INTELLIGENT VIDEO SEGMENTATION FOR LESION DETECTION IN GASTROINTESTINAL ENDOSCOPY

by

LI YINAN (D-C1-2780-5)

ZHANG HANYU (D-C1-2705-4)

B.Sc. in Electromechanical Engineering

2024/2025

Faculty of Science and Technology
University of Macau

INTELLIGENT VIDEO SEGMENTATION FOR LESION DETECTION IN GASTROINTESTINAL ENDOSCOPY

by

LI YINAN (D-C1-2780-5)

ZHANG HANYU (D-C1-2705-4)

Graduation Project Report submitted in partial
fulfillment of the requirements for the degree of

BSc. in Electromechanical Engineering

Faculty of Science and Technology
University of Macau

2024/2025

University of Macau

Abstract

INTELLIGENT VIDEO SEGMENTATION FOR LESION DETECTION IN GASTROINTESTINAL ENDOSCOPY

by

LI YINAN (D-C1-2780-5)

ZHANG HANYU (D-C1-2705-4)

Project Supervisor
Prof. WONG PAK KIN

BSc. in Electromechanical Engineering

Digestive diseases, including gastroesophageal reflux disease (GERD) and upper gastrointestinal (GI) polyps, represent a significant global health burden due to their high prevalence and potential for adverse outcomes. GERD, primarily caused by lower esophageal sphincter dysfunction, manifests as acid reflux and heartburn, with progression to complications such as esophagitis and esophageal ulcers. Gastric polyps—benign or precancerous mucosal proliferations—are frequently asymptomatic but may lead to obstruction, bleeding, or malignant transformation in advanced cases. Early detection of these conditions by using endoscopy is critical for these two kinds of diseases.

Traditional GI endoscopy relies on clinician expertise and subjective visual assessment for lesion detection, a process prone to variability due to image quality, procedure duration, and operator fatigue. Although recent advances in deep learning (DL) have enhanced diagnostic accuracy, the majority of the existing models focus on static image analysis.

To address these limitations, we propose a model called SAMClass for real-time lesion localization and classification using continuous endoscopy video inputs. It is the first attempt to integrate a classification head into the decoder of Segmentation Anything Model (SAM) for lesion segmentation and classification simultaneously in endoscopic video. By adding a classification head to this state-of-the-art SAM and training on diverse, large-scale endoscopic datasets consisting of GERD, gastric polyps, and heathy GI videos, the new model can concurrently identify and segment lesions with high precision, providing clinicians with clinically actionable insights. Additionally, the proposed model is compared cooperatively with UNet, which is a traditional segmentation model and widely acted as baseline model for comparison. Ablation studies are then carried out to optimize the proposed model. The SAMClass has a final Intersection Over Union (IoU) value of 0.7642, compared to 0.4196 for the UNet. It also obtains the classification accuracy of 0.9808, demonstrating the effectiveness of our proposed model.

TABLE OF CONTENTS

LIST OF FIGURES i
LIST OF TABLES ii
LIST OF ABBREVIATIONS iii
CHAPTER 1: INTRODUCTION 1
  1.1 BACKGROUND 1
CHAPTER 2: LITERATURE REVIEW ON GASTROINTESTINAL ENDOSCOPY AND ITS INTELLIGENT DIAGNOSIS 4
  2.1 OVERVIEW OF GI DISEASES 4
  2.2 OVERVIEW OF MACHINE LEARNING IN MEDICAL IMAGE ANALYSIS 4
  2.3 LITERATURE REVIEW ON SEGMENTATION MODELS FOR UPPER ENDOSCOPY 7
  2.4 REVIEW OF SAM MODEL 10
  2.5 CLINICAL MOTIVATION OF THIS STUDY AND PROJECT OBJECTIVES 12
CHAPTER 3: METHODOLOGY 14
  3.2 SOFTWARE DEVELOPMENT 15
  3.3 DATA SELECTION AND LABELLING 15
    3.3.1 SELF-COLLECTED VIDEOS 16
    3.3.2 OPEN-SOURCED VIDEOS 18
  3.4 DATA PREPROCESSING AND AUGMENTATION 22
    3.4.1 DATA AUGMENTATION 22
    3.4.2 DATASET CREATION AND SPLITTING 23
  3.5 MODELS 24
    3.5.1 PROPOSED MODEL 24
    3.5.2 MODEL FOR COMPARISON AND ITS SELECTION RATIONALE 26
  3.6 LOSS FUNCTION 28

3.6.1 DICE LOSS 29
3.6.2 FOCAL LOSS 31
3.6.3 SEGMENTATION LOSS FOR UNET 33
3.6.4 MULTI-TASK LOSS FUNCTION 34
CHAPTER 4: EXPERIMENT AND RESULTS 36
4.1 EXPERIMENT 36
4.1.1 OBJECTIVES AND EXPECTATION 36
4.1.2 TRAINING CONFIGURATIONS AND PARAMETERS 36
4.1.3 EXPERIMENTAL DESIGN 37
4.1.4 EVALUATION METRICS 38
4.2 RESULTS 41
4.2.1 ABLATION STUDY OF USING DIFFERENT LEARNING RATES 41
4.2.2 ABLATION STUDY ON USING DIFFERENT SCHEDULERS .. 48
4.2.3 ABLATION STUDY ON USING DIFFERENT WEIGHTS FOR MUTI-TASK LOSS FUNCTION IN SAMCLASS 53
4.2.4 ABLATION STUDY ON USING DIFFERENT DROPOUT RATES FOR SAMCLASS MODEL 54
4.2.5 QUANTITATIVE SEGMENTATION PERFORMANCE COMPARISON BETWEEN SAMCLASS AND UNET 57
4.2.6 QUALITATIVE RESULTS OF SAMCLASS AND UNET 59
4.3 DISCUSSION 61
CHAPTER 5: CONCLUSION AND FUTURE WORK 63
REFERENCES 65
APPENDIX I: WORK BREAKDOWN 75
APPENDIX II: PYTHON CODE FOR MASK GENERATION 76
APPENDIX III: PYTHON CODE FOR VIDEO TEXT REMOVAL 79
APPENDIX IV: PYTHON CODE FOR CONTINUOUS FRAME DETECTION ... 82
APPENDIX V: PYTHON CODE FOR THE UNET MODEL 85
APPENDIX VI: PYTHON CODE FOR TRAINING THE UNET MODEL............. 89
APPENDIX VII: PYTHON CODE FOR VALIDATING UNET MODEL 95

APPENDIX VIII: PYTHON CODE FOR SAMCLASS MODEL 98
APPENDIX IX: PYTHON CODE FOR TRAINING SAMCLASS MODEL.........102
APPENDIX X: PYTHON CODE FOR TRAINING SAMCLASS MODEL 115

LIST OF FIGURES

Figure 1: Upper GI examination by (Mayo Clinic, n.d.) 2
Figure 2: Examples of original frame with text and the processed frame without text. 18
Figure 3: Workflow of orderFinds.py for continuous frame reconstruction. 19
Figure 4: Overall structure of proposed SAMClass model................................... 24
Figure 5: Overall structure of UNet model. 27
Figure 6: Training and validation losses for UNet model under three learning rates 44
Figure 7: Training and validation losses for SAMClass model under three learning rates 47
Figure 8: Training and validation loss of using different schedulers on UNet model 50
Figure 9: Training and validation losses of using different schedulers on SAMClass model.. 52
Figure 10: Training and validation losses for different dropout rates 57

