.794**	**0.775**	**0.187**	**0.464**	**0.311**	**0.627**	yolo11m-seg	original
0.666	0.936	0.704	0.708	0.691	0.230	0.284	0.203	0.549	yolo11m-seg	SR enhanced

Global Metrics: Table 5.1 reports the overall segmentation performance of the two YOLO models. YOLO11m-seg trained on the original data had the best overall performance with maximum values for the evaluated metrics (Dice = 0.739, Accuracy = 0.949, AP50 = 0.464, Mean IoU = 0.627), and a lower error (MSE = 0.187). This suggests that this model was the most balanced and efficient in the segmentation performance. YOLOv8m-seg trained with enhanced images had a competitive score, especially in precision and recall with the values (0.744 and 0.792) respectively. Super Resolution had positive impact on YOLOv8 global metrics but a negative impact on YOLOv11 global metrics. Also worth noting that YOLOv11 model trained on SR dataset had worse results than the worse YOLOv8 model meaning that possible artifacts introduced in the SR augmented images may have affected a model that was slightly more sensitive in fine detail edge segmentation.

Figure 5.2: Schematic representation of the class-wise segmentation performance table for the YOLO model versions shown in the appendix A.5 table A.1

Class Metrics: As shown in Figure 5.2, regarding the model YOLOv8m-seg, the version trained on augmented data had overall better performance in the Bubble class, with a clear increase in IoU (0.635 vs 0.606) and mAP50-95 (0.505 vs 0.370), suggesting the augmentation improved generalization and confidence. In the cut class, there were slight improvements in IoU and precision but recall dropped, meaning it reduced false positives but made the model less sensitive. The Aggregate class training using SR enhanced images improves detection confidence and recall but reduces precision of the masks. For the Spherical class, SR enhanced dataset training led to a slight decline in segmentation accuracy across most metrics (IoU, AP50, precision), despite a small gain in recall (0.798 vs 0.812).

For the YOLO11m-seg model, the use of SR enhanced data had a overall negative impact on segmentation performance across classes. The Bubble class metrics experienced a decrease in all major metrics when trained with enhanced image data (IoU dropped from 0.618 to 0.577, AP50

from 0.519 to 0.434, and recall from 0.821 to 0.731), indicating a reduced ability to generalize to real instances. The Cut class suffered a noticeable decline in performance as well, with IoU falling from 0.416 to 0.294, AP50 from 0.350 to 0.221, and recall from 0.589 to 0.413. While precision remained relatively high (0.752). This may suggest that the model became significantly more conservative, missing many Cut features. For the Aggregate class, IoU and AP50 decreased slightly, but recall remained high (0.880 vs. 0.865), showing that SR enhanced training preserved detection capability, though with reduced segmentation precision. The Spherical class was most negatively impacted, with IoU decreasing from 0.441 to 0.321, and mAP50-95 from 0.241 to 0.104, indicating severe degradation in segmentation quality. In summary, SR enhanced data did not improve YOLO11m-seg performance and even degraded results across most classes, particularly in complex or finely shaped structures like Cut and Spherical bubbles.

Across all YOLO-based models evaluated, the classes Cut and Spherical consistently posed the greatest challenges for semantic segmentation with the Cut class showing the lowest IoU scores in nearly all cases and missed detections. While recall was sometimes high, particularly with YOLOv8m trained on SR enhanced images, precision and mAP50-95 remained low-indicating frequent false positives or poor spatial alignment of masks. Overall, SR enhancing improved generalization for some YOLOv8 classes (notably Bubbles) but introduced trade-offs in others, with increased confidence often coming at the cost of mask quality or recall. These consistent difficulties likely arise from the subtle, inconsistent visual differences and smaller size of Spherical and Cut features compared to more distinct classes like Bubble or Aggregate, which have a bigger representation and therefore more training data, and due to all the occurrences having larger areas represented. The results highlight the need for either more representative training data or class-specific loss weighting to improve segmentation fidelity in these under-performing categories.

Inference: Analyzing the Figure 5.3 we can see three images plotted side by side. This is going to be the consistent structure used to present the inference results:

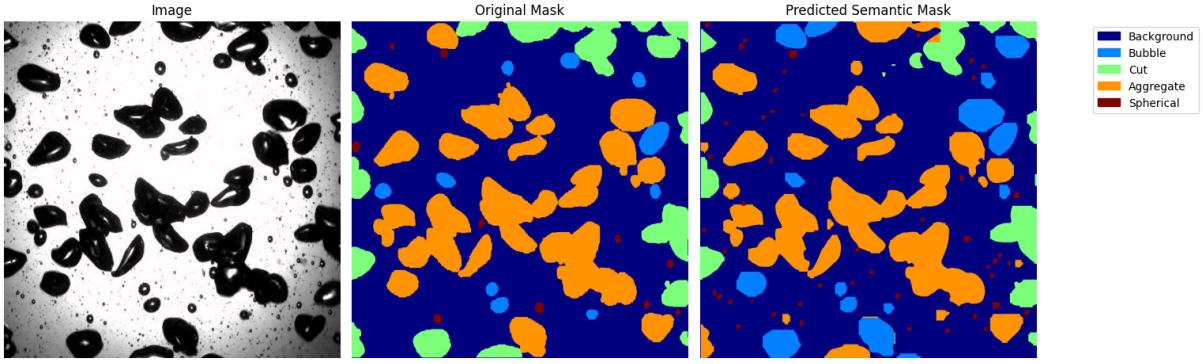

Figure 5.3: From left to right the input image, original mask and the predicted mask obtained with a YOLO model are displayed

Figure 5.3 shows a scenario with bubbles closely packed. The segmentation is overall successful, with most bubbles being detected, however some segmentation problems can be identified, such as imperfections regarding bubble edge accuracy and misclassifications mainly of the cut class. In less dense bubbly flow images, these issues are less noticeable, reflecting a more accurate mask. There are occasionally cut bubble class misclassifications, but less prominent. Despite not being a common occurrence it is important to notice that a classification error in the human-made 'original' mask was identified (bubble on the bottom right corner). An example of this less dense flow can be seen in the appendix figure A.4. There is also a over classification and segmentation of bubbles of the class spherical meaning that can justify some of the poor results in this class as the model is considering bubbles of a smaller scale to be part of this class this could be rectified post segmentation using a threshold area value for example. These findings are consistent in the rest of the dataset. Another observed limitation to the implementation of this model was the bounding box. Since its size cannot be tailored (increased), the segmentation edges have remnants of the box sides. This hinders the validation quality and therefore reduces the effects of using an original or SR enhanced dataset to train the model. Increasing the size from 512 px to 1024 px of the image, so that the resolution of the matrix operations is better, slightly improved the outlines but did not eliminate the box outline presence. This effect is not always visible, but under careful observation, it can be noticed as highlighted in A.5.

5.2 Detectron 2

5.2.1 Training metrics results

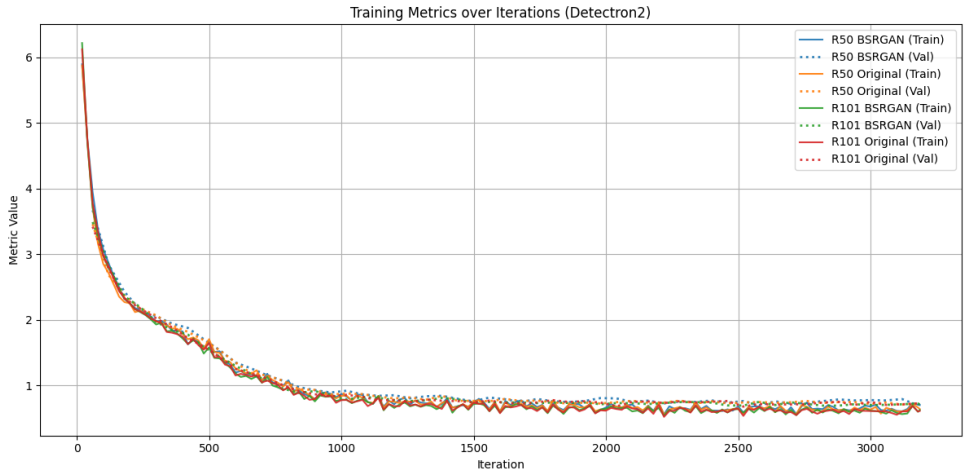

Figure 5.4: Training and validation loss across iterations for Detectron2 models, comparing original and BSRGAN-augmented datasets. Results are shown for R50 and R101 backbone configurations.

All four models exhibit a steep decline in total loss within the first few hundred iterations, indicating rapid initial learning - Figure 5.4. The R-101 FPN 3× Original configuration reaches the lowest final loss (≈ 0.54) followed by the R-50 FPN 3× Original (≈ 0.55) and the two SR enhanced variants (≈ 0.58). Despite these small differences, every model converges to a similar plateau in loss in the training dataset.

The SR enhanced runs start with higher initial loss and lower initial accuracy, likely reflecting the added complexity of synthetic training examples, but ultimately reach comparable final performance. This suggests that while SR enhanced data may slow early convergence, it does not impair the model's ability to learn reliable segmentation boundaries over a full training schedule.

As for the Backbone Size, the larger R-101 backbone shows slightly better final loss and accuracy than its R-50 counterpart when trained on original data.Concerning the convergence behavior, none of the loss curves display sudden spikes or oscillations, implying stable training dynamics and an appropriate choice of learning rate. The losses and accuracies plateau after roughly 1000 to 1500 iterations, suggesting that further training yields diminishing returns.

All metrics obtained in the training log are derived from the training set exclusively. The

model shows high final mask accuracy and low loss risk of overfitting, particularly for the larger R-101 model, but to ensure generalization, these training metrics are compared to the ones obtained in the custom test pipeline.

5.2.2 Test metrics results

Detectron2 test metrics obtained can be seen in the following tables.

Table 5.2: Global metrics for Detectron2 Mask R-CNN models (ResNet-50 and ResNet-101 with FPN) trained with and without SR enhanced data.

Dice	Accuracy	Precision	Recall	F1 Score	MSE	AP50	AP50-95	Mean IoU	Backbone	Data Type
0.829	0.961	0.850	0.821	0.833	0.156	0.628	0.498	0.727	R_50_FPN_3x	original
0.827	0.961	0.843	0.826	0.833	0.159	0.631	0.499	0.727	R_50_FPN_3x	SR enhanced
0.823	0.961	0.849	0.818	0.832	0.161	0.611	0.481	0.720	R_101_FPN_3x	original
0.821	0.959	0.840	0.811	0.825	0.166	0.630	0.498	0.715	R_101_FPN_3x	SR enhanced

Global Metrics: In Table 5.2 the global metrics for the Detectron model are shown. All models performed strongly with differences between model variations are minimal, with Dice coefficients above 0.820, accuracies near 0.960, and F1 scores around 0.830. Mean IoU values were consistent across all models, ranging from 0.715 to 0.727.

The SR had mixed effects with metrics remaining practically unchanged or slightly improved but again the variations are minimal between model versions.

There is no single best-performing model among the Detectron2 architectures; however, unexpectedly, the model Mask R-CNN R_50_FPN_3x consistently obtains better metrics than the model with the backbone Mask R-CNN R_101_FPN_3. Notably, the model with ResNet-50 backbone demonstrates surprisingly strong and consistent performance when compared to the deeper counterpart. Despite ResNet-101 generally offering more representational capacity due to its increased depth, it does not appear to translate into superior segmentation performance in this context.

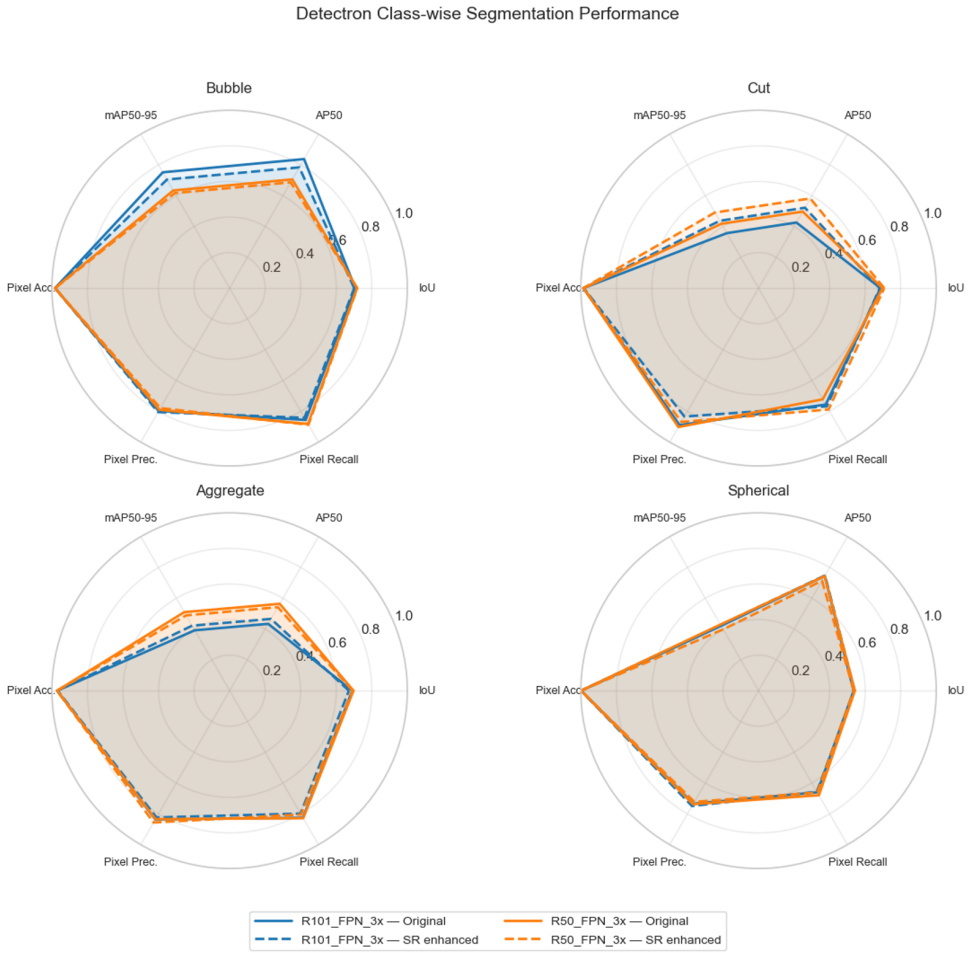

Figure 5.5: Schematic representation of the class-wise segmentation performance table for the Detectron2 model versions shown in the appendix A.5 Table A.2

Class Metrics: From the results presented in Figure 5.5 and Table A.2, the "Bubble", "Cut", and "Aggregate" classes display moderate segmentation performance, with IoU values ranging between 0.670 and 0.720, and AP50 and mAP50-95 scores showing some variability, especially for the "Cut" and "Aggregate" classes. These results suggest that while object boundaries are generally well localized, inter-class similarity and structural complexity, particularly in SR-enhanced data, may contribute to minor detection ambiguities. The "Spherical" class presents the most challenging detection task, reflected in its consistently lower Recall and IoU scores, despite the high pixel-level accuracy. This discrepancy points to an over-representation of background pixels or small object sizes that can inflate pixel accuracy but penalize region-based metrics like IoU and mAP.