LIST OF TABLES

Table 1: Comparisons between relative projects 9

Table 2: Data distribution of video frames of the final dataset.................... 21

Table 3: Training parameters that are kept constant throughout all training......... 36

Table 4: Ablation study on IoU with high and low learning rates.................... 41

Table 5: Ablation study on different schedulers with a high learning rate............ 48

Table 6: Different weighting factors v.s. training results 53

Table 7: Different dropout rates v.s. training results 54

Table 8: Configurations of the best models 58

Table 9: Models with the best classification and segmentation performances...... 58

Table 10: Class-wise mIoU of optimized SAMClass model 58

Table 11: Gastric polyp visualization results for both optimized models of continuous frames, ... 59

Table 12: GERD visualization results for both optimized models of continuous frames 60

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
BCE	Binary cross entropy loss
CNNs	Convolutional Neural Network
dHash	Perceptual hashes
DL	Deep learning
FL	Focal loss
FPS	Frames per second
GERD	Gastroesophageal reflux disease
GI	Gastrointestinal disease
IoU	Intersection over union
mIoU	Mean intersection over union
ML	Machine Learning
HD	High Definition
NCC	Normalized cross-correlation
RE	reflux esophagitis
ReLU	Rectified linear unit
ROI	Region of interest
SAM	Segment Anything Model
ViT	Vision Transformer

ACKNOWLEDGMENTS

We wish to extend our deepest gratitude to our advisor, Professor Wong Pak Kin, whose expert guidance, profound insights, and constructive feedback shaped every phase of this project. Without his visionary leadership and careful revisions, the successful completion of this endeavor would not be possible. We are equally indebted to our senior colleague, Ms. Chan In Neng, for her unwavering support throughout the research process. Her technical expertise and mentorship were really helpful. Furthermore, we would also like to thank the Kiang Wu Hospital, Macao for providing the video data for this project. Finally, we acknowledge all individuals who provided encouragement and support to us.

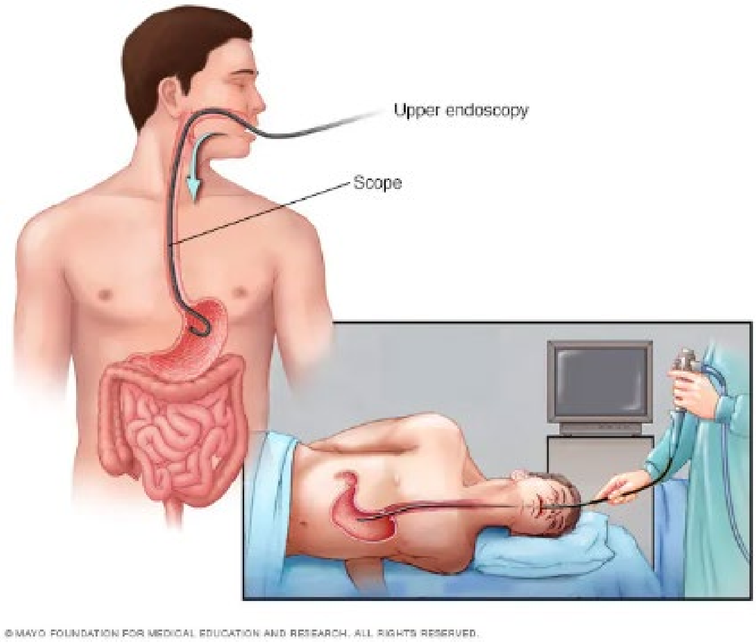
CHAPTER 1: INTRODUCTION

1.1 BACKGROUND

Gastrointestinal (GI) disorders, including reflux esophagitis (RE) and GI polyps, represent significant global health challenges due to their prevalence and potential for malignant progression (Sloan & Katz, 2019; Xu et al., 2024). Reflux esophagitis, a manifestation of gastroesophageal reflux disease (GERD), results from chronic esophageal mucosal exposure to gastric acid, and lead to inflammation, erosions, and complications such as Barrett's esophagus—a precursor to esophageal adenocarcinoma (Clavero et al., 2006). GERD affects over 10% of the global population, with regional variations linked to lifestyle factors, obesity, and dietary habits (Izzaturrahmah et al., 2021). Accurate severity grading of RE under the Los Angeles (LA) classification system remains challenging, particularly for inexperienced clinicians, due to subjectivity in interpreting subtle mucosal changes.

Similarly, gastric polyps—abnormal mucosal growths in the digestive tract—pose diagnostic challenges. While most are benign, adenomatous polyps carry significant malignant potential, and they contribute to colorectal cancer, the third most common cancer globally (De Santiago et al., 2016). Current screening protocols, such as endoscopy, depend on clinician expertise for polyp detection, yet studies report miss rates of 20–30% for small or flat lesions (Ma & Bourke, 2017). These limitations highlight the need for standardized, objective diagnostic tools to improve early detection and risk stratification.

Endoscopy is a minimal invasion medical procedure allowing doctors to visually examine the body from inside using an endoscope, which is a flexible tube equipes with light source and camera (Esposito & Cappabianca, 2013). GI endoscopy is to insert an endoscope through the mouth to perform upper GI inspection (Hughes, 2021). Real-time videos are played to doctor as the criteria of diagnosis. An illustrative picture of upper GI endoscopy examination is displayed in Figure 1.



© MAYO FOUNDATION FOR MEDICAL EDUCATION AND RESEARCH. ALL RIGHTS RESERVED.

Figure 1: Upper GI examination by (Mayo Clinic, n.d.)

Artificial intelligence (AI) integration into endoscopy offers transformative potential to address these challenges. Deep convolutional neural networks (CNNs) demonstrate proficiency in automating lesion detection, segmentation, and classification. For example, AI systems achieve over 90% accuracy in classifying RE severity under the LA system and identifying polyp morphology (e.g., sessile vs. pedunculated) (Nie et al., 2024; Cho et al., 2019). Such advancements reduce inter-observer variability and provide real-time decision support, enhance diagnostic consistency and clinical training.

By leveraging computational innovations, this study advances in computer-aided analysis in upper GI endoscopy by novelly proposed a multi-task lesion detection framework SAMClass, which realizes the simultaneous lesion segmentation and classification ability, improving patient outcomes via early interventions and personalized therapies.

CHAPTER 2: LITERATURE REVIEW ON GASTROINTESTINAL ENDOSCOPY AND ITS INTELLIGENT DIAGNOSIS

2.1 OVERVIEW OF GI DISEASES

Gastroesophageal reflux disease (GERD) and GI polyps are prevalent digestive disorders with distinct etiologies, clinical implications, and diagnostic challenges. GERD, characterized by chronic acid reflux due to lower esophageal sphincter dysfunction or excessive gastric acid secretion, presents with symptoms such as heartburn and regurgitation, often progressing to complications like Barrett's esophagus (Kellerman & Kintanar, 2017). Its pathogenesis involves multifactorial contributors, including obesity, dietary habits, anatomical anomalies, and genetic predispositions (Katzka & Kahrilas, 2020). Gastric polyps, conversely, are mucosal growths with variable malignancy risks. While hyperplastic polyps are typically benign, adenomatous polyps exhibit significant neoplastic potential and serve as precursors to gastric cancer (Bhattacharjee & Chakraborty, 2023). Polyp development is driven by genetic mutations, chronic inflammation, and environmental factors such as smoking.

2.2 OVERVIEW OF MACHINE LEARNING IN MEDICAL IMAGE ANALYSIS

Machine learning (ML) and deep learning (DL) have transformed medical image analysis through automated lesion detection, segmentation, and classification, improving diagnostic accuracy and workflow efficiency. Traditional ML methods, reliant on handcrafted features, are limited by poor generalizability across diverse datasets and anatomical variations. The emergence of DL, particularly convolutional neural networks (CNNs), addresses these limitations by enabling end-to-end learning from raw pixel data. For example, the encoder-decoder architecture of UNet has become a cornerstone in medical segmentation, capturing spatial hierarchies and localizing fine details in MRI and CT scans (Ahmadi et al., 2023; Rana & Bhushan, 2022).

However, challenges, such as dataset bias, limit cross-modality generalizability and reliance on large, annotated datasets. These issues are amplified in clinical settings like GI endoscopy, where robust early detection of polyps or pre-neoplastic lesions is critical for patient outcomes (Martins et al., 2023). Despite DL models achieving high accuracy in controlled benchmarks, performance degradation in heterogeneous clinical data remains problematic due to underrepresented rare pathologies and misaligned research-clinical priorities (Martins et al., 2023). A meta-analysis of 478 Alzheimer's studies revealed that larger datasets did not enhance real-world diagnostic accuracy, highlighting the disconnection between academic benchmarks and clinical utility. Similarly, DL systems for colorectal cancer detection reduce missed diagnoses but struggle with false positives and hardware-dependent generalizability (Martins et al., 2023).

Traditional endoscopic diagnosis relies on subjective visual assessment, which is a process prone to variability, particularly among trainees, and this limitation highlights the necessity for intelligent systems in real-time GI endoscopic examination. By employing advanced computer vision algorithms, such as deep learning-based semantic segmentation, these systems automate real-time lesion localization and characterization (e.g., erosive regions in GERD or polyp margins) within endoscopic video streams. Imagine a doctor moving an endoscope inside the patient's body—the screen flashes dozens of frames per second. If only analyzing single frames, it likes taking random snapshots from a movie, making it hard to track changes in lesions caused by tissue movement or blood flow. Video segmentation connects these "isolated moments," using time-based analysis to detect key details that might be blurry, hidden, or misjudged in single frames. Architectures like U-Net or Transformer-based models improve precision by distinguishing subtle pathological features from complex mucosal backgrounds under dynamic imaging conditions.

Technically, video segmentation also saves computing power. Single-frame methods process each image independently, but video frames share repetitive information (e.g., stomach lining textures stay similar across frames). Lightweight models (e.g., MobileNet) with hardware acceleration (e.g., GPU) allow the system to run smoothly on standard medical devices—no expensive hardware is needed. This efficiency is critical clinically: doctors see real-time AI annotations overlaid on the endoscopy screen, like an "AI dynamic highlighter," speeding up diagnosis.

Intelligent video segmentation integration enhances diagnostic consistency and expedites high-risk lesion identification, enabling timely interventions. For GERD, automated quantification of erosive areas aligned with the Los Angeles classification reduces grading discrepancies (Ge et al., 2023). Similarly, polyp segmentation systems improve adenoma detection rates, directly advancing cancer prevention strategies (Spadaccini et al., 2025). As endoscopic imaging evolves toward higher resolution and multimodal data acquisition, AI-clinical synergy redefines GI disease management standards, addressing global gaps in diagnostic accuracy and accessibility.

2.3 LITERATURE REVIEW ON SEGMENTATION MODELS FOR UPPER ENDOSCOPY

AI applications in medical diagnosis demonstrate the potential to enhance diagnostic accuracy and reduce clinician workload (Nia et al., 2023). Deep learning (DL), a validated approach in clinical settings, has achieved success across medical domains, including GI disease analysis (Fan et al., 2023; He et al., 2024). For example, the UNet model was employed to grade gastroesophageal reflux disease (GERD) severity, achieving a 42.64% intersection-over-union (IoU) score (Tran et al., 2022). However, this study was limited to a dataset of 795 static images. Similarly, a hybrid Transformer-

CNN architecture (Tang et al., 2023) achieved 67.40% IoU in segmentation and 96.94% classification accuracy for GI diseases but utilized only 1645 static images, neglecting the dynamic nature of endoscopic videos.

There are a few studies that explore video segmentation of UGI diseases, and the majority are related to polyp segmentation. Pogorelov et al. (Pogorelov et al., 2017) proposed an automatic diagnosis system that integrated a multi-class UGI abnormality classification subsystem with a localization subsystem that marked potential disease locations. Despite achieving real-time processing (30 FPS) for high-definition (HD) streams, the system had several limitations. Firstly, the localization system only detects polyps as it is trained solely on the public ASU-Mayo Clinic polyp video database. Moreover, the system only provides approximate locations of disease regions rather than precise boundary delineations. Furthermore, the system only provides approximate locations rather than precise boundaries and achieves a modest 59.5% F1-score using their best method (Darknet-YOLO-EIR).

Xu et al. (2021) proposed a real-time system called ENDOANGEL to detect precancerous stomach conditions, namely gastric atrophy (GA) and intestinal metaplasia (IM). The system achieves 90.1% and 90.8% accuracy for GA and IM using data from the same institute, respectively. However, the system provides only classification, without the ability to detect lesion boundaries. Also, the study only focuses on gastric precancerous lesions, which is limited in disease diversity.

Chang et al. (2024) constructed an ESFPNet focusing on lesion detection and segmentation on endoscopic video. It is a Transformer-CNN-based architecture, with an encoder using mixed Transformer (MiT) derived from SegFormer and an efficient stage-wise feature pyramid (ESFP) decoder combining linear projections and convolutional fusion. This study targets endoscopic video segmentation in the lung and achieves a precision of 0.862 and a recall of 0.823 on the autofluorescence bronchoscopy (AFB) dataset. This model is also effective for small lesions, as 0.3% of the frame area. However, the dataset only consists of a single type of lesion. Additionally, this study stressed the lung more than the GI diseases.

A summary regarding the above-listed models for Chang et al. and Xu et al. is shown in Table 1.

Table 1: Comparisons between relative projects

Category	Chang et al. (2024)	Xu et al. (2021)
Characteristics	•Lesion segmentation in endoscopic video •Transformer-CNN hybrid architecture: MiT encoder and ESFP decoder with linear projections & convolutional fusion	•Precancerous stomach conditions classification •Deep convolutional neural network using ResNet-50, VGG-16, DenseNet-169, and EfficientNet-B4 architectures

Advantages	• Real-time processing • 0.862 precision, 0.823 recall on AFB dataset • Effective for small lesions as small as 0.3% of total frame area	• 90.1% accuracy for GA, 90.8% accuracy testing on data from the same institute
Limitations	• Dataset limitations: only 685 frames and single lesion type •Lacks classification/generalization ability • Focuses on lung, not GI diseases	• No lesion localization, classification only • Small sample size (5 cases in **77** patients) •Excludes 63% frames during preprocessing

To overcome the drawbacks of Transformer-CNN and deep-CNN for segmentation, a model called Segment Anything Model (SAM) was proposed by (Kirillov et al., 2023), so this study considers studying SAM and its variants.