SR enhance enhancement showed mixed results in the "Bubble", "Aggregate" and "Spherical" classes shifting the detection balance of precision and recall depending on the backbone. Under

R50 backbone, the AP50 metric decreased and Recall sightly improving and the opposite can be seen in the results from the backbone R101. The "Cut" class benefited the most from SR enhancement with consistent gains across most metrics in both backbones.

The counterintuitive result presented in the global metrics is further represented in the class metrics, suggesting that increased model depth does not necessarily translate to better performance for all classes. The R101 backbone demonstrates a slight but consistent improvement over the R50 in "Bubble" and "Spherical" class segmentation, especially in terms of mAP, highlighting the importance of deeper feature hierarchies in better capturing the most uniform classes. However, in the "Cut" and "Aggregate" classes, both IoU and mAP50-95 metrics are generally lower compared to the shallower R50_FPN_3x, regardless of the input data type. A possible explanation may be that R101_FPN_3x may lead to an over-segmentation in morphologically ambiguous regions such as cuts or aggregates, which often lack consistent textural or geometric cues.

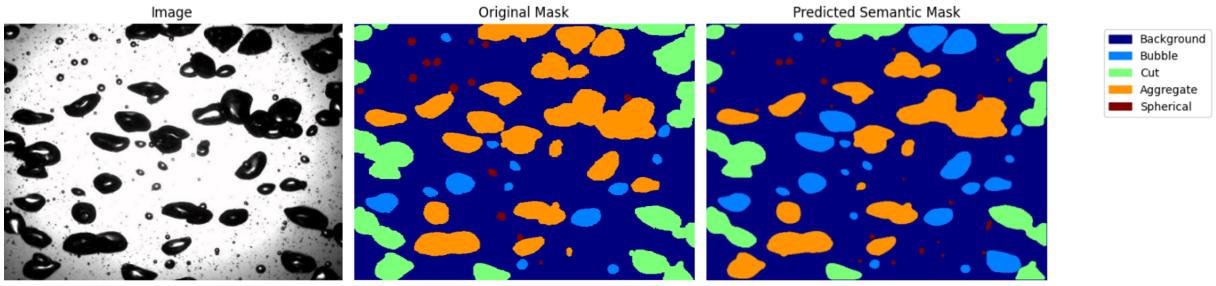

Figure 5.6: From left to right the input image, original mask and the predicted mask obtained with a Detectron2 model are displayed

Inference: The Detectron2 predicted masks show very smooth contours, which is advantageous for identifying simple, near-spherical bubbles. However, for the “Aggregate” and “Cut” classes, where instances arise from the fusion of multiple bubbles, this smoothing erases important edge features. As a result, adjacent bubbles that should remain distinct occasionally merge in the prediction, leading to misclassification. For example, in Figure 5.6, many aggregate regions are incorrectly labeled as individual bubbles due to these lost boundary details.

Moreover, the “Spherical” class predictions are inconsistent: the model sometimes identifies bubbles that lack annotations and misses others that are obvious. This likely reflects not only the variability in the manual ground truth annotations but also the fact that spherical bubbles are the least-represented class in the training data.

5.3 TransUNet

5.3.1 Training metrics results

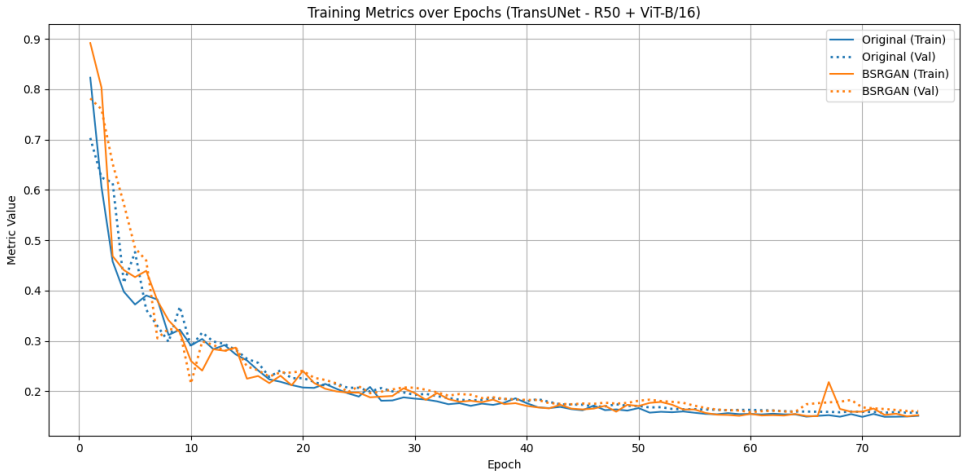

Figure 5.7: Training and validation loss across epochs for the TransUNet model (R50 + ViT-B/16), comparing original and BSRGAN-augmented datasets.

Figure 5.7 compares training and validation loss for two models trained with the TransUNet architecture (ResNet50 + ViT-B/16 backbone): one on the original dataset and the other using SR enhanced data. Both models show relatively smooth convergence in both training and validation loss, with validation accuracy improving steadily. The initial dispersion in the validation results is resolved in later epochs, showing no signs of overfitting. SR trained model shows slightly more variation in training loss, which might imply that, while SR enhancements enrich the training diversity, they could introduce noise or distributional shifts that hinder generalization, at least under the current training configuration.

5.3.2 Test metrics results

Table 5.3: Comparison of TransUNet Performance on SR enhanced vs. Original Data

Dice	Accuracy	Precision	Recall	F1 Score	MSE	AP50	AP50-95	Mean IoU	Model Name	Data Type
0.837	0.968	0.869	0.843	0.855	0.138	0.799	0.662	0.749	TransUNet_R50+ViT-B_16	Original
0.826	0.964	0.846	0.836	0.840	0.152	0.803	0.661	0.730	TransUNet_R50+ViT-B_16	SR enhanced

Global Metrics: From the results described in Table 5.3, the model trained on the original dataset slightly outperforms the SR enhanced version in most metrics. The Dice score improves

from 0.826 to 0.837, and Mean IoU increases from 0.730 to 0.749. Recall also better in the model trained on the original dataset (0.843 vs 0.836), the MSE decreases from 0.152 to 0.138 and the overall F1 score rises from 0.840 to 0.855, suggesting a more balanced trade-off between precision and recall with the original dataset. Interestingly, AP50 is marginally higher in the SR enhanced case (0.803 vs. 0.799), but this does not translate across the whole IoU threshold range as mAP50-95 is lower, meaning the model finds objects well but struggles to localize them accurately.

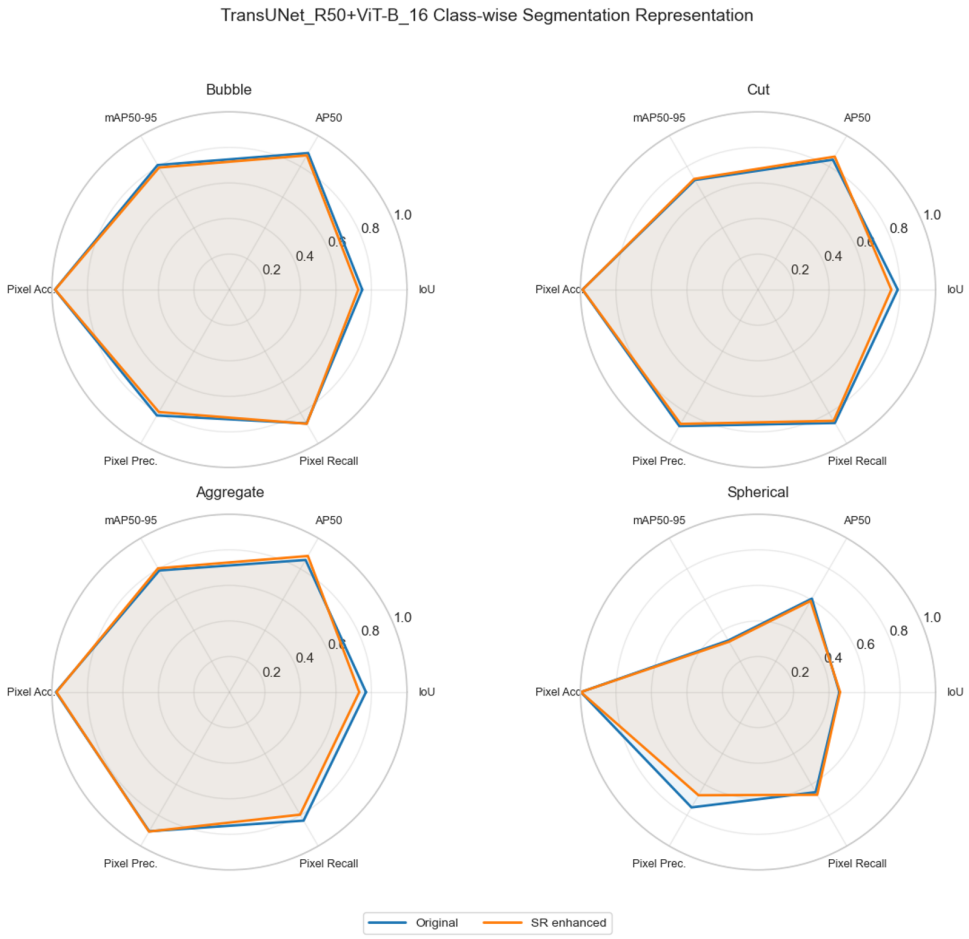

Figure 5.8: Schematic representation of the class-wise segmentation performance table for the TransUnet model versions shown in the appendix A.5 table A.3

Class Metrics: As shown in Figure 5.8 and Table A.3, the Bubble class sees consistent gains in all metrics when trained on the original data. Cut class metrics such as IoU rises from 0.751 to 0.787, Pixel Precision from 0.872 to 0.886, and Pixel Recall from 0.852 to 0.866 for the model trained on the original dataset. However, mAP50-95 is slightly higher with SR-enhanced trained model (0.719 vs. 0.712). This may mean that SR enhancing slightly helps with detection at

lower IoU thresholds but leads to less accurate masks. For the Aggregate class, results are very close between both datasets with the original data trained model having slightly better results. The Spherical class shows the lowest segmentation performance overall. Both datasets yield IoU below 0.47, mAP50-95 below 0.34 and precision increases considerably with the Original data.

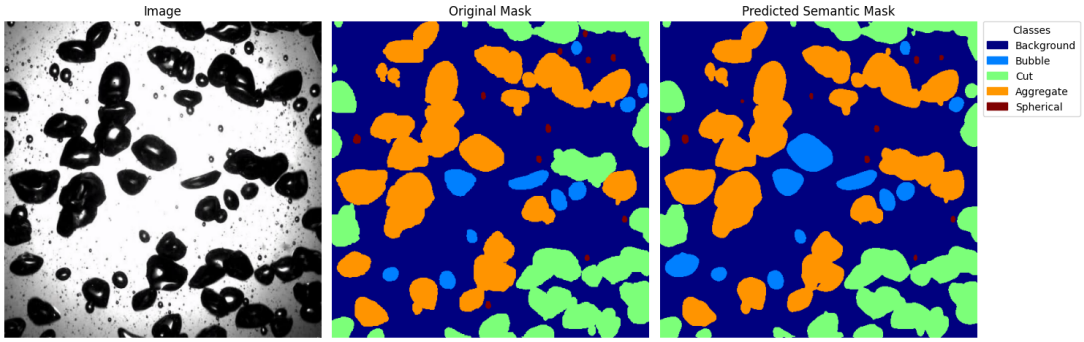

Figure 5.9: From left to right the input image, original mask and the predicted mask obtained with a TransUNet ViT-B model are displayed

Inference: Observing Figure 5.9, the model's predicted outlines are exceptionally precise, capturing even the most subtle bubble boundaries. Although it occasionally misses a few spherical bubbles, these instances are rare. An error in the annotated mask can be seen (center right green bubble annotated as cut but that should be aggregate). The model generally produces the expected mask, and in cases where there are discrepancies, the true class is often ambiguous, even for human evaluation. Overall the model's performance is highly satisfactory and extends to all images of the test dataset, with another example of generated masks in A.7.

5.4 SAM-LST

5.4.1 Training metrics results

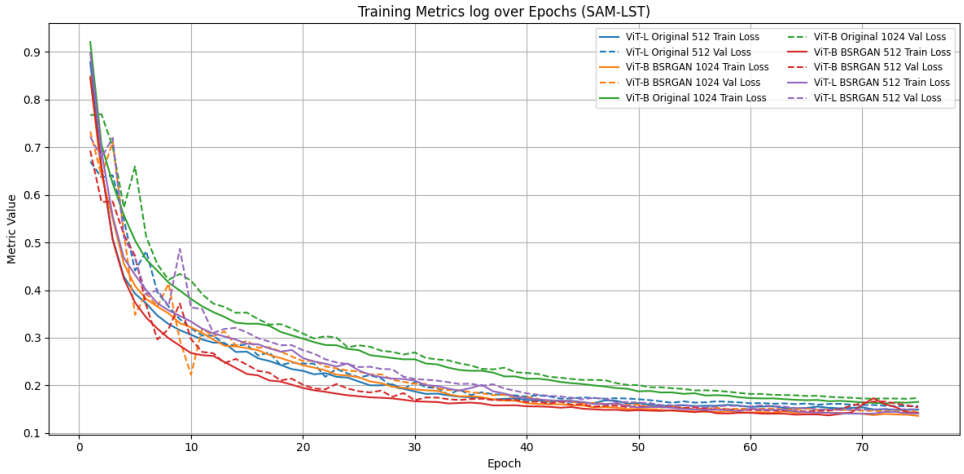

Figure 5.10: Training and validation loss across epochs for SAM-LST models, comparing original and BSRGAN-augmented datasets. Results are shown for ViT-B configurations with 512 and 1024 input sizes and ViT-L configurations with input size 512.

In Figure 5.10, among the tested configurations, regarding the backbone ViT-B, the model trained with SR enhanced data and input image size 512 pixels achieved the lowest overall loss, closely followed by the model with input image size 1024 px, suggesting that the use of SR enhanced data consistently benefits training. The model trained with the original data and an image size of 1024 pixels exhibited the slowest convergence and the highest loss across most epochs. The ViT-L-based models (vitl_original_512 px, vitl_gan_512 px) demonstrated intermediate performance, with the model variant trained on SR enhanced again outperforming its original-data counterpart. It appears that augmented images have a positive impact in improving training efficiency and convergence.

5.4.2 Test metrics results

Table 5.4: Global metrics for the models SAM-LST

Dice	Accuracy	Precision	Recall	F1 Score	MSE	AP50	AP50-95	Mean IoU	Model Name	Data Type	Image Size
0.806	0.957	0.826	0.815	0.817	0.169	0.710	0.577	0.702	sam_lst_vit-b	original	1024 px
0.777	0.957	0.817	0.819	0.817	0.176	0.663	0.536	0.684	sam_lst_vit-b	SR enhanced	1024 px
0.791	0.954	0.815	0.781	0.792	0.199	0.719	0.537	0.675	sam_lst_vit-b	original	512 px
0.800	0.955	0.816	0.803	0.807	0.177	0.726	0.567	0.688	sam_lst_vit-b	SR enhanced	512 px
0.795	0.952	0.820	0.805	0.805	0.173	0.704	0.571	0.684	sam_lst_vit-l	original	1024 px
0.776	0.956	0.817	0.804	0.810	0.166	0.678	0.549	0.670	sam_lst_vit-l	SR enhanced	1024 px

Global Metrics: Table 5.4 shows the global SAM-LST metrics.