2.4 REVIEW OF SAM MODEL

Introduced by Meta AI, SAM leverages a Transformer-based architecture trained on millions of images and masks to enable interactive segmentation via prompts. While its zero-shot capability initially shows promise for medical applications, SAM underperforms in tasks requiring precise boundary delineation for irregular or low-contrast lesions. For instance, in breast tumor segmentation, SAM achieved lower Dice scores than U-Net, particularly for malignant tumors with ambiguous margins (Ahmadi et al., 2023). This limitation is that SAM is trained on natural images, which lack anatomical complexity and modality-specific features (e.g., speckle noise in ultrasound) inherent to medical data.

To address the shortcomings of SAM, researchers (Wzh, n.d.) proposed SAM-UNet, integrating a parallel CNN branch into the Vision Transformer (ViT) encoder of SAM to enhance local feature extraction while preserving the zero-shot capabilities of SAM. Trained on SA-Med2D-16M—the largest 2D medical segmentation dataset—SAM-UNet achieves state-of-the-art performance (Dice: 0.883) on common modalities like CT and MRI. However, its zero-shot performance on underrepresented modalities (e.g., microscopy, X-ray) remains suboptimal, revealing persistent gaps in generalizability.

The above integration of SAM-UNet utilizes the global context of the Transformer and the spatial precision of CNN. However, there are several issues with the model. Firstly, the model requires large computational resources due to the complex dual-branch design (Martins et al., 2023). Secondly, the model is pre-trained on a natural dataset, indicating its lack of domain-specific abilities for endoscopic videos, especially for detecting GI lesions. Third, the SAM-UNet can carry out segmentation only; it cannot handle lesion classification.

Inspired by the SAM and its impressive performance, this study combines SAM and a classification head to form a new model and applies it to a more specific medical area. Based on this integration, the classification ability is added, making the model capable of both segmenting the lesion area and producing the lesion type. The following chapter mainly discusses the design of the new model, data pre-processing, and loss functions. Chapter 4 consists of ablation studies of different parameters and their effects on the training result, and Chapter 5 draws the conclusion of this project.

2.5 CLINICAL MOTIVATION OF THIS STUDY AND PROJECT OBJECTIVES

Overall, deep learning (DL) has been applied to GI disease analysis (Fan et al., 2023; He et al., 2024). For example, the UNet model was employed to grade gastroesophageal reflux disease (GERD) severity, achieving a 42.64% intersection-over-union (IoU) score (Tran et al., 2022). However, this study was limited to a dataset of 795 static images. Similarly, a hybrid Transformer-CNN architecture (Tang et al., 2023) achieved 67.40% IoU in segmentation and 96.94% classification accuracy for GI diseases but utilized only 1,645 static images, neglecting the dynamic nature of endoscopic videos.

These limitations are critical in clinical practice because upper GI diseases should be diagnosed and the lesion segmented with sequential endoscopy video frames instead of isolated images. Static-image approaches neglect temporal dependencies between frames, hindering lesion evolution tracking and ambiguity resolution in noisy frames. Inspired by the state-of-the-art segmentation ability of SAM, it is utilized in this study. However, the ability of SAM is limited to segmentation; it cannot perform the classification task directly (Liu et al., 2024). To address the above issues, the objective of the study is to propose a video segmentation and classification framework, which processes continuous endoscopy videos for real-time spatiotemporal lesion segmentation and classification based on the integration of SAM and a classification head. This new model is called SAMClass. By incorporating temporal context, this framework advances AI-driven GI disease detection, enabling clinicians to utilize temporally aware diagnostic tools (Sharma et al., 2023). As mentioned before, video segmentation is the main part of this study, while disease classification is the cherry on the cake, so the second objective of the study is to compare the proposed SAMClass with the classical segmentation model, UNet. UNet is also widely used as a baseline model for comparison and is suitable for modifying to video segmentation tasks.

CHAPTER 3: METHODOLOGY

This chapter discusses the methodology for developing a novel lesion detection and classification model named SAMClass. There are 7 sections in the chapter. The first two sections focus on system architecture, followed by the third and fourth sections emphasizing the data pre-processing. Section 3.5 presents the proposed model and UNet model employed for comparison, and the loss functions for this study are also presented in the last section of this chapter.

3.1 SYSTEM ARCHITECTURE OF PROPOSED LESION DETECTION FRAMEWORK

The proposed system integrates two parts: a data processing module and a DL model, which is either a UNet or SAMClass. During training, input videos are processed by MedicalDataset to establish frame-to-mask correspondence. MedicalDataset is a custom dataset class employed in the model training stage. It consists of input video reading, mask binarization, data transformation, and class-wise organization. After data preprocessing, the video frames are passed to the DL model. The SAMClass architecture incorporates an auxiliary classifier to identify specific disease types (e.g., polyps, GERD). Once trained, this SAM-based system achieves dual functionality: segmenting lesion regions and predicting lesion types. In contrast, the UNet architecture prioritizes lightweight training and minimal architectural complexity to ensure compatibility with medical devices constrained by limited GPU resources. Consequently, the UNet excludes classification modules to optimize efficiency.

A validation framework evaluates trained models by processing medical videos and generating lesion detection outputs. Post-training validation confirms the capability of UNet for binary segmentation (lesion vs. background), while the SAMClass model achieves binary segmentation with simultaneous disease-type classification.

3.2 SOFTWARE DEVELOPMENT

This project employs Python to develop the proposed system. PyTorch machine learning library is also adopted due to its flexibility and computational efficiency (Lu & Liu, 2024).

3.3 DATA SELECTION AND LABELLING

To develop the system, training and testing videos are necessary. The data used in this project consists of videos from an open-sourced GI endoscopy dataset named ITI-GERD (Openmedlab, n.d.; Quandhiti, n.d.), and our own videos obtained from the Kiang Wu Hospital, Macao.