Starting with an analysis of how augmented images affect test metrics depending on image input size, for ViT-B models trained with 1024 pixels, the original data had a higher performance across almost all metrics, especially in AP50 (0.710 vs 0.663), mAP50-95 (0.577 vs 0.536), and Mean IoU (0.702 vs 0.684). For the ViT-B models trained with 512 pixels input images, the usage of augmented images slightly improved Dice (0.800 vs 0.791) and Mean IoU (0.688 vs 0.675), and AP50 (0.726 vs 0.719), as well as AP50-95 (0.567 vs 0.537). This indicates that when the input image has a lower resolution, the use of enhanced images for training had a positive effect, possibly compensating for the loss in detail. For higher resolution inputs, there were no benefits from using the SR enhancement.

The comparison between the SAM-LST segmentation model backbones, ViT-B and ViT-L, was conducted using the same input image size of 1024 pixels. Both architectures demonstrate strong performance; however, ViT-B models trained on the original data achieved the best overall performance, leading in key metrics such as mAP50-95 (0.577), Mean IoU (0.702), indicating superior localization and overlap accuracy and highest Dice coefficient (0.806), reflecting balanced segmentation across classes. It was expected that the deeper SAM model would have better performance due to its deeper connections and more parameters but that is not what is observed. A possible explanation for this is the initialization of the weights. Due to the lack of a pretrained ViT-L transformer, the missing initial weights of the model SAM-LST ViT-L were initialized randomly. Since a large dataset is needed to fine-tune all these new parameters, it is possible that the training dataset available was not enough to achieve beneficial weights in these tasks due to its specificity.

Figure 5.11: Schematic representation of the class-wise segmentation performance table for the SAM-LST model versions shown in the appendix A.5 table A.4

Class Metrics: In Figure 5.11 and Table A.4, when assessing the effect of Image Size on Class Performance in the ViT-B model, starting with the model trained on the original images, the bigger image size (1024 px) in the classes Bubble and Spherical led to higher precision and improved mAP50-95. The other metrics and classes had mixed results appart from a dramatic improvement in pixel recall for the spherical class.

Now analyzing how using SR enhanced images affects class performance, there was a slight advantage in Recall and Dice but a slight drawback in the other metrics, notably in mAP50 and mAP50-95 suggesting a better ability to capture these objects' presence but indicating a trade-off between detecting more instances and possibly introducing more false positives. The classes most affected by this were the Bubble and Cut classes, with the Aggregate Class remaining relatively stable. As for the Spherical class, the introduction of GAN revealed to worsen the results in the models trained with the bigger image size possibly due to the introduction of artifacts on already

well-defined structures.

Among all classes, the Spherical class is the worst performing class across the board, consistently showing the lowest IoU (as low as 0.339), AP50, and mAP50-95 regardless of model, image size, or augmentation. This suggests that the Spherical class, despite its simple shape, is being under-classified and segmented. The Cut class occupies an intermediate position, with variability based on model and configuration, but overall better consistency than Spherical. The Bubble and Aggregate classes have the best average precision values consistently.

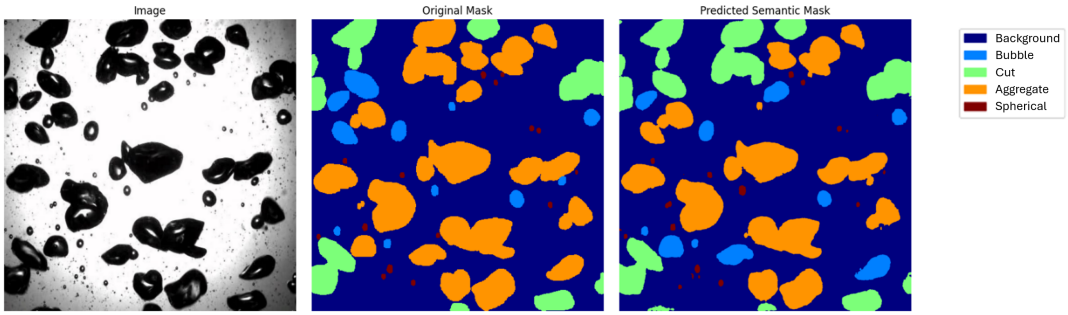

Figure 5.12: From left to right the input image, original mask and the predicted mask obtained with a SAM ViT-B model are displayed

Inference: Regarding the inference shown in Figure 5.12, the generated masks closely match the original hand-segmented mask, exhibiting high quality and sharp edge definition. There are a few cases in which the model erroneously groups independent bubbles into aggregates or cuts (top left corner), but these remain exceptions. When an instance is ambiguous-particularly near the image borders the model still tends to assign a reasonable class. Finally, in this example, the model successfully detects most spherical instances, although it struggles to delineate their boundaries with the same level of precision.

5.5 Discussion

5.5.1 Global performance

The global values are summed up in the table A.5. TransUNet achieved the highest global testing metrics scores, especially when trained on the original dataset (Dice: 0.837, IoU: 0.749, AP50: 0.799). It exhibited consistent segmentation quality across all classes, with a strong

balance between precision and recall. SAM-LST (ViT-B 1024 px original) closely followed, offering similarly high Dice and AP50 values (Dice: 0.806, mAP50-95: 0.577, IoU: 0.702), demonstrating that transformer-based architectures can perform competitively, especially when pretrained weights are leveraged. Detectron2 also presented fairly competitive results with AP50 and IoU values around (0.630 and 0.720 respectively). YOLOv11m-seg trained on original data outperformed other YOLO models, but still fell short of TransUNet, SAM, and Detectron2 in metrics like mAP50-95 (0.310). Overall, these results suggest that architectures integrating transformer mechanisms with strong encoder-decoder designs, combined with large-scale pretraining and enhanced through task-specific fine-tuning, a promising direction for accurate and reliable segmentation in this context.

An important limitation to consider is the quality of the masks. The annotation process is extremely time-consuming and subject to substantial variability in each annotator's style. Moreover, even a single annotator's style can fluctuate depending on multiple factors, such as fatigue or concentration. This introduces an upper bound on the achievable precision of the results: some models may, in fact, be closer to the true ground-truth than the metrics suggest.

As a consequence, many detections labeled as False Positives may actually correspond to True Positives that were simply not annotated. This is particularly relevant in classes where annotation inconsistency is more pronounced due to object ambiguity, leading to poorer metric scores that do not necessarily reflect inferior model performance. A possible way to mitigate this issue is to apply post-processing techniques to reduce problems such as over-classification, as observed in the Spherical class.

The inference times were also calculated for each model during the inference step of the validation pipeline, the following values were obtained:

Table 5.5: Timing summary for the different segmentation models.

Model	Images Processed	Total runtime [s]	Avg total / image [s]
YOLO	43	24.5190	0.5701
Detectron2	43	34.3527	0.7987
TransUNet	43	44.8130	1.0421
SAM-LST	43	74.6620	1.7362

The YOLO is the model with the fastest runtime closely followed by Detectron2 and then TransUnet with SAM being slowest model. In terms of the actual inference pass for each model, a standard way to calculate it was not achieved, however the results were approximately 0.1 seconds per image. This contrasts with the average processing time per image as we can see

YOLO and Detectron2 architectures are better integrated for loading and processing images than TransUnet and SAM-LST. This means 'fluid' real time segmentation at 50 fps would not be possible with the current configurations, as the models tested are only capable of making 10 frames a second (disregarding image loading times). This, however does not take away from the already very capable and possible implementation of these models in real time scenarios where an idea of the current status of the flow can be obtained albeit with a delay.

5.5.2 Class-Level trends

Some consistent patterns can be found in the class-level testing metrics. **Bubble** class generally had high precision and recall, with top IoU values (0.74-0.68) in TransUNet and SAM-LST. It was also the best-represented class in the dataset with the largest number of instances. It benefited the most from SR enhancement in some models, particularly in YOLOv8 and SAM_LST_ViT-B with image size 512 pixel. **Aggregate** class performed moderately well, with IoU values typically ranging 0.63-0.77 across all architectures. It showed resilience to SR enhancing, generalizing well regardless of the dataset used to train the model. In cases where bubbles were extremely close to one another, the model occasionally revealed some problems in classification being the most noticeable error detected through visual analysis.

Cut and Spherical classes were the most challenging with **Cut** frequently showing lowest IoU and recall scores. These metrics suggest that because Cut regions are ambiguous with irregular visual boundaries they are not easily learned by the models, paired with an under representation in the dataset and not improved with SR enhanced techniques. The visual analysis backs this result with several misclassifications partially due to edge contact ambiguity. As for the **Spherical** class, despite seemingly simple geometry, it had low AP50 and IoU across all models, likely due to its small size and inconsistent annotations. Best-performing configurations (e.g., SAM ViT-L original) only reached mAP50-95 around 0.32-0.43.

5.5.3 Effects of SR enhanced Data

SR-enhanced datasets had inconsistent effects on global performance across models. In the YOLO models, SR enhancement increased recall, notably for YOLOV11m-seg, but led to a drop in AP50, AP50-95, and IoU, suggesting a trade-off between detection sensitivity and localization accuracy. For SAM-LST, the enhancement helped the deeper ViT-L model, slightly

improving metrics like Dice and AP scores, while in ViT-B, the results were more erratic. The effect was heavily dependent on the input resolution. At 512 px, SR enhancing helped recover some lost detail; at 1024 px, original data yielded better performance in most classes except Cut and Bubble. Detectron2 models saw consistently slightly positive improvements in most metrics with SR-enhanced data, particularly in recall and Dice, suggesting it improved models ability to maintain consistent and reliable performance despite variations in the input data. In contrast, TransUNet did not benefit from SR enhancement, with original data yielding better performance across most global metrics, especially Dice, precision, and mean IoU but showing slight improvements in the broader average precision metric AP50.

In summary, SR proved most effective when used with deep architectures (e.g., SAM ViT-B, Detectron2 R101) and at lower input resolutions. Despite this, it did not provide consistent gains across all classes and metrics, sometimes negatively impacting model performance. The limited dataset size and computational power also possibly restricted these techniques' effectiveness since the fine details introduced by this SR enhancement are not picked up by most models. Moreover, SR-enhanced data may lack sufficiently realistic characteristics or introduce unintended biases that negatively affect the generalization capabilities of segmentation models.

Inference Quality and Model Limitations

Through all the inference images as well as side by side comparisons such as the figure A.9 the quantitative findings were corroborated: SAM and TransUNet showed cleaner, better-aligned mask edges, while Detectron2 had a smoother appearance, along with YOLO models suffering from more misclassification instances and worse edge precision.

These observations highlight the importance of architectural design in final mask quality: end-to-end mask generation (as in SAM or TransUNet) provides more precise outlines, outperforming models where masks are derived post-region proposal.

5.6 Maze Entrapment Experiment

The following work explores the application of the top-performing models to images obtained from a Bubble Entrapment Experiment, thereby assessing their performance across a novel domain. An objective of this experiment was to visualize and characterize the bubbles in the input chamber (Zone 1) and compare them to the output chamber (Zone 2). Figure 5.13 (right)

illustrates the pipeline's progression:

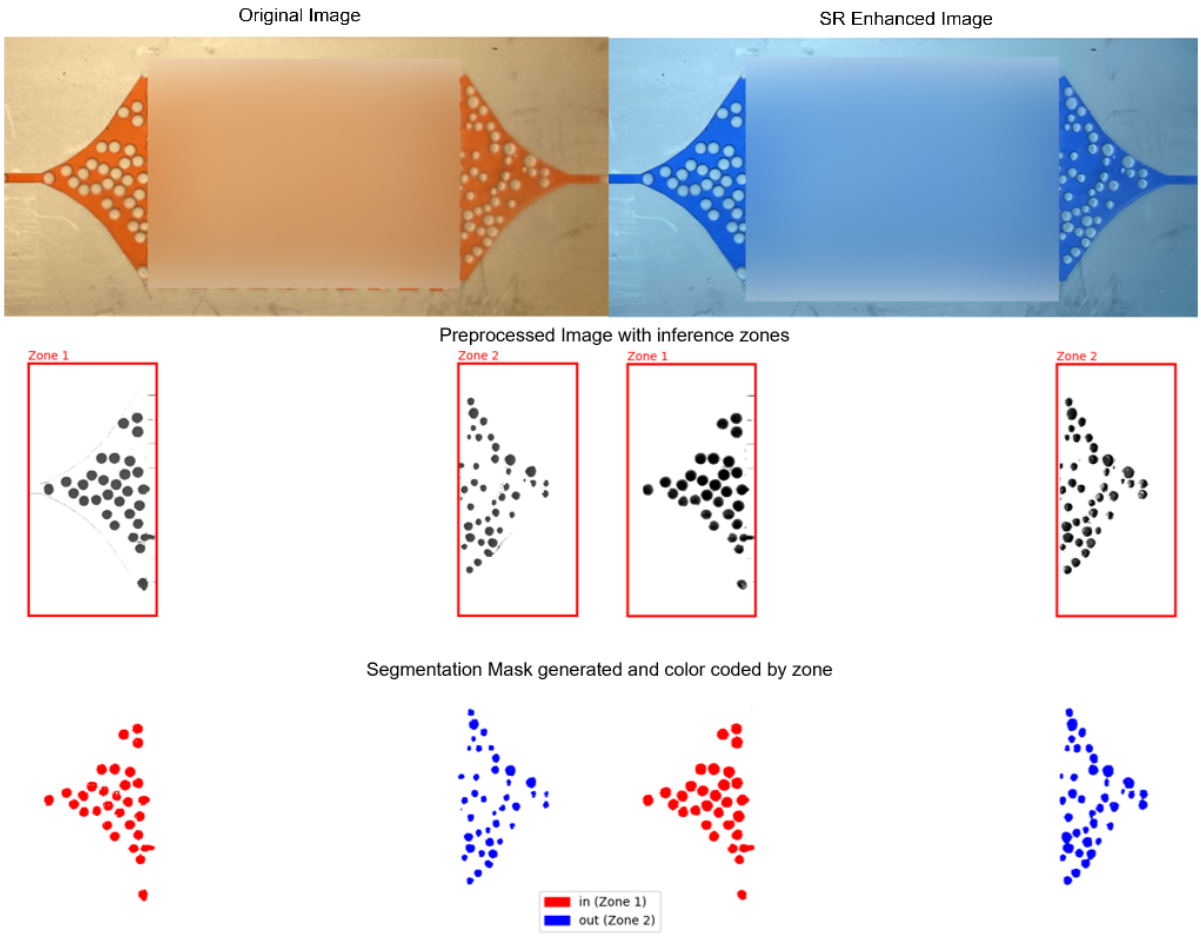

Figure 5.13: Sam Inference maze experiment with Original and SR enhanced images

To adapt the models to these new images a preprocessing step was required. From the original images, the inference zones were defined and isolated in a segmentation mask. Then, the image was converted to grayscale in order to better distinguish bubble edges and optimize the models effect since they were trained on grayscale images. The last step was to apply the preciously created pipelines to perform inference and bubble characterization, with the final output mask and object characteristics being obtained.

In the Original image, poor image quality and shadow edges created due to inadequate lighting can be observed. The shadow artifacts obscured and partially compromised true bubble boundaries. In the SR enhanced preprocessed image, this was mitigated with a dramatic improvement in bubble definition. The left-hand side (SR-enhanced region) shows much sharper, more distinct bubble boundaries compared to the non-enhanced areas. Moreover, the pre-process

step was considerably less complex and resulted in far better results due to the finer edges and more uniform background. With the preprocessed image a SAM-LST trained on the dataset discussed in the section 4.1 was applied and a segmentation mask was obtained.

The SR-enhanced masks show improvement across most F1 jumped from 0.807 to 0.851while pixel-level metrics-accuracy remained constant and MSE decreased (from 0.018 to 0.014), indicating tighter pixel-wise agreement. Per-class IoU and Dice both increased, meaning the augmented data led to more accurate region overlaps. At the instance level, AP@0.5 remained high for both classes, while mAP@0.50-0.95 notably improved in zone 2 highly affected by shadows (0.283 to 0.425), suggesting the model generalizes better across stricter IoU thresholds after augmentation.

This proof of concept can be considered successful, and with some slight adjustments, this working method can be used to efficiently categorize the images of the experiments' results.

5.7 Bubble Characterization

Only the best performing models, SAM-LST and TransUNet were utilized for the bubble characterization. These models encompass SAM-LST ViT-B and ViT-L versions and TransUNet ViT-B version trained in an original and augmented dataset. The metrics were calculated from the inferences obtained from the test dataset original images for each model configuration.

5.7.1 SAM ViT-B

From the metrics evaluated in this section circularity had the most noticeable variation and therefore its plots will be presented as an example, with the other metrics being shown in the Annex A.6

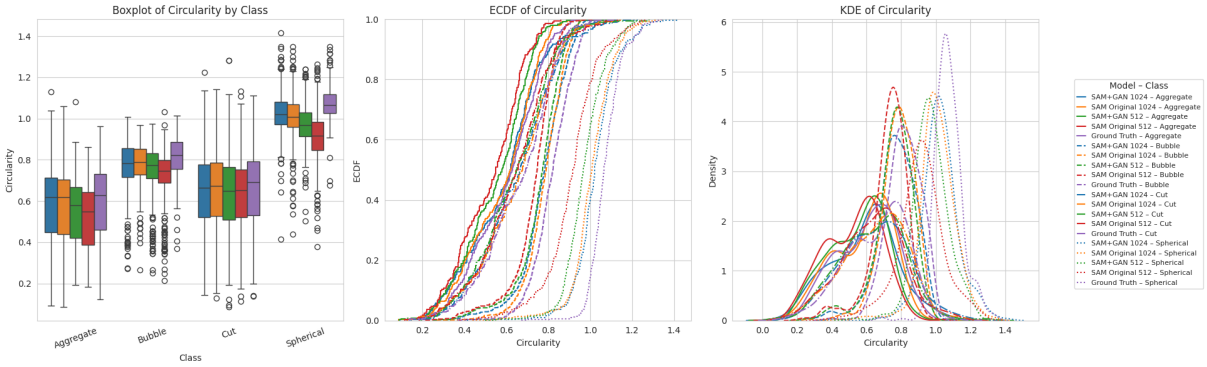

Figure 5.14: Sam Circularity metric characterization graphs (from left to right the boxplot, ECDF and KDE)

In the boxplots A.10, it is possible to see that 1024 px models have a box interquartile range (IQR) more similar to the ground truth. For SAM-LST trained on the Original dataset for the image size of 512 px, its boxplots are consistently bigger and skewed, with longer whiskers when compared to the ground truth. This means it's over-estimating variability and overshooting segmentation masks, for example, defining two close particles as one huge blob, or chopping a single particle into multiple fragments.

In ECDF plot A.11, the area and diameters seem to be relatively close to ground truth (GT). As for circularity, for the Spherical class is where most distinctions can be found. The original SAM ECDFs are too far left and often too shallow meaning they systematically underestimate how round particles really are and exaggerate shape variability. Augmented SAM ECDFs snap back toward the GT curves, by training on augmented data, the model learns to predict shapes whose roundness distribution nearly matches the true distribution in both level and spread.

In the kernel-density estimate (KDE) plot A.12 The most noticeable differences are again in the Spherical class. GAN appears to improve the metrics curve to the GT curve shape resemblance, especially in the Equivalent Diameter and Circularity.

5.7.2 TransUNet

Similarly to what was done previously the following picture represents the plots used to perform this study focused on one single metric with the rest being presented in the annex A.6

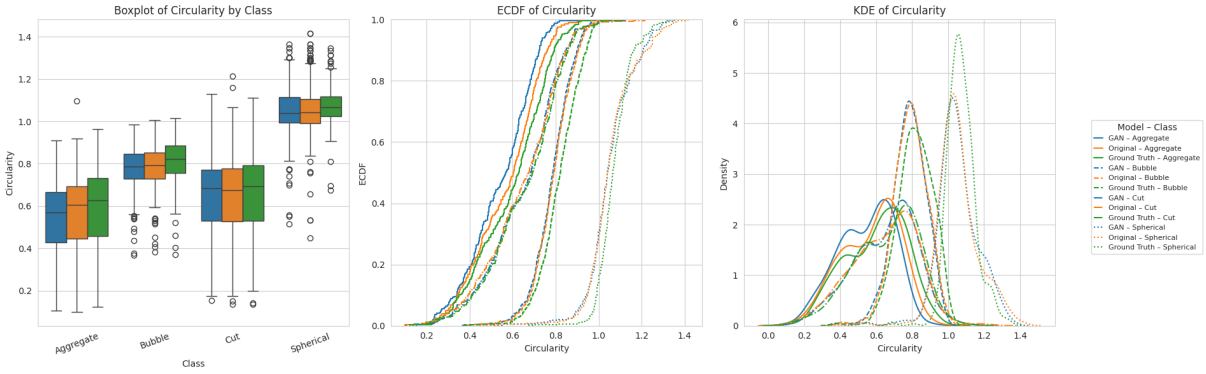

Figure 5.15: TransUnet Circularity metric characterization graphs (from left to right the boxplot, ECDF and KDE)

In the box-plots referring to the TransUNet models (FigureA.13) the biggest discrepancy can be found in the aggregate class, with the model trained on augmented images seemingly skewing the values up in metrics like Area Eq and Major Diameter and down on Circularity. This reveals a systematic bias introduced by the SR enhanced model towards predicting particles larger with less circularity in this class. This can be explained by the eventual fusion of different instances. In the circularity box-plot, regarding the classes Bubble and Spherical both models seem to generate instances less circular than the original mask.

In the ECDF graph (Figure A.14), the Aggregate class's circularity curve deviates most noticeably from the ground-truth curve, indicating that all models tend to underestimate its circularity. However, the value of this finding is limited since the "Aggregate" class inherently comprises multiple merged bubbles, producing complex irregular shapes whose circularity is not well defined.

In the KDE graph A.12, the curve for the Spherical class regarding the Circularity metric is slightly higher than the predicted models. This means that the models did not fully capture the circularity aspect of the spherical bubble class. Besides that, no major differences are found, meaning the model was effective, resulting in a good distribution of results.

Leveraging the quantization and visualization of these metrics, a deeper insight into the models' behavior is gained, allowing to pinpoint specific areas like shape retention with room to improve in multiclass segmentation architectures.

Chapter 6

Conclusions

In this work, we study the effectiveness of several deep learning models for the task of multiclass experimental image segmentation for two-phased flow systems. This is important since the study of this phenomenon affects a wide range of applications from enhancing energy conservation and minimizing equipment fouling to preventing hazardous conditions associated with improper flow or phase distribution in several industries.

To provide a comprehensive overview of the background knowledge related to this topic, the main concepts were defined, as well as an extensive overview of the existing models and their progression from the early stages of image segmentation to their current state and the challenges most commonly faced. A gap in the application of multiclass image segmentation models for the task of bubbly flow image analysis was identified, with most multiclass models being tailored to medical imaging.

In order to promote an easier approach for the application of multiclass image segmentation models to analyze bubbly flow images, training, validation, inference, and characterization pipelines were devised and described with the objective of developing an approachable way to compare different segmentation models. Using these pipelines, the effect of super-resolution preprocessing was assessed in order to find out whether segmentation performance improved with this technique in bubbly flows.

The dataset had 4 classes of bubbles (Bubble, Cut, Aggregate, and Spherical). Four model architectures were implemented: YOLO (You Only Look Once), Detectron2 , TransUNet, and SAM (Segment Anything Model). The best performing model was the ViT based TransUNet followed by SAM. Detectron2 came third and YOLO was the worst performing model.

The pipelines developed revealed to have a structure easily replicable between models. This improved efficiency when testing various configurations of the same model architecture when altering aspects like backbone, image input size, and, most importantly, training dataset between original and SR enhanced images. The effect of image enhancing had mixed results, with models benefiting from it in some classes and not having a difference or slightly degrading results in others.

To test generalization, the best-performing models were used to segment images from entirely new experiments, specifically, a maze-entrapment study that wasn't represented in the training set. After a straightforward image preprocessing step, the developed segmentation pipelines proved extremely useful for the intuitive application of the models, which performed well regarding the visual characteristics that were quite different from the training data. The images in this experiment were of extremely low quality and the SR-enhancement improved significantly the segmentation task. The resulting masks allowed for the extraction of fairly accurate bubble metrics from this experiment.

As for the research questions presented in the first chapter:

- **What are the most effective deep learning methods to perform multiclass bubbly-flow image segmentation?** Among the tested architectures, the TransUNet variant with a ViT-B backbone trained on the original dataset delivered the strongest overall performance, leading every evaluation metric except AP50, in which the SR-enhanced version was best.
- **How does Super-Resolution pre-processing affect segmentation performance in bubbly flows, particularly under low-quality or low-resolution imaging conditions?** When used to to train a model with limited resources and already high quality images the effectiveness of Super-Resolution has mixed results. It improves performance for some architectures while slightly degrading it for others, with no clear, consistent trend across all models. However, when the same SR workflow was transferred to a different test case (Bubble Entrapment Experiment) with lower quality images, it proved extremely valuable, enabling more precise feature extraction and yielding noticeably sharper, more accurate segmentation.
- **Can Deep Learning models for multiclass segmentation be effectively used to extract reliable geometric and physical characteristics of bubbles in multiphase**

flow images? Yes. Multiclass segmentation models, most notably the TransUNet with a ViT-B backbone, produce highly accurate masks that directly enable precise computation of bubble properties (number, area, perimeter, equivalent diameter, circularity, etc.). The strong metric scores across these descriptors confirm their reliability for quantitative bubble characterization.

- **Does the proposed end-to-end segmentation pipeline offer a generalizable framework for multiclass bubble analysis in multiphase flow experiments?** Yes. The proposed end-to-end segmentation pipeline provides a flexible, modular framework for multiclass bubble analysis across diverse multiphase-flow experiments. It standardizes data preparation, model training, and validation so that different architectures can be directly compared and readily deployed.

The significance of this research has already been recognized, with part of the work presented at an international conference (AI and Fluids Mechanics-Greece) and a paper currently in preparation. Building upon this, future improvements could include a further generalization of the developed pipelines in order to enable faster integration with new models, as well as the exploration of different architectures than the ones implemented; a cross-domain validation in other multiphase flow scenarios and datasets; a deeper analysis of the flow's characteristics regarding bubble behavior such as mass/heat transfers and even partially omitted bubble reconstruction.

Due to effects of SR-enhancing not having consistent improvements across all models and classes, further studies should also be performed in this regard. Based on the premise that a network capable of capturing finer-grained details will leverage greater benefits of this technique, by increasing depth whether through additional convolutional layers, residual blocks, or attention mechanisms it is possible that improvements in segmentation accuracy, particularly on subtle boundaries and small features might be uncovered using SR techniques. Moreover, a study on weather the annotation precision is significantly impacting the discovery of trends in segmentation could be performed, as it could provide a deeper understanding of the effect of SR.

Furthermore, systematically evaluating a broader range of state-of-the-art architectures like U-Net++, DeepLabv3+, or other transformer-based encoders could reveal which design choices work best with GAN augmentation. Such an analysis would not only identify the most SR-'friendly' topology but also inform guidelines for selecting or tailoring models to maximize performance gains.

Appendix A

Appendix

A.1 Metrics Description

Appendix: Evaluation Metrics

The performance of the segmentation models was evaluated using several standard metrics. Let TP, TN, FP, and FN represent the number of true positives, true negatives, false positives, and false negatives, respectively. All metrics are computed at the pixel level unless otherwise noted.

- **Accuracy**:
$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$
Measures the proportion of correctly labeled pixels among all pixels.

- **Precision**:
$$\text{Precision} = \frac{TP}{TP + FP}$$
Indicates the proportion of predicted positive pixels that are actually positive.

- **Recall (Sensitivity)**:
$$\text{Recall} = \frac{TP}{TP + FN}$$

Measures the ability of the model to identify all actual positive pixels.

- **F1 Score**:

$$F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Harmonic mean of precision and recall, especially useful for imbalanced datasets.

- **Intersection over Union (IoU)**:

$$\text{IoU}_c = \frac{TP_c}{TP_c + FP_c + FN_c}$$

Also known as the Jaccard Index, it quantifies the overlap between predicted and ground truth masks.

- **Mean Intersection over Union (mIoU)**:

$$\text{mIoU} = \frac{1}{C} \sum_{c=1}^{C} \frac{TP_c}{TP_c + FP_c + FN_c}$$

Average of class-wise IoUs across all C classes.

- **Dice Coefficient**:

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

Also called the Sørensen–Dice index, it emphasizes the overlap between prediction and ground truth masks.

- **Mean Squared Error (MSE)**:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Where y_i and $\hat{y}_i$ are the ground truth and predicted values at pixel i, and N is the total number of pixels. Used primarily for soft mask comparison or regression-based segmentation.

- **Average Precision (AP) at IoU=0.50**:

A custom implementation of AP is used to evaluate instance segmentation. A prediction is considered correct if it matches a ground truth instance with an IoU of at least 0.50. For each class, precision and recall are computed across multiple confidence thresholds, and

the precision-recall curve is integrated to yield:

$$\mathrm{AP}_{50} = \int_0^1 \mathrm{Precision}(r)\, dr \quad \text{at IoU} = 0.50$$

- **Mean Average Precision (mAP)**:

  Mean Average Precision is computed by averaging AP over multiple IoU thresholds ranging from 0.50 to 0.95 with a step size of 0.05:

$$\mathrm{mAP} = \frac{1}{10} \sum_{k=0}^{9} \mathrm{AP}_{0.50+0.05k}$$

  Matching of predicted instances to ground truth is performed on a per-image basis, and predictions are ranked by confidence to construct the precision-recall curve.

A.2 Developed Pipeline Diagrams

The Training Pipeline starts with model definition, environment initialization, and training data loading. After verifying dataset–model compatibility, the training loop is initialized. The pipeline concludes with saving the training metrics and model weights.