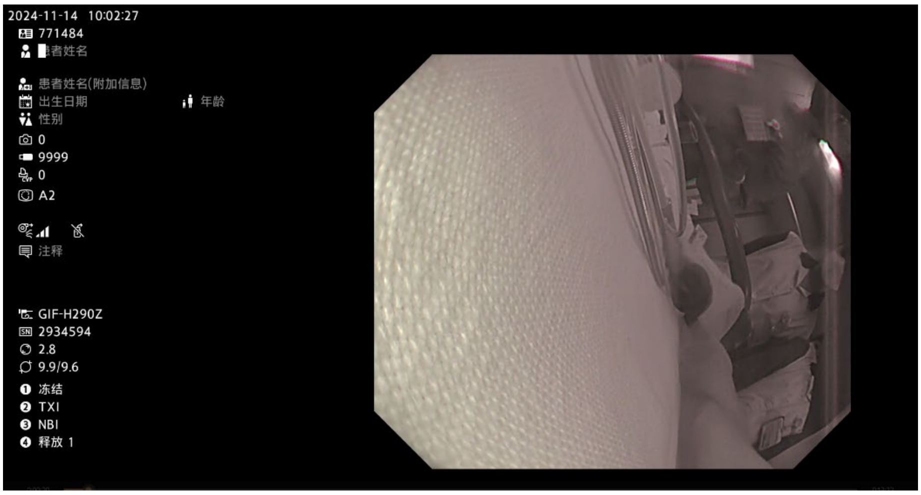
3.3.1 SELF-COLLECTED VIDEOS

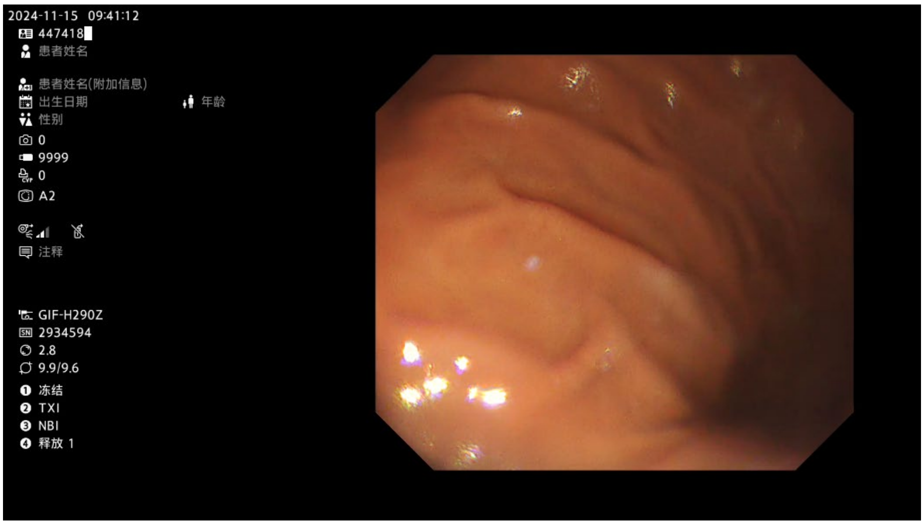
A total of 94 clinical endoscopy videos were retrospectively acquired from routine examinations at Kiang Wu Hospital, Macao. The study dataset was collected following ethical guidelines established in the Declaration of Helsinki, with formal approval from the Medical Ethics Committee of Kiang Wu Hospital in Macao (approval number 2019-005). Due to the retrospective design of the research, the ethics committees granted a waiver for written informed consent requirements.

The videos undergo structured preprocessing to ensure compatibility with downstream model training. Hospital-acquired endoscopy videos typically exhibit frame rates between 30–40 fps (Liu et al., 2014). To balance temporal continuity and annotation efficiency, frames are extracted at a rate of 1 frame per 30 frames from the original videos using the software Turnimage (Redfish-Github, n.d.), to generate sequential PNG files. Extracted frames are manually annotated to delineate disease regions (e.g., polyps, GERD lesions) using Anylabeling (Vietanhdev, n.d.), which produces JSON files storing lesion mask coordinates and other information. A Python script subsequently converts these JSON annotations into TIFF-format masks and transforms PNG frames into lossless TIFF files to preserve diagnostic image quality (Guarneri et al., 2008).

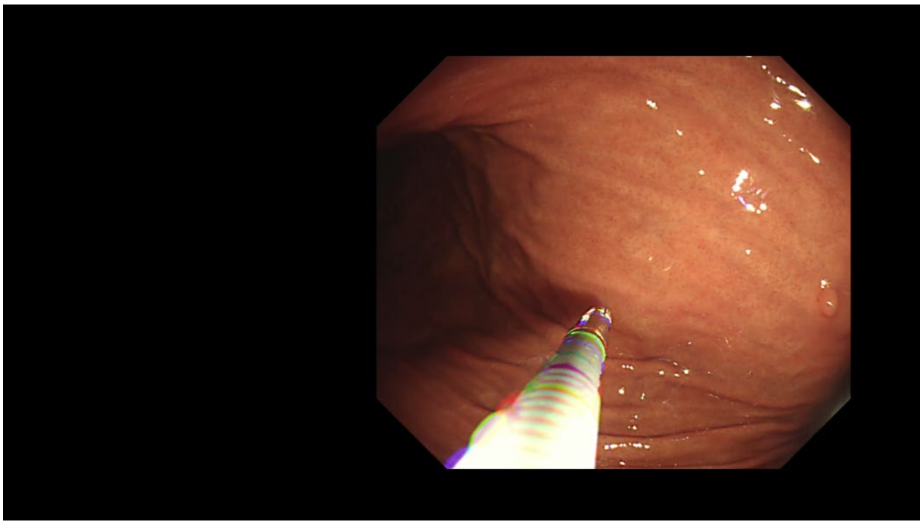
To further streamline the data for model training, a separate Python script is used to reconstruct synchronized video sequences by stacking chronologically numbered TIFF frames and their corresponding masks. The standardization of filenames ensures sequential alignment. In other words, frames and masks are assigned with identical integer names in temporal order. The script incorporates integrity checks to verify equal frame-mask pairs and filename consistency, minimizing mismatches during reconstruction.

Medical videos often contain non-diagnostic overlays (e.g., timestamps, device parameters) occupying the left third of the frames. Within the extracted frames, there is some text on the left original portion of the video. To isolate the endoscopic region of interest (ROI) and reduce segmentation noise, a custom Python script (textBlock.py) masks these extraneous regions, ensuring the model focuses solely on diagnostically relevant regions. Examples of the original frame and the preprocessed video frames are shown in Figure 2.





(a) Examples of texts on videos obtained from hospitals



(b) Example of processed frames without text

Figure 2: Examples of original frame with text and the processed frame without text.

3.3.2 OPEN-SOURCED VIDEOS

The ITI-GERD dataset, an open-source collection of medical images focused on gastroesophageal reflux disease (GERD), contains 688 upper GI endoscopy images with corresponding lesion masks (Kohli et al., 2017). However, preprocessing is required to reconstruct sequential video frames by sorting the unordered images of the dataset. A Python script (orderFinds.py) addresses this by identifying the temporal continuity among frames. The workflow of the Python script is displayed in Figure 3.

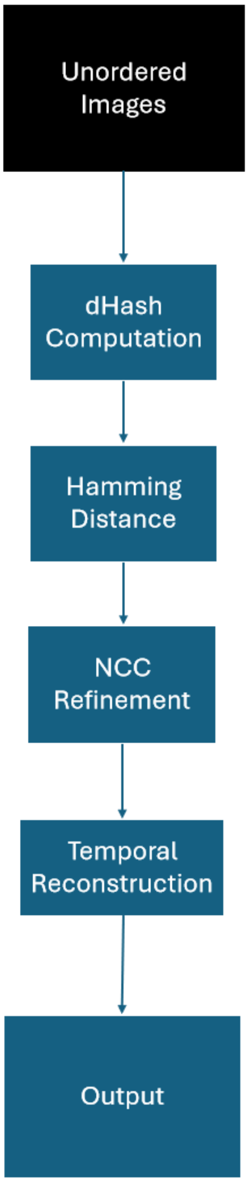


Figure 3: Workflow of orderFinds.py for continuous frame reconstruction.