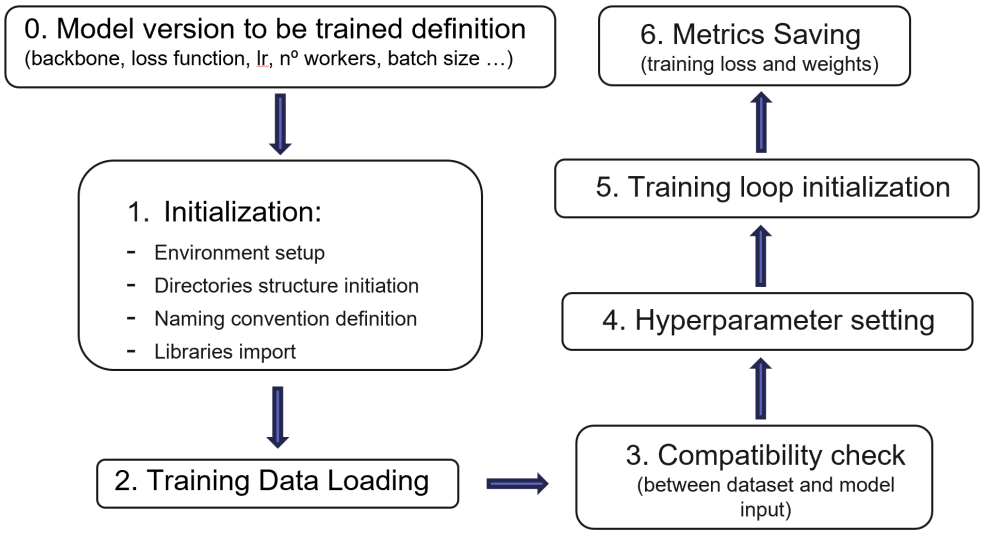

Figure A.1: Training Pipline diagram

The following is the Testing Pipeline diagram with workflow divided into two main blocks: (1) model-dependent steps; and (2) model-independent steps, common to all architectures (per-image inference, semantic and instance evaluation, metrics computation, and final results saving)

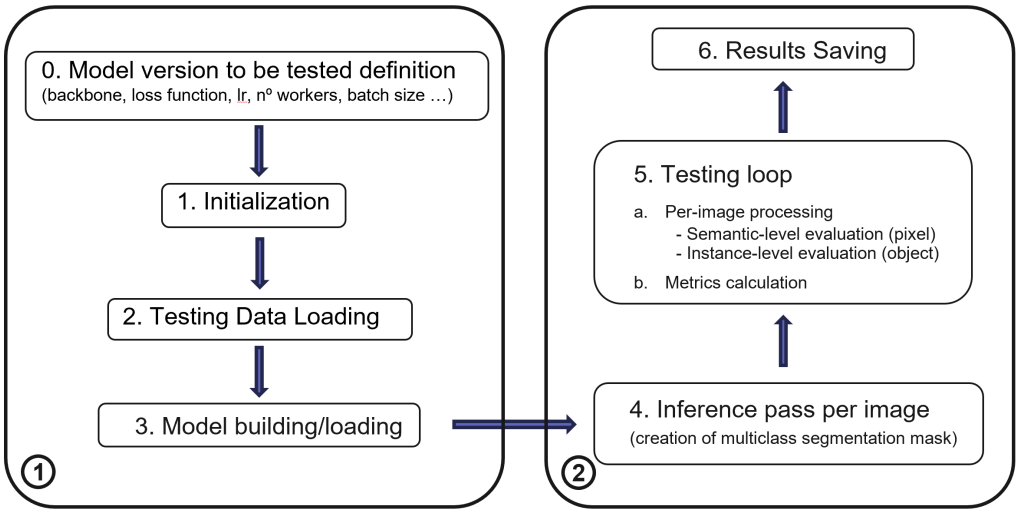

Figure A.2: Testing pipeline diagram

A.3 YOLO v8 Model Arquitecture

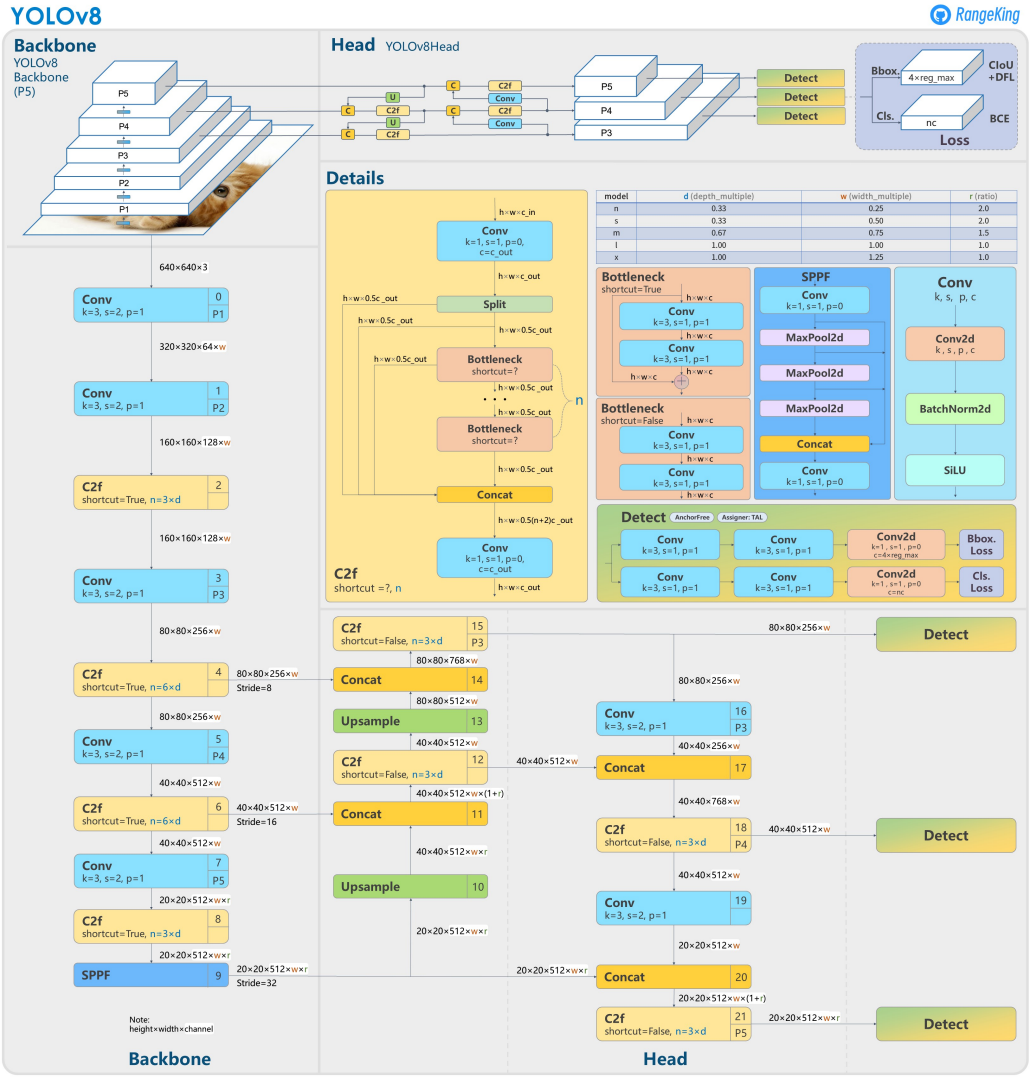

Figure A.3: YOLO v8 object detection architecture [64]. The rectangles represent layers, with the labels describing the type of layer (Conv, Upsample, etc.) and any relevant parameters (kernel size, number of channels, etc.). The arrows represent data flow between layers, with the direction of the arrow indicating the flow of data from one layer to the next.

A.4 Inference Results

Other inference results of the models can be observed in the following figures:

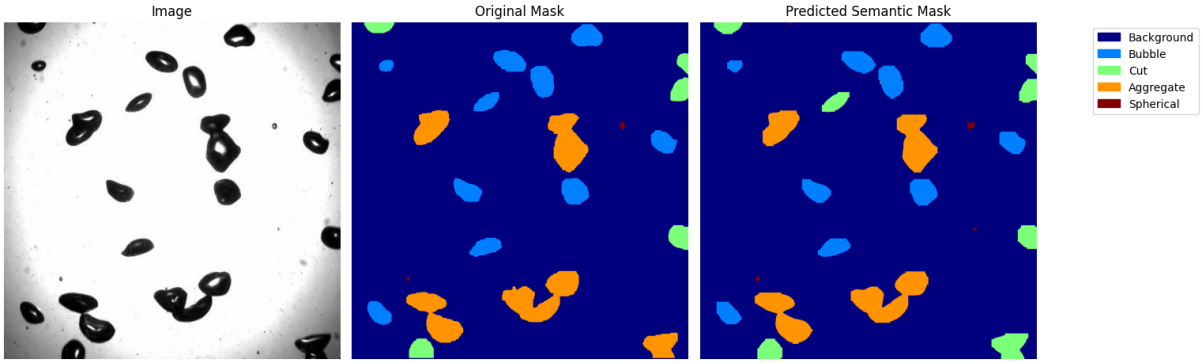

Figure A.4: YOLO inference example 2

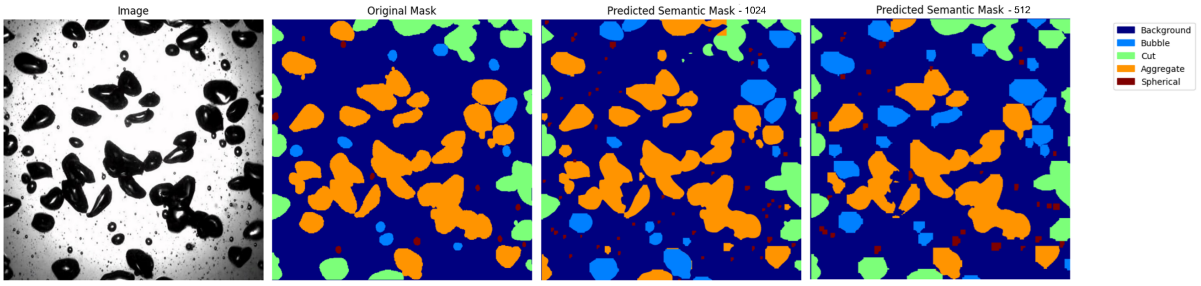

Figure A.5: YOLO inference bounding box size limitation example

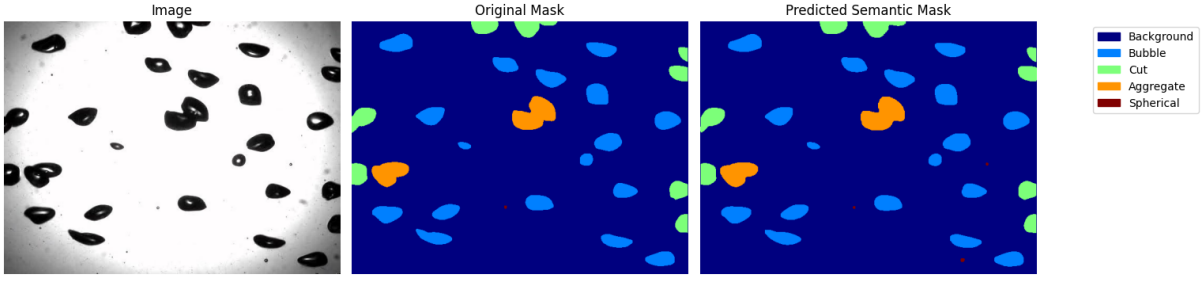

Figure A.6: Detectron2 inference example 2

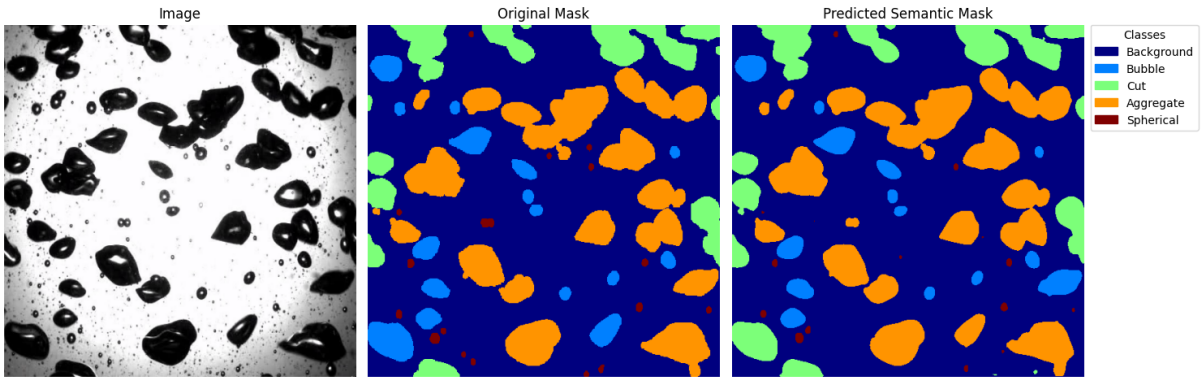

Figure A.7: TransUNet ViT-B inference example 2

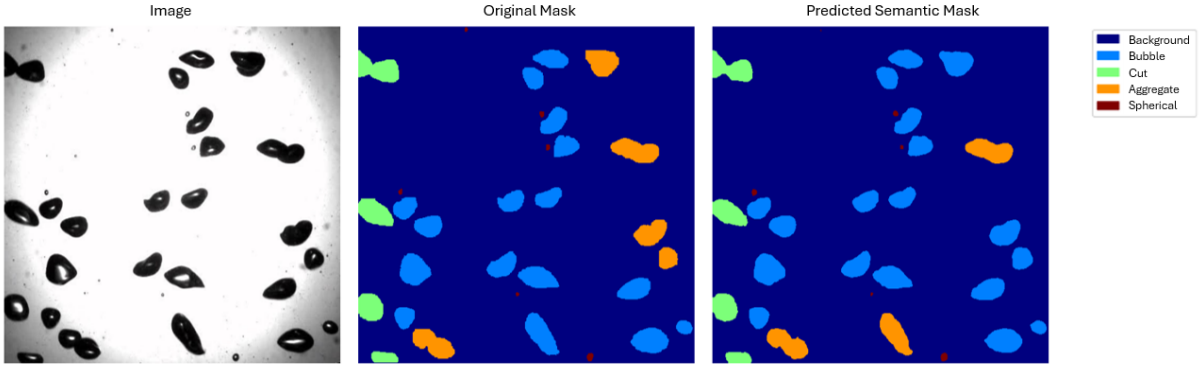

Figure A.8: SAM ViT-B inference example 2

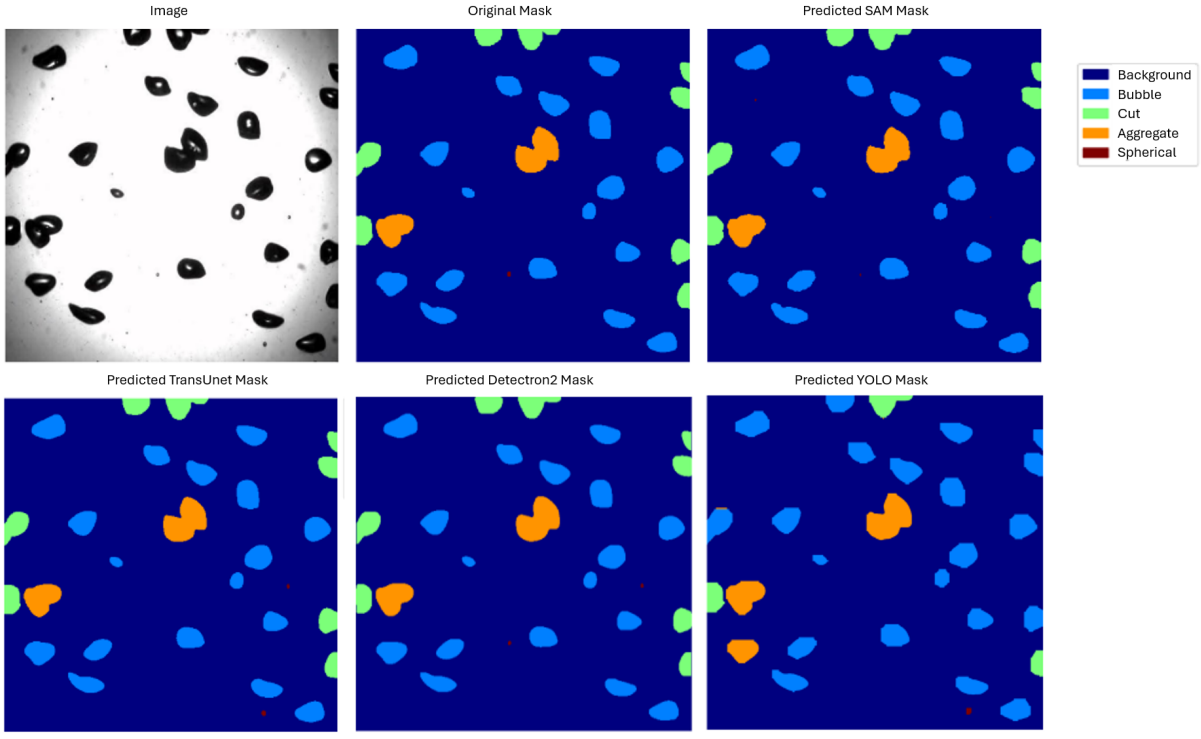

Figure A.9: Inference example All models side byside comparison

A.5 Validation Results

Table A.1: Class-wise Segmentation Performance of the models YOLOv8m-seg and YOLO11m-seg

Class	IoU	AP50	mAP50-95	Pixel Acc.	Pixel Prec.	Pixel Recall	Model	Data Type
Background	0.976			0.981	0.994	0.983	YOLOv8m-seg	original
Bubble	0.607	0.452	0.370	0.973	0.685	0.841	YOLOv8m-seg	original
Cut	0.318	0.223	0.168	0.968	0.686	0.513	YOLOv8m-seg	original
Aggregate	0.641	0.217	0.187	0.960	0.754	0.813	YOLOv8m-seg	original
Spherical	0.436	0.546	0.208	0.997	0.506	0.798	YOLOv8m-seg	original
Background	0.975			0.981		0.983	YOLOv8m-seg	SR enhanced
Bubble	0.635	0.626	0.505	0.975	0.689	0.894	YOLOv8m-seg	SR enhanced
Cut	0.344	0.229	0.167	0.971	0.843	0.420	YOLOv8m-seg	SR enhanced
Aggregate	0.625	0.277	0.241	0.960	0.735	0.854	YOLOv8m-seg	SR enhanced
Spherical	0.407	0.473	0.176	0.997	0.456	0.812	YOLOv8m-seg	SR enhanced
Background	0.977			0.982	0.994	0.984	YOLOv11m-seg	original
Bubble	0.618	0.519	0.421	0.975	0.721	0.821	YOLOv11m-seg	original
Cut	0.416	0.350	0.263	0.976	0.816	0.589	YOLOv11m-seg	original
Aggregate	0.683	0.363	0.318	0.967	0.771	0.880	YOLOv11m-seg	original
Spherical	0.441	0.624	0.241	0.997	0.548	0.697	YOLOv11m-seg	original
Background	0.975			0.981	0.992	0.985	YOLOv11m-seg	SR enhanced
Bubble	0.577	0.434	0.357	0.972	0.721	0.731	YOLOv11m-seg	SR enhanced
Cut	0.294	0.221	0.166	0.968	0.752	0.413	YOLOv11m-seg	SR enhanced
Aggregate	0.576	0.208	0.186	0.953	0.685	0.865	YOLOv11m-seg	SR enhanced
Spherical	0.321	0.274	0.104	0.996	0.372	0.545	YOLOv11m-seg	SR enhanced

Table A.2: Segmentation Metrics by Class, Model, and Data Type for Detectron2 models

Class	IoU	AP50	mAP50-95	Pixel Acc.	Pixel Prec.	Pixel Recall	Backbone	Data Type
Background	0.981	–	–	0.985	0.989	0.993	R50_FPN_3x	Original
Bubble	0.717	0.706	0.634	0.983	0.790	0.882	R50_FPN_3x	Original
Cut	0.698	0.497	0.418	0.985	0.902	0.722	R50_FPN_3x	Original
Aggregate	0.697	0.564	0.510	0.971	0.834	0.828	R50_FPN_3x	Original
Spherical	0.542	0.744	0.429	0.999	0.737	0.679	R50_FPN_3x	Original
Background	0.980	–	–	0.985	0.989	0.992	R50_FPN_3x	SR enhanced
Bubble	0.718	0.687	0.617	0.982	0.779	0.887	R50_FPN_3x	SR enhanced
Cut	0.707	0.582	0.491	0.986	0.869	0.788	R50_FPN_3x	SR enhanced
Aggregate	0.692	0.542	0.489	0.971	0.856	0.804	R50_FPN_3x	SR enhanced
Spherical	0.536	0.714	0.398	0.998	0.722	0.661	R50_FPN_3x	SR enhanced
Background	0.980	–	–	0.985	0.988	0.993	R101_FPN_3x	Original
Bubble	0.707	0.838	0.752	0.982	0.797	0.856	R101_FPN_3x	Original
Cut	0.686	0.426	0.357	0.986	0.889	0.756	R101_FPN_3x	Original
Aggregate	0.690	0.434	0.392	0.971	0.836	0.825	R101_FPN_3x	Original
Spherical	0.535	0.745	0.421	0.998	0.734	0.660	R101_FPN_3x	Original
Background	0.980	–	–	0.985	0.988	0.993	R101_FPN_3x	SR enhanced
Bubble	0.701	0.784	0.705	0.982	0.805	0.842	R101_FPN_3x	SR enhanced
Cut	0.682	0.522	0.437	0.984	0.834	0.768	R101_FPN_3x	SR enhanced
Aggregate	0.671	0.466	0.422	0.968	0.823	0.798	R101_FPN_3x	SR enhanced
Spherical	0.540	0.748	0.427	0.999	0.749	0.656	R101_FPN_3x	SR enhanced

Table A.3: Class-wise Segmentation Performance of TransUNet_R50+ViT-B_16 on SR enhanced and Original Data

Class	IoU	AP50	mAP50-95	Pixel Acc.	Pixel Prec.	Pixel Recall	Data Type
Background	0.983	–	–	0.987	0.989	0.995	Original
Bubble	0.748	0.887	0.809	0.983	0.817	0.869	Original
Cut	0.787	0.844	0.712	0.989	0.886	0.866	Original
Aggregate	0.769	0.858	0.790	0.977	0.904	0.835	Original
Spherical	0.458	0.607	0.335	0.998	0.749	0.650	Original
Background	0.982	–	–	0.986	0.987	0.996	SR enhanced
Bubble	0.726	0.872	0.793	0.982	0.795	0.871	SR enhanced
Cut	0.751	0.864	0.719	0.988	0.872	0.852	SR enhanced
Aggregate	0.731	0.884	0.804	0.974	0.906	0.796	SR enhanced
Spherical	0.462	0.592	0.327	0.998	0.670	0.667	SR enhanced

Table A.4: Per-class segmentation performance metrics for SAM-LST ViT-B and ViT-L models with and without SR enhancing, at 512 px and 1024 px image sizes.

Class	IoU	AP50	mAP50-95	Pixel Acc.	Pixel Prec.	Pixel Recall	Model Name	Data Type	Image Size
Background	0.981	–	–	0.986	0.989	0.993	SAM_LST_ViT-B	original	1024 px
Bubble	0.647	0.911	0.806	0.980	0.864	0.728	SAM_LST_ViT-B	original	1024 px
Cut	0.708	0.736	0.610	0.983	0.866	0.731	SAM_LST_ViT-B	original	1024 px
Aggregate	0.718	0.697	0.620	0.967	0.779	0.881	SAM_LST_ViT-B	original	1024 px
Spherical	0.458	0.497	0.273	0.998	0.632	0.745	SAM_LST_ViT-B	original	1024 px
Background	0.981	–	–	0.985	0.990	0.991	SAM_LST_ViT-B	SR enhanced	1024 px
Bubble	0.648	0.855	0.749	0.977	0.803	0.743	SAM_LST_ViT-B	SR enhanced	1024 px
Cut	0.741	0.655	0.545	0.985	0.781	0.922	SAM_LST_ViT-B	SR enhanced	1024 px
Aggregate	0.667	0.709	0.629	0.967	0.836	0.789	SAM_LST_ViT-B	SR enhanced	1024 px
Spherical	0.383	0.433	0.222	0.998	0.674	0.651	SAM_LST_ViT-B	SR enhanced	1024 px
Background	0.974	–	–	0.980	0.984	0.992	SAM_LST_ViT-B	original	512 px
Bubble	0.673	0.816	0.668	0.977	0.730	0.881	SAM_LST_ViT-B	original	512 px
Cut	0.725	0.793	0.607	0.985	0.824	0.848	SAM_LST_ViT-B	original	512 px
Aggregate	0.663	0.844	0.704	0.968	0.901	0.722	SAM_LST_ViT-B	original	512 px
Spherical	0.339	0.423	0.167	0.997	0.634	0.463	SAM_LST_ViT-B	original	512 px
Background	0.978	–	–	0.983	0.989	0.990	SAM_LST_ViT-B	SR enhanced	512 px
Bubble	0.656	0.860	0.731	0.978	0.775	0.813	SAM_LST_ViT-B	SR enhanced	512 px
Cut	0.711	0.735	0.588	0.982	0.758	0.882	SAM_LST_ViT-B	SR enhanced	512 px
Aggregate	0.653	0.857	0.735	0.969	0.868	0.771	SAM_LST_ViT-B	SR enhanced	512 px
Spherical	0.444	0.454	0.212	0.998	0.689	0.559	SAM_LST_ViT-B	SR enhanced	512 px
Background	0.981	–	–	0.986	0.988	0.994	SAM_LST_ViT-L	original	1024 px
Bubble	0.603	0.940	0.834	0.978	0.901	0.641	SAM_LST_ViT-L	original	1024 px
Cut	0.658	0.661	0.549	0.978	0.693	0.909	SAM_LST_ViT-L	original	1024 px
Aggregate	0.665	0.655	0.583	0.965	0.819	0.779	SAM_LST_ViT-Ll	original	1024 px
Spherical	0.512 px	0.557	0.320	0.998	0.696	0.700	SAM_LST_ViT-L	original	1024 px
Background	0.981	–	–	0.986	0.990	0.992	SAM_LST_ViT-L	SR enhanced	1024 px
Bubble	0.666	0.856	0.756	0.980	0.796	0.820	SAM_LST_ViT-L	SR enhanced	1024 px
Cut	0.654	0.667	0.553	0.981	0.771	0.821	SAM_LST_ViT-L	SR enhanced	1024 px
Aggregate	0.632	0.703	0.629	0.967	0.837	0.787	SAM_LST_ViT-Ll	SR enhanced	1024 px
Spherical	0.418	0.485	0.259	0.998	0.693	0.599	SAM_LST_ViT-L	SR enhanced	1024 px

Table A.5: Global segmentation performance metrics across all model versions

Dice	Accuracy	Precision	Recall	F1 Score	MSE	AP50	AP50-95	Mean IoU	Model Name	Backbone	Data Type	Image Size
0.777	0.957	0.817	0.819	0.817	0.176	0.663	0.536	0.684	SAM-LST	ViT-B	SR enhanced	1024 px
0.806	0.957	0.826	0.815	0.817	0.169	0.710	0.577	0.702	SAM-LST	ViT-B	original	1024 px
0.800	0.955	0.816	0.803	0.807	0.177	0.726	0.567	0.688	SAM-LST	ViT-B	SR enhanced	512 px
0.791	0.954	0.815	0.781	0.792	0.199	0.719	0.537	0.675	SAM-LST	ViT-B	original	512 px
0.776	0.956	0.817	0.804	0.810	0.166	0.678	0.549	0.670	SAM-LST	ViT-L	SR enhanced	1024 px
0.795	0.952	0.820	0.805	0.805	0.173	0.704	0.571	0.684	SAM-LST	ViT-L	original	1024 px
0.826	0.964	0.846	0.836	0.840	0.152	0.803	0.661	0.730	TransUNet	R50+ViT-B_16	SR enhanced	–
0.837	0.968	0.869	0.843	0.855	0.138	0.799	0.662	0.749	TransUNet	R50+ViT-B_16	original	–
0.827	0.961	0.843	0.826	0.833	0.159	0.631	0.499	0.727	Detectron2	R_50_FPN_3x	SR enhanced	–
0.829	0.961	0.850	0.821	0.833	0.156	0.628	0.498	0.727	Detectron2	R_50_FPN_3x	original	–
0.821	0.959	0.840	0.811	0.825	0.166	0.630	0.498	0.715	Detectron2	R_101_FPN_3x	SR enhanced	–
0.823	0.961	0.849	0.818	0.832	0.161	0.611	0.481	0.720	Detectron2	R_101_FPN_3x	original	–
0.708	0.942	0.744	0.792	0.740	0.205	0.401	0.272	0.597	YOLO	v8m-seg	SR enhanced	–
0.706	0.940	0.725	0.790	0.746	0.202	0.359	0.233	0.595	YOLO	v8m-seg	original	–
0.666	0.936	0.704	0.708	0.691	0.230	0.284	0.203	0.549	YOLO	v11m-seg	SR enhanced	–
0.739	0.949	0.770	0.794	0.775	0.187	0.464	0.311	0.627	YOLO	v11m-seg	original	–

A.6 Bubble Characterization

Bubble characterization results that allow for a better understanding of model performance when comparing the models' version distribution and the ground truth distribution obtained from the original masks hand-generated.