The algorithm first computes perceptual hashes (dHash) for all images, which encodes image features into 64-bit representations for efficient similarity comparisons (Yang et al., 2022). Each image is resized to 9×8 pixels, converted to grayscale, and analyzed via sliding-window comparisons of adjacent pixel brightness. The formula is shown in Eq. (1).

$$H_i = \begin{cases} 1 & if\ I(x,y) > I\ (x+1,y) \\ 0 & Otherwise \end{cases} \tag{1}$$

where $i = 8y + x$, and both $x, y \in [0,7]$. After comparison and computation, a 64-bit hash table is generated.

The resulting hash values enable Hamming distance calculation to quantify pairwise image similarities. Images with a Hamming distance below a threshold (α=5) are clustered as sequential frames (Lan et al., 2018; Huo et al., 2010). The mathematical expression for Hamming distance is shown in Eq. (2).

$$D(H^A, H^B) = \sum_{i=0}^{63} \left| H_i^A - H_i^B \right| \tag{2}$$

where H_i^A and H_i^B represent the dHash value of i^{th} bit in images A and B, respectively. To further refine the ordering, the normalized cross-correlation (NCC) is used to evaluate pixel intensity patterns between resized (256×256) image pairs. NCC values range from -1 (inverted patterns) to 1 (identical patterns), with higher values indicating stronger frame continuity (Li, 2017). This hierarchical approach combines dHash clustering and NCC ranking to reconstruct temporally coherent video sequences. Renamed frames and masks are then compiled into videos for downstream tasks. The NCC is calculated by Eq. (3).

$$NCC(I_1, I_2) = \sum \frac{1}{\sigma_1 \sigma_2} (I_1(x, y) - \mu_1)(I_2(x, y) - \mu_2) \quad (3)$$

where I_1 and I_2 are two greyscale images, σ_1 and σ_2 are the standard deviation of I_1 and I_2; μ_1 and μ_2 are the means of I_1 and I_2.

After performing all the above steps, the continuous frames can be constructed by starting with a random frame and then computing and ranking the NCC values from highest to lowest. After renaming the corresponding frame and mask files in the founded order, the medical video can be reconstructed, making the dataset usable for later classification and segmentation.

The final dataset combines the preprocessed ITI-GERD images with the self-collected clinical videos, totaling 1,758 frames partitioned into a 9:1 training-validation split. For the SAMClass model, frames are categorized into gastric Healthy, upper GI gastric polyp, and GERD classes. The detailed dataset composition is provided in Table 2.

Table 2: Data distribution of video frames of the final dataset

	upper GI Healthy	Gastric Polyp	GERD	Total

Training	66	829	688	1582
Test	7	92	76	176
Total	73	921	764	1758

3.4 DATA PREPROCESSING AND AUGMENTATION

3.4.1 DATA AUGMENTATION

As the number of training data is also limited, data augmentation is required. Data augmentation enhances model generalization by diversifying training data through geometric and optical transformations (Oh et al., 2020). This study employs the Albumentations Python library (Buslaev et al., 2020) to apply augmentation techniques to simulate endoscopic environments.

For both models, basic geometry and optical transformations are applied to generate more data. Elastic transformation and grid distortion are employed to better simulate the deformation of organs and the moving environment of endoscopy, as proven by (Castro-Pareja et al, 2006) and (Jiajun et al, 2023).

In addition, spatial augmentations are applied, which include resizing (to 256×256), random horizontal flipping (50% probability), 30-degree rotation, and elastic deformations (σ=15, α=30) to mimic tissue flexibility during endoscopic motion (Castro-Pareja et al., 2006b; Jiajun et al., 2023). To simulate clinical occlusions, CoarseDropout masks (8×32×32 regions) and reflux-like artifacts (fill value=180) are introduced. Finally, brightness or contrast adjustments (±0.2) and Contrast Limited Adaptive Histogram Equalization (CLAHE) are utilized to optimize feature visibility. Afterwards, augmented images are standardized as PyTorch tensors for model input.

3.4.2 DATASET CREATION AND SPLITTING

The MedicalDataset class, which is applied uniformly across both models (see Appendices VI and IX), processes video data by organizing frame-mask pairs and standardizing inputs for training. In the UNet framework, this class handles the unlabeled frame-mask sequences without disease-type annotations. It systematically maps temporal correspondences between frames and masks across video files. During data loading, the __getitem__ method extracts paired frames and masks, which undergo sequential preprocessing as follows: grayscale masks are first binarized using a threshold of 127 (foreground: pixel intensity ≥127; background: <127), while frames and masks are normalized to the mean (0.485, 0.456, 0.406) and standard deviation (0.229, 0.224, 0.225) of ImageNet (ImageNet, n.d.). After being augmented via the pipelines described in Section 3.4.1, the processed data is returned as paired tensors—normalized RGB images and single-channel binary masks—ready for model training.

3.5 MODELS

3.5.1 PROPOSED MODEL

Inspired by the strong segmentation ability of SAM, this study employs SAM to perform binary segmentation tasks for endoscopy medical videos. To accomplish the classification task, this study originally adapted a classification head on the decoder of SAM to form the SAMClass model. The SAMClass model combines the Segment Anything Model (SAM) framework with a classification head to enable simultaneous lesion segmentation and disease-type classification. The overall structure of the proposed SAMClass is illustrated in Figure 4.

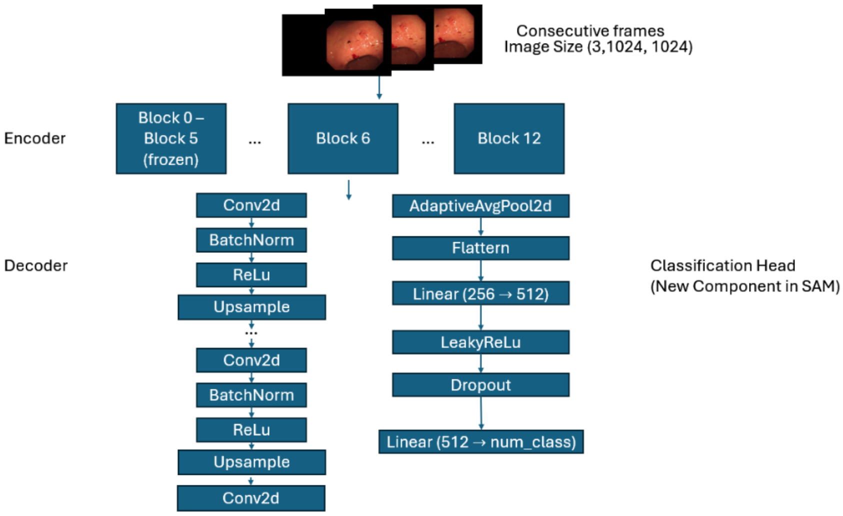


Figure 4: Overall structure of proposed SAMClass model.