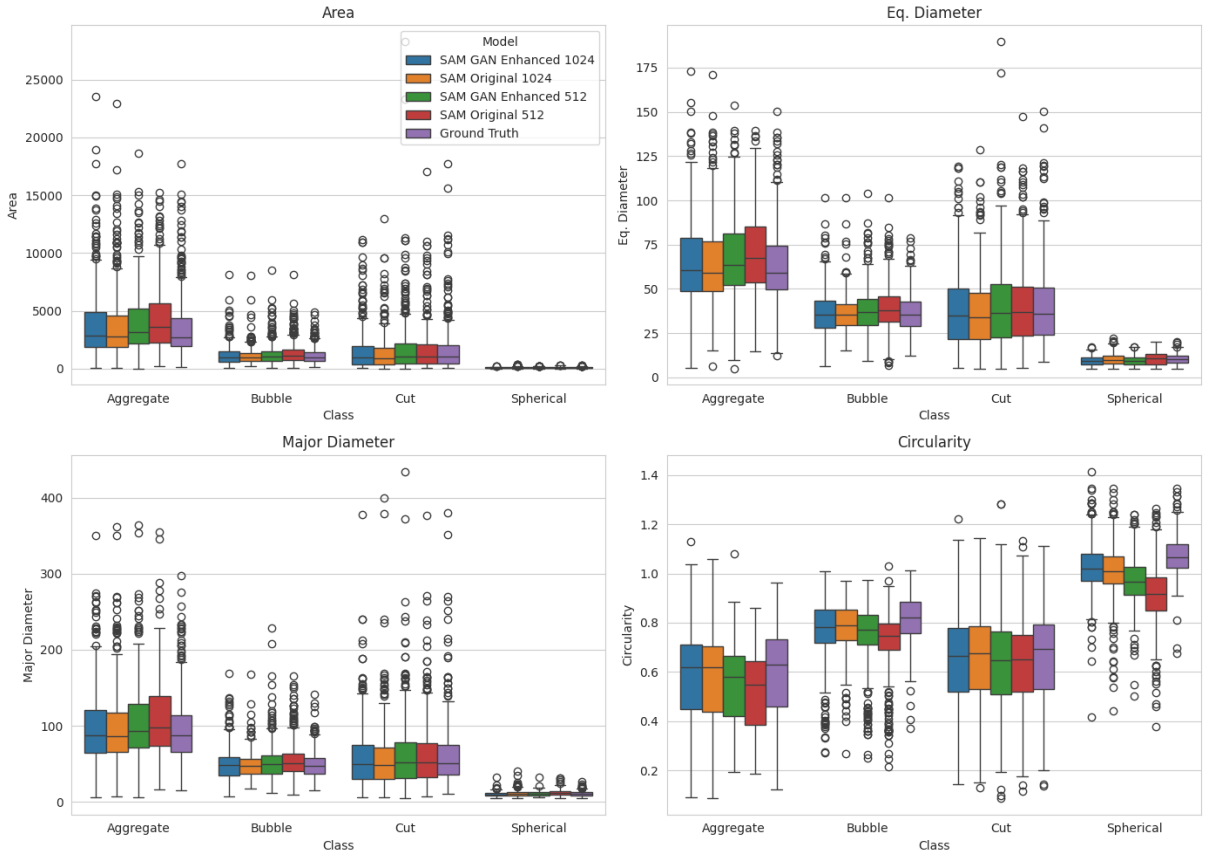

Figure A.10: SAM models metrics boxplot description

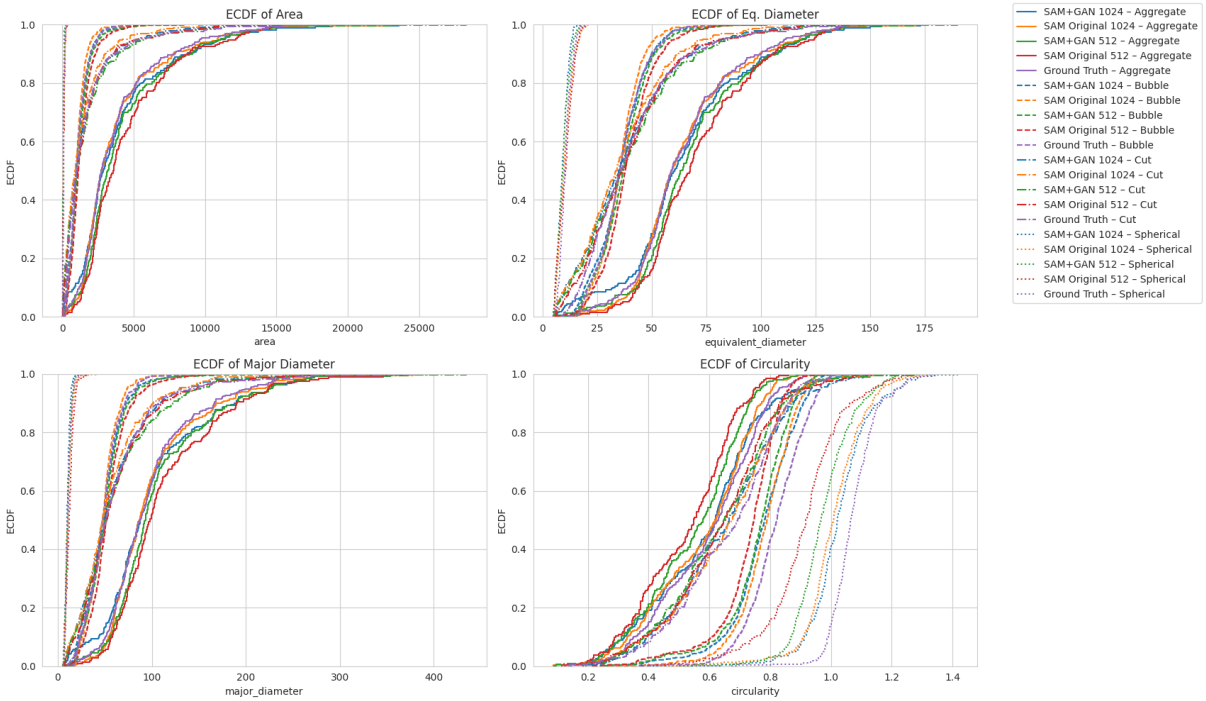

Figure A.11: SAM models ecdf plot

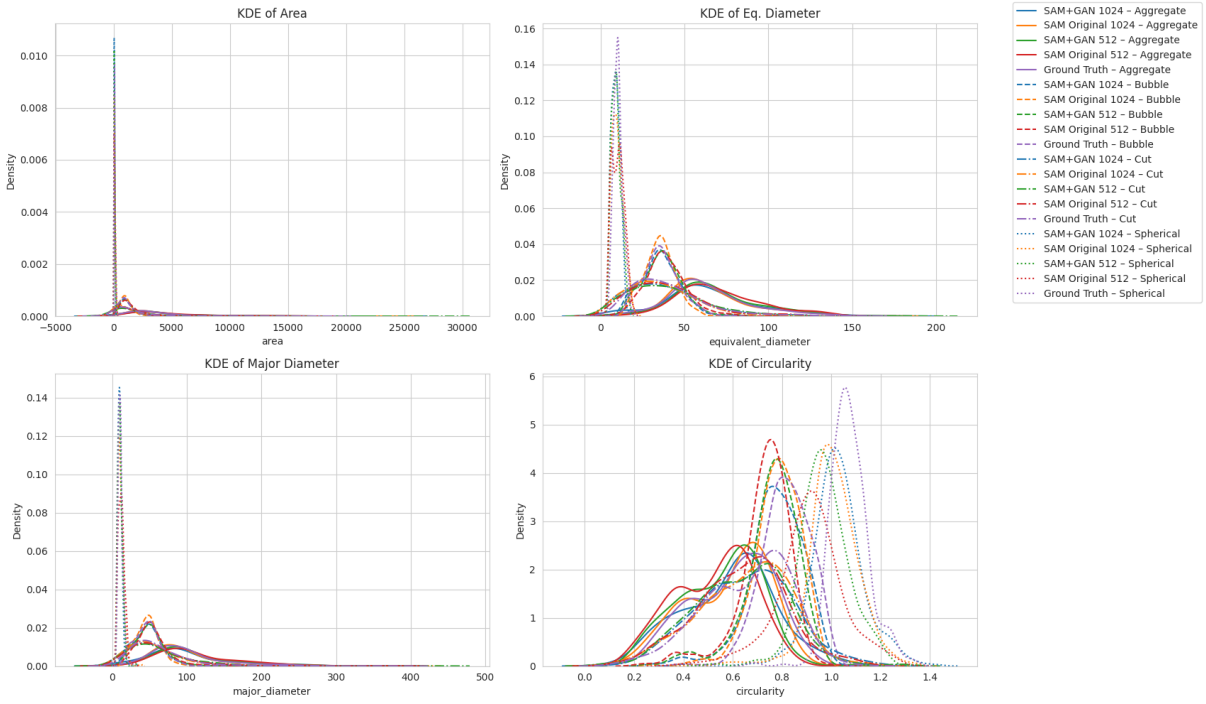

Figure A.12: SAM models kde plot

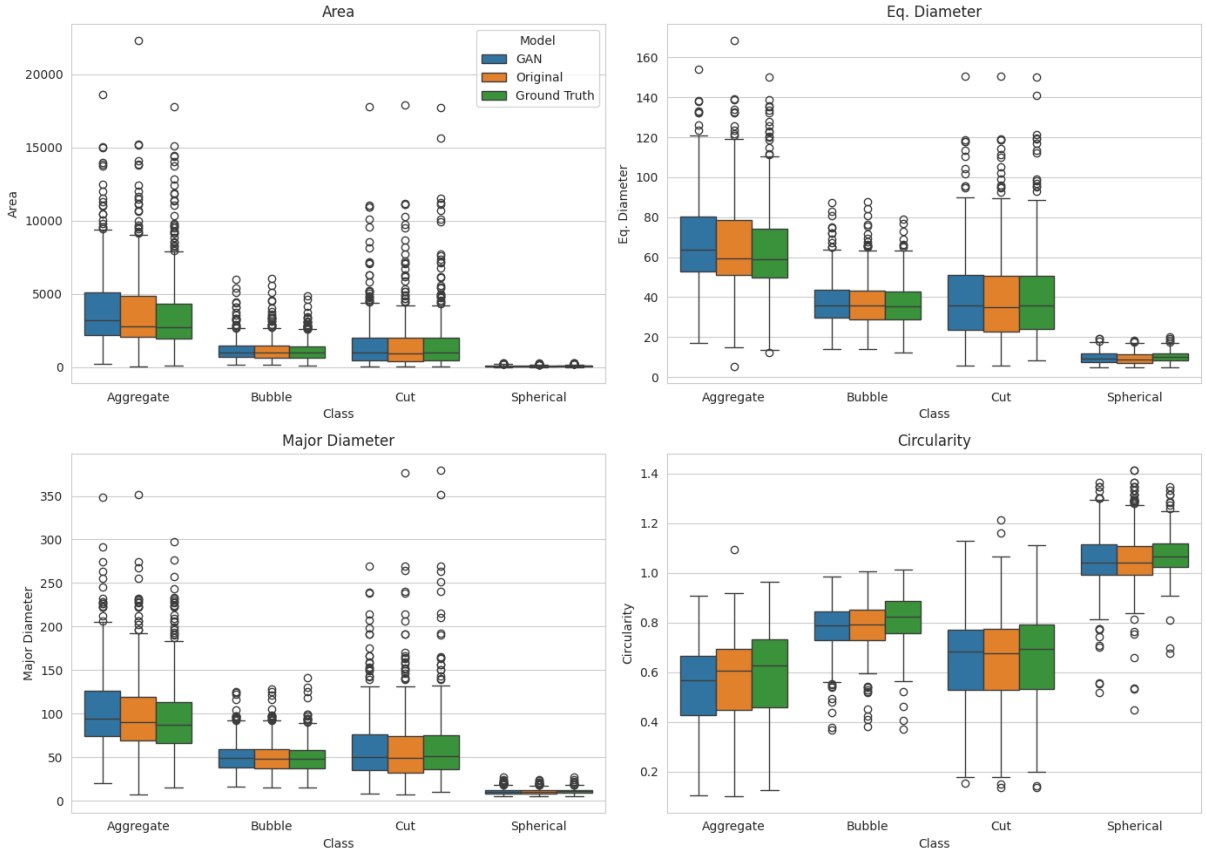

Figure A.13: TransUNet models metrics boxplot description

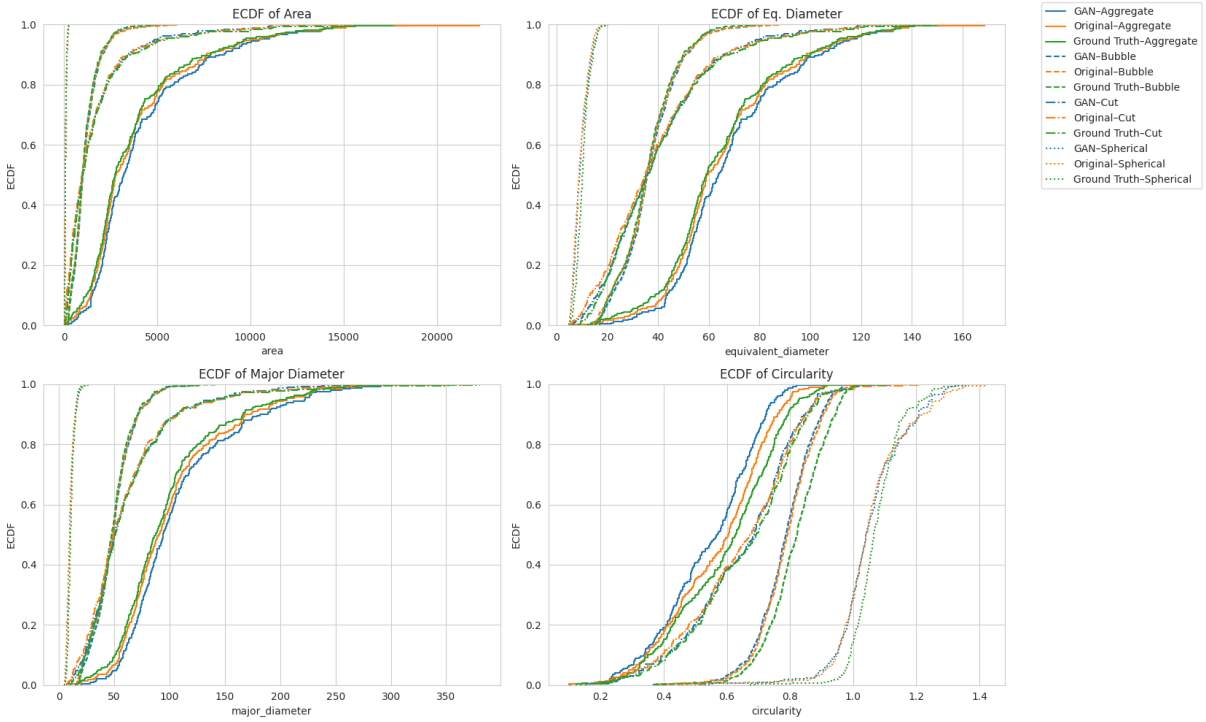

Figure A.14: TransUNet models ecdf plot

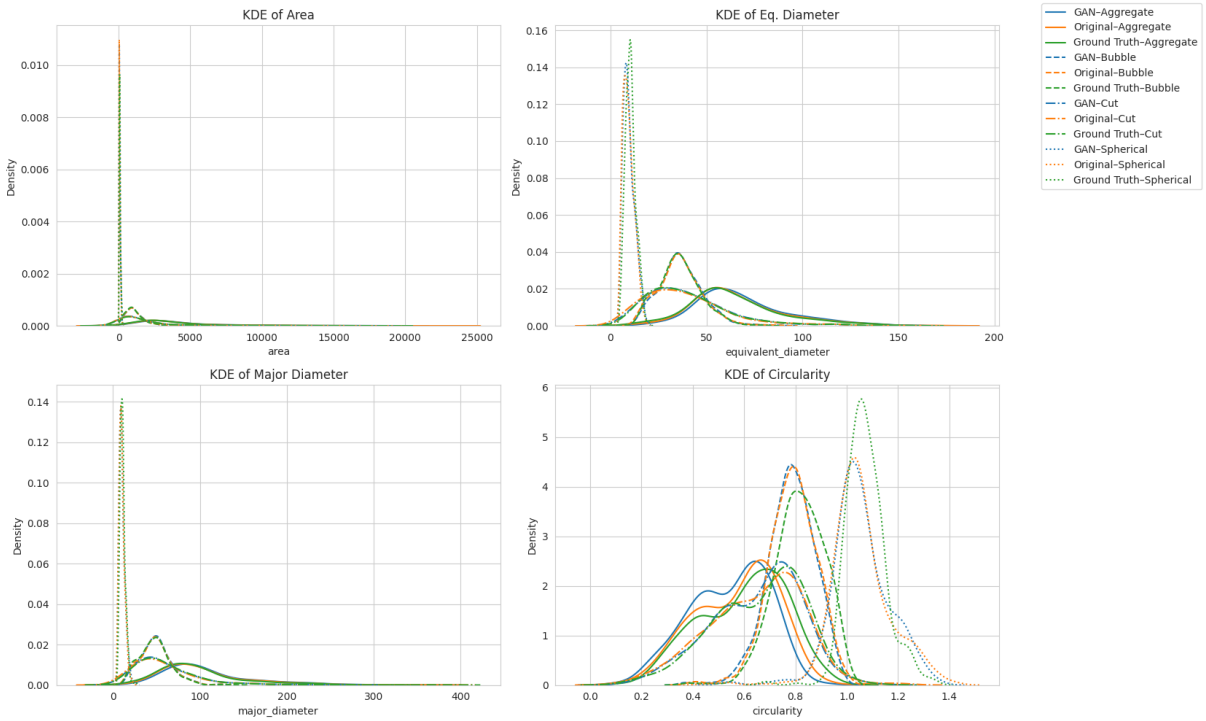

Figure A.15: TransUNet models kde plot

Bibliography

[1] Maithra Raghu and Eric Schmidt. A survey of deep learning for scientific discovery, 2020. URL https://arxiv.org/abs/2003.11755.