As shown in Figure 4, the SAMClass model includes an encoder, a decoder with the segmentation head, and a classification head. The encoder employs a pre-trained Vision

Transformer (ViT)-B architecture (Sharma et al., 2024), which contains 12 Transformer blocks with a patch size of 16×16. Layers from Block 0 to Block 5 are frozen to mitigate overfitting risks on small-to-medium datasets while retaining pre-trained low-level feature extractors, as demonstrated in prior work (Cuccu et al., 2020).

The SAMClass decoder comprises four sequential convolutional blocks. Each block contains a 2D convolutional layer, a batch normalization layer, a ReLU activation layer, and an upsampling layer. The decoder processes an input image map of dimensions (256, 64, 64) from the encoder, progressively reducing output channels from 256 to 32. The upsampling sub-layer with a scale factor of 2 doubles the image map resolution at each step. The final decoder layer is a 2D convolution with 32 input channels and 1 output channel. After processing through the decoder, the output is a segmented image map of dimensions (1, H, W), where H and W denote the original height and width of the input image. This allows precise distinction between lesion and background regions.

The traditional SAM model (Kirillov et al., 2023) employs a transformer-based decoder and cross-attention mechanism to embed prompts and images. However, this reliance leads to high computational demands. Furthermore, since SAM is trained on 2D images, segmenting 3D videos requires dividing the video into patches to achieve pseudo-3D segmentation (Chen et al., 2024). This approach ignores inter-frame relationships, resulting in spatial information loss and reduced segmentation accuracy in medical videos (Chen et al., 2024). To address these limitations, SAMClass introduces a custom encoder that replaces the transformer architecture with sequential convolutional blocks. Through four convolutional operations, spatial resolution is progressively restored from 64×64 to 1024×1024. This design minimizes computational resource utilization while preserving pixel-level precision in medical video segmentation.

The classification head performs lesion-type classification through sequential operations. Input frame images first pass through a 1×1 adaptive average pooling layer to aggregate spatial information, followed by flattening into a 1D feature vector. This method ensures compatibility with variable-sized inputs while emphasizing texture features over exact spatial relationships (Han et al., 2019; Cao et al., 2022; Hu et al., 2019). The resulting 256-dimensional feature vector is fed into a fully connected network with a 512-node hidden layer (LeakyReLU activation and 0.6 dropout for regularization) to generate the final classification output. Spatial attention mechanisms are excluded due to their increased training complexity (Xuanhao & Min, 2023).

3.5.2 MODEL FOR COMPARISON AND ITS SELECTION RATIONALE

The model selected for comparison is the UNet model. UNet excels in localized lesion segmentation, particularly for small datasets (<1,000 images) or low-contrast anatomical regions. It remains as the gold standard for medical image segmentation and is widely adopted as the baseline model in medical imaging research. (Sengar et al., 2022) Other models, however, are rarely utilized in clinical workflows due to computational complexity and poor performance on medium, imbalanced medical datasets, such as the Mask R-CNN or the DeepLabV3+ model. (Sahu et al., 2022; Nguyen & Huynh-The, 2024)

Therefore, the UNet model is selected as the baseline model and compared with the proposed model for segmentation ability. Visual comparisons are also presented in Chapter 4 between the proposed SAMClass model and the UNet model.

The UNet architecture (Milesial, n.d.) serves as a non-classifying lesion detection framework, processing paired frame-mask videos as inputs. The structure of this model is depicted in Figure 5.

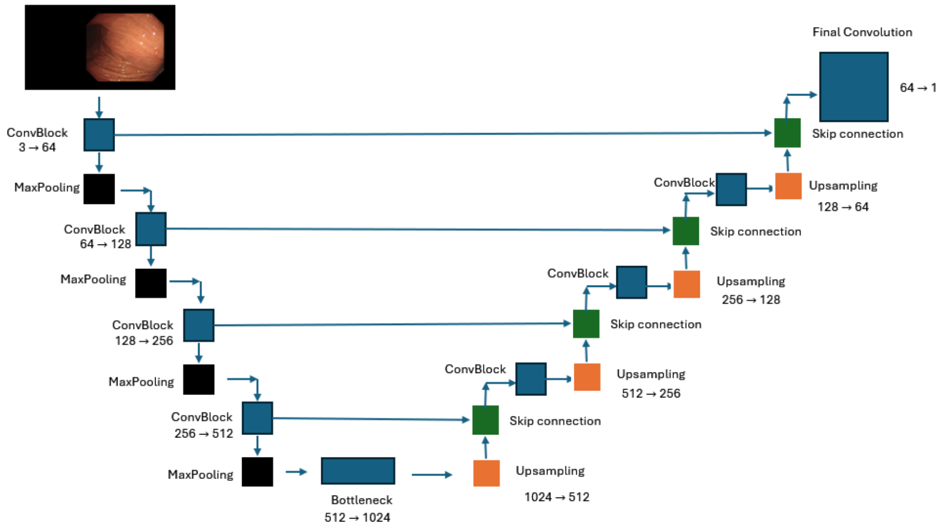


Figure 5: Overall structure of UNet model.

The model begins with a ConvBlock module comprising two sequential 3×3 convolutional layers (padding=1), each followed by 2D batch normalization and ReLU activation. Four encoder layers progressively downsample input features from 64 to 512 channels. Each encoder sublayer integrates a 2D convolutional layer and ReLU activation. At the bottleneck of the network, a max pooling layer (kernel size=2) reduces spatial resolution by retaining the maximum pixel value within 2×2 windows to enhance spatial feature extraction (Gao et al., 2021). The bottleneck expands channel dimensions to 1024 to capture high-level abstractions. Four decoder layers are then employed to reconstruct spatial resolution via transposed convolutions (upsampling) and ConvBlock operations. The decoder pathway progressively reduces feature channel dimensions through four sequential stages: from 1024 to 512, 512 to 256, 256 to 128, and finally 128 to 64 channels. Skip connections bridge encoder and decoder layers, which concatenate feature maps along the channel axis to recover spatial details lost during downsampling (Feng et al., 2024). The final layer employs a 2D convolutional operation (input channels=64, output channels=1) to generate a single-channel segmentation mask. This design enables binary segmentation between background and lesion regions.

3.6 LOSS FUNCTION

The loss function is to quantify the difference between the prediction of machine learning models and ground truth (Tang & Fan, 2022). By minimizing this difference, the model prediction can be improved. Multiple loss functions are utilized in this study and are presented in the following subsections.

3.6.1 DICE LOSS

Dice loss addresses the class imbalance between large background areas and small target regions in segmentation tasks. Initially proposed by (Wang et al., 2020) and (Zhao et al., 2020), its effectiveness in improving segmentation accuracy was further validated by (Shirokikh et al., 2020; Zhang et al., 2021).

Dice loss quantifies the dissimilarity between two sets and is defined by Eq. (4).

$$Dice\ Loss\ =\ 1-2\times\frac{|M\cap P|}{|M|\cup|P|} \tag{4}$$

here, M represents the ground truth area, and P denotes the predicted lesion area. Minimizing this loss encourages the prediction of the model to align closely with the ground truth.

In the SAMClass training, an edge-aware weighting function is integrated to enhance noise robustness (Zhang et al., 2022). This function calculates the pixel value difference between the max-pooled and average-pooled versions of a mask image. It is computed by looking for the pixel value difference between the max pooling and average pooling of a mask image. Weight is then created to amplify the loss at edge pixels and applied to the dice loss. The edge weight can be calculated by binarization as shown in Eq. (5).

$$w = \begin{cases} 1, & if\ e > 0.1 \\ 0, & otherwise \end{cases} \tag{5}$$

If the edge value e is larger than 0.1, then the edge-weighted loss is 3, otherwise 0. e is computed by Eq. (6).

$$e = MaxPooling(mask) - AvgPooling(mask) \tag{6}$$

Subsequently, the edge weight w is applied to calculate the dice loss, and the edge-weighted target set $M_w$for all masks is computed via Eq. (7).

$$M_w = \sum wM \tag{7}$$

Meanwhile, the edge-weighted predicted set P_w is calculated by using Eq. (8), where w is the equivalent edge weight for calculating M_w.

$$P_w = \sum wP \tag{8}$$

Hence, the resultant dice loss is shown in Eq. (9).

$$Dice\ Loss\ =\ 1 - 2 \times \frac{|M_w \cap P_w|}{|M_w| \cup |P_w|} \tag{9}$$

3.6.2 FOCAL LOSS

An improved form of focal loss is derived from binary cross entropy loss (BCE) (Chen, 2022). The BCE loss is used to measure the differences between predicted possibility p after activation function and the ground truth labels ($g \in \{0,1\}$). It can be calculated by Eq. (10).

$$BCE(P, g)\ =\ -\left[g \cdot \log(p) + (1 - g) \cdot \log(1 - p)\right] \tag{10}$$

where P denotes the predicted possibilities after applying activation for all samples, p is the predicted probability for a single sample, and g is the ground truth label for single sample.

By changing the sample weights, the model is forced to focus more on difficult samples than simple ones (Zhao & Liu, 2022). Using focal loss can increase the accuracy of segmenting classes with fewer samples and also reduce the risk of overfitting. (Hossain et al., 2021)

Focal loss (FL) can be calculated by weighting probabilities using Eq. (11).

$$FL = -\alpha(1-p_t)^{\gamma}\log p_t \tag{11}$$

where p_t is the predicted probability for sample belonging to specific class, α is the weighting factor for adjusting the importance of specific class, and γ is the factor controlling the decay of easy samples.

When the predicted possibility p_t is small, the specific type of samples is considered to be hard samples, leading to a large contribution to the overall focal loss. When p_t is a large value, indicating samples are easy to classify, hence its contribution to the focal loss is small due to decay coefficient. Both types of loss function are used in the model training stage.

3.6.3 SEGMENTATION LOSS FOR UNET

The loss function for training the UNet model consists of a dice loss and a focal loss. The total segmentation loss L_{seg} can be computed via Eq. (12).

$$L_{seg} = w_d \times Dice\ Loss + w_f \times FL \tag{12}$$

where w_d and w_f are dice and focal loss coefficients, which are set to be 1 by default.

The dice loss is computed by Eq. (13).

$$DL = 1 - 2 \times \frac{P \cap G + \epsilon}{P \cup G + \epsilon} \tag{13}$$

where P is the predicted lesion area, G is the lesion area in ground truth, and ϵ is a very small number preventing the division of 0 in the denominator.

The focal loss is calculated by Eq. (14).

$$FL = \frac{1}{N} \sum_{i=1}^{N} ((1 - p_t)^{\gamma} BCE(P, G)) \tag{14}$$

where N is total pixel number.

3.6.4 MULTI-TASK LOSS FUNCTION

The multi-task loss function combines and weighs both dice loss and focal loss functions for segmentation and classification. Similar to UNet, the segmentation loss for SAMClass is the addition of the dice loss and the focal loss as shown in Eq. (12). Meanwhile, the classification focal loss function is adopted for the loss function of the classification task.

It is proposed by Singh et al. (2023) that the goal of the classification focal loss function is to balance the unequal sample distribution across several classes. It finds the average focal loss across different classes, and a weighted factor is given to reduce loss contribution from well-classified examples. The classification focal loss (CL) can be calculated by Eq. (15).

$$CL = -\sum_{c=1}^{C} \omega_c (1 - q_c)^{\gamma} \log(q_c) \tag{15}$$

where C is the number of classes, ω_c is the weighting factor for each class, and q_c is the class possibility after Softmax operation. ω_c can be found by Eq. (16).

$$\omega_c = \frac{\frac{1}{\sqrt{n_c + \epsilon}}}{\sum_{k=1}^{C} \frac{1}{\sqrt{n_k + \epsilon}}} \tag{16}$$

where n_c is the number of training samples of class c.

After combining each loss component, the multi-task loss function can be computed by Eq. (17).

$$Multi - task\ Loss = w_d \times Dice\ Loss + w_f \times Focal\ Loss + w_{cls} \times Classfication\ Loss \tag{17}$$

where w_d, w_f and w_{cls} are weights of each loss for the multi-task loss. Initially, the values of these weights are all set to be 1 by default, and the ablation study on effects of using different weights coefficients is presented in Chapter 4.

CHAPTER 4: EXPERIMENT AND RESULTS

4.1 EXPERIMENT

4.1.1 OBJECTIVES AND EXPECTATION

The experiment identifies the optimal UNet and SAMClass models through systematic ablation studies, in which individual parameters (e.g., learning rate, loss weights) are modified sequentially to evaluate their impact on training outcomes. After performing the ablation studies, the refined models with optimal configuration processes inputted medical videos. Finally, segmentation and classification results derived from continuous frames of the same input videos are compared.

4.1.2 TRAINING CONFIGURATIONS AND PARAMETERS

All the models are implemented using PyTorch framework (Pytorch, n.d.) in Python; and training and evaluation are performed on an NVIDIA RTX 4090 GPU with 24 GB memory. To ensure fair comparison across experiments, several training parameters are kept constant throughout all training configurations, which are listed in Table 3.

Table 3: Training parameters that are kept constant throughout all training.

Parameter	Value
Epoch	100

Optimizer	AdamW
Weight decay	1×10^{-4}
Libraries	os cv2 glob torch tqdm albumentations

4.1.3 EXPERIMENTAL DESIGN

The experimental evaluation consists of two major components: ablation studies to optimize model parameters and a comparative analysis between UNet and SAMClass architectures.

Four ablation studies are conducted to determine the optimal training configuration as follows:

1. Learning rate optimization: Models are trained with three different learning rates (1×10^{-3}, 1×10^{-4}, and 1×10^{-5}) to identify the optimal rate for convergence and performance.

2. Scheduler comparison: Two learning rate schedulers including ReduceOnPlateau and CosineAnnealingWarmRestarts are used to train the models.

3. Multi-task loss weighting: For the SAMClass model, different weighting schemes for the segmentation and classification losses are evaluated to determine the optimal combination for overall performance enhancement.

4. Dropout rate: For the SAMClass model, dropout rates from low to high (0.2, 0.4, and 0.6) are utilized to explore the appropriate dropout rate.

Following the ablation studies, a comprehensive evaluation comparing the performance of UNet