[2] J. M. Coulson and J. F. Richardson, editors. *Coulson and Richardson's Chemical Engineering*. Elsevier, 7th edition, 2017.

[3] C. E. Brennen. *Fundamentals of Multiphase Flows*. Cambridge University Press, 2005.

[4] Yehuda Taitel, Dvora Barnea, and A. E. Dukler. Modelling flow pattern transitions for steady upward gas–liquid flow in vertical tubes. *AIChE Journal*, 26(3):345–354, 1980. doi: https://doi.org/10.1002/aic.690260304. URL https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/aic.690260304.

[5] Mamoru Ishii and Takashi Hibiki. *Thermo-Fluid Dynamics of Two-Phase Flow*. Springer, 2nd edition, 2010. doi: 10.1007/978-1-4419-7985-8. URL https://link.springer.com/book/10.1007/978-1-4419-7985-8.

[6] Bo Liu, Qiuli Li, Wenhao Sun, and Yili Wang. Mechanisms and modeling of bubble dynamic behaviors and mass transfer under gravity: A review. *Chemical Engineering Science*, 275:118653, 2023. doi: 10.1016/j.ces.2023.118653. URL https://www.researchgate.net/publication/370692646_Mechanisms_and_Modeling_of_Bubble_Dynamic_Behaviors_and_Mass_Transfer_under_Gravity_A_Review.

[7] Neves do Amaral, Andreia. *Towards a detailed understanding of oxygen transfer in wastewater treatment: the effect of bubble size distribution*. PhD thesis, Ghent University, 2019.

[8] R.C. Bueno, P.H.F. Masotti, J.F. Justo, D.A. Andrade, M.S. Rocha, W.M. Torres, and R.N. de Mesquita. Two-phase flow bubble detection method applied to natural circulation system using fuzzy image processing. *Nuclear Engineering and Design*, 335:255–264, 2018.

ISSN 0029-5493. doi: https://doi.org/10.1016/j.nucengdes.2018.05.026. URL https://www.sciencedirect.com/science/article/pii/S0029549318306125.

[9] Volfango Bertola. *Two-Phase Flow Measurement Techniques*. 01 2003. ISBN 978-3-211-20757-4. doi: 10.1007/978-3-7091-2538-0_6.

[10] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979. doi: 10.1109/TSMC.1979.4310076.

[11] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15:11–15, 1972. URL https://api.semanticscholar.org/CorpusID:1105637.

[12] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986. doi: 10.1109/TPAMI.1986.4767851.

[13] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Pearson, Boston, 4th edition, 2018. ISBN 9780133356724.

[14] Nikhil R Pal and Sankar K Pal. A review on image segmentation techniques. *Pattern Recognition*, 26(9):1277–1294, 1993. ISSN 0031-3203. doi: https://doi.org/10.1016/0031-3203(93)90135-J. URL https://www.sciencedirect.com/science/article/pii/003132039390135J.

[15] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Prentice Hall, 3rd edition, 2008. ISBN 9780131687288.

[16] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015. URL https://arxiv.org/abs/1505.04597.

[17] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation, 2016. URL https://arxiv.org/abs/1511.00561.

[18] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs, 2016. URL https://arxiv.org/abs/1412.7062.

[19] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016. URL https://arxiv.org/abs/1506.02640.

[20] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *arXiv preprint arXiv:2001.05566*, 2020. URL https://arxiv.org/abs/2001.05566.

[21] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. doi: 10.1162/neco.1989.1.4.541.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105, 2012.

[23] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014. URL https://arxiv.org/abs/1409.4842.

[24] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. URL https://arxiv.org/abs/1409.1556.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

[26] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size, 2016. URL https://arxiv.org/abs/1602.07360.

[27] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017. URL https://arxiv.org/abs/1704.04861.

[28] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020. URL https://arxiv.org/abs/1905.11946.

[29] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2015. URL https://arxiv.org/abs/1411.4038.

[30] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++: A nested u-net architecture for medical image segmentation, 2018. URL https://arxiv.org/abs/1807.10165.

[31] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation, 2017. URL https://arxiv.org/abs/1706.05587.

[32] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation, 2018. URL https://arxiv.org/abs/1802.02611.

[33] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014. URL https://arxiv.org/abs/1311.2524.

[34] Ross Girshick. Fast r-cnn, 2015. URL https://arxiv.org/abs/1504.08083.

[35] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016. URL https://arxiv.org/abs/1506.01497.

[36] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. https://github.com/facebookresearch/detectron, 2018.

[37] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2017. URL https://arxiv.org/abs/1612.03144.

[38] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018. URL https://arxiv.org/abs/1708.02002.

[39] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2018. URL https://arxiv.org/abs/1703.06870.

[40] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

[41] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics yolov8, 2023. URL https://github.com/ultralytics/ultralytics.

[42] Glenn Jocher and Jing Qiu. Ultralytics yolo11, 2024. URL https://github.com/ultralytics/ultralytics.

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

[44] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. URL https://arxiv.org/abs/2010.11929.

[45] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows, 2021. URL https://arxiv.org/abs/2103.14030.

[46] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything, 2023. URL https://arxiv.org/abs/2304.02643.

[47] Yewon Kim and Hyungmin Park. Deep learning-based automated and universal bubble detection and mask extraction in complex two-phase flows. *Scientific Reports*, 11, 04 2021. doi: 10.1038/s41598-021-88334-0.

[48] Yizhou Cui, Chengxiang Li, Wanli Zhang, Xiaoqi Ning, Xiaogang Shi, Jinsen Gao, and Xingying Lan. A deep learning-based image processing method for bubble detection, segmentation, and shape reconstruction in high gas holdup sub-millimeter bubbly flows. *Chemical Engineering Journal*, 449:137859, 07 2022. doi: 10.1016/j.cej.2022.137859.

[49] H. Hessenkemper, S. Starke, Y. Atassi, T. Ziegenhein, and D. Lucas. Bubble identification from images with machine learning methods. *International Journal of Multiphase Flow*, 155: 104169, October 2022. ISSN 0301-9322. doi: 10.1016/j.ijmultiphaseflow.2022.104169. URL http://dx.doi.org/10.1016/j.ijmultiphaseflow.2022.104169.

[50] Daizhou Wen, Wuguang Chen, Junlian Yin, Yuchen Song, Mingjun Ren, and Dezhong Wang. Overlapping bubble detection and tracking method based on convolutional neural network and kalman filter. *Chemical Engineering Science*, 263:118059, 2022. ISSN 0009-2509. doi: https://doi.org/10.1016/j.ces.2022.118059. URL https://www.sciencedirect.com/science/article/pii/S0009250922006431.

[51] Jerol Soibam, Valentin Scheiff, Ioanna Aslanidou, Konstantinos Kyprianidis, and Rebei Bel Fdhila. Application of deep learning for segmentation of bubble dynamics in subcooled

boiling. *International Journal of Multiphase Flow*, 169:104589, 2023. ISSN 0301-9322. doi: https://doi.org/10.1016/j.ijmultiphaseflow.2023.104589. URL https://www.sciencedirect.com/science/article/pii/S0301932223002094.

[52] Youngjoon Suh, Sanghyeon Chang, Peter Simadiris, Tiffany B. Inouye, Muhammad Jahidul Hoque, Siavash Khodakarami, Chirag Kharangate, Nenad Miljkovic, and Yoonjin Won. Vision-it: A framework for digitizing bubbles and droplets. *Energy and AI*, 15:100309, 2024. ISSN 2666-5468. doi: https://doi.org/10.1016/j.egyai.2023.100309. URL https://www.sciencedirect.com/science/article/pii/S2666546823000812.

[53] Sanghyeon Chang, Youngjoon Suh, Chinmay Shingote, Cho-Ning Huang, Issam Mudawar, Chirag Kharangate, and Yoonjin Won. Bubblemask: Autonomous visualization of digital flow bubbles for predicting critical heat flux. *International Journal of Heat and Mass Transfer*, 217:124656, 2023. ISSN 0017-9310. doi: https://doi.org/10.1016/j.ijheatmasstransfer.2023.124656. URL https://www.sciencedirect.com/science/article/pii/S0017931023008013.

[54] Jonggyu Lee, Youngjoon Suh, Max Kuciej, Peter Simadiris, Michael T. Barako, and Yoonjin Won. Deep vision-inspired bubble dynamics on hybrid nanowires with dual wettability, 2022. URL https://arxiv.org/abs/2202.09417.

[55] Tim Haas, Christian Schubert, Moritz Eickhoff, and Herbert Pfeifer. Bubcnn: Bubble detection using faster rcnn and shape regression network. *Chemical Engineering Science*, 216:115467, 04 2020. doi: 10.1016/j.ces.2019.115467.

[56] Irina Nizovtseva, Pavel Mikushin, Ilya Starodumov, Ksenia Makhaeva, Simon Kraev, and Dmitrii Chernushkin. Bubble detection in multiphase flows through computer vision and deep learning for applied modeling. *Mathematics*, 12(23), 2024. ISSN 2227-7390. doi: 10.3390/math12233864. URL https://www.mdpi.com/2227-7390/12/23/3864.

[57] Qizhou Kang, Feng Ye, Qin Li, Ru Li, Jianfeng Wang, Haoliang Wang, Hui Yu, Jingcai Cheng, Xiangyang Li, and Chao Yang. Bubble boundary r-cnn: A multitask model for segmenting and reconstructing overlapping bubbles. *Separation and Purification Technology*, 358:130300, 2025. ISSN 1383-5866. doi: https://doi.org/10.1016/j.seppur.2024.130300. URL https://www.sciencedirect.com/science/article/pii/S1383586624040395.

[58] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial

network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 105–114, 2017. doi: 10.1109/CVPR.2017.19.

[59] M.C. Neves, J. Filgueiras, Z. Kokkinogenis, M.C.F. Silva, J.B.L.M. Campos, and L.P. Reis. Enhancing experimental image quality in two-phase bubbly systems with super-resolution using generative adversarial networks. *International Journal of Multiphase Flow*, 180:104952, 2024. ISSN 0301-9322. doi: https://doi.org/10.1016/j.ijmultiphaseflow.2024.104952. URL https://www.sciencedirect.com/science/article/pii/S0301932224002295.

[60] Jieneng Chen, Jieru Mei, Xianhang Li, Yongyi Lu, Qihang Yu, Qingyue Wei, Xiangde Luo, Yutong Xie, Ehsan Adeli, Yan Wang, et al. Transunet: Rethinking the u-net architecture design for medical image segmentation through the lens of transformers. *Medical Image Analysis*, page 103280, 2024.

[61] Kaidong Zhang and Dong Liu. Customized segment anything model for medical image segmentation. *arXiv preprint arXiv:2304.13785*, 2023.

[62] Shurong Chai, Rahul Kumar Jain, Shiyu Teng, Jiaqing Liu, Yinhao Li, Tomoko Tateyama, and Yen wei Chen. Ladder fine-tuning approach for sam integrating complementary network, 2023. URL https://arxiv.org/abs/2306.12737.

[63] Jieneng Chen, Yongyi Lu, Qihang Yu, Xiangde Luo, Ehsan Adeli, Yan Wang, Le Lu, Alan L. Yuille, and Yuyin Zhou. Transunet: Transformers make strong encoders for medical image segmentation, 2021. URL https://arxiv.org/abs/2102.04306.

[64] RangeKing. Comment on "instance segmentation in yolov8". https://github.com/ultralytics/ultralytics/issues/189, 2023. GitHub issue comment.

[65] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks, 2015. URL https://arxiv.org/abs/1507.06228.

[66] Hiroto Honda. Digging into detectron2. https://medium.com/@hirotoschwert/digging-into-detectron-2-47b2e794fabd, 2019. Medium article.

[67] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. Pointrend: Image segmentation as rendering, 2020. URL https://arxiv.org/abs/1912.08193.

[68] Christy Dunlap, Changgen Li, Hari Pandey, Ngan Le, and Han Hu. Bubbleid: A deep learning framework for bubble interface dynamics analysis, 2024. URL https://arxiv.org/abs/2405.07994.

[69] André Colliard-Granero, Keusra A. Gompou, Christian Rodenbücher, Kourosh Malek, Michael H. Eikerling, and Mohammad J. Eslamibidgoli. Deep learning-enhanced characterization of bubble dynamics in proton exchange membrane water electrolyzers. *Phys. Chem. Chem. Phys.*, 26:14529–14537, 2024. doi: 10.1039/D3CP05869G. URL http://dx.doi.org/10.1039/D3CP05869G.

[70] Ahmed A. Taha and Allan Hanbury. Metrics for evaluating 3d medical image segmentation: analysis, selection, and tool. *BMC Medical Imaging*, 15(1):29, 2015. doi: 10.1186/s12880-015-0068-x. URL https://doi.org/10.1186/s12880-015-0068-x.

[71] COCO Consortium. Coco detection evaluation metrics. https://cocodataset.org/#detection-eval, 2024. Accessed: 2025-06-15.

[72] Ultralytics. Ultralytics yolo: Real-time object detection. https://github.com/ultralytics/ultralytics, 2023.

[73] Ultralytics. Yolov11: Ultralytics real-time object detection models. https://docs.ultralytics.com/models/yolo11/, 2025. Accessed on June 19, 2025.

[74] Rahima Khanam and Muhammad Hussain. Yolov11: An overview of the key architectural enhancements, 2024. URL https://arxiv.org/abs/2410.17725.

[75] Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. Lst: Ladder side-tuning for parameter and memory efficient transfer learning, 2022. URL https://arxiv.org/abs/2206.06522.

[76] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

[77] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. Yolox: Exceeding yolo series in 2021. *arXiv preprint arXiv:2107.08430*, 2021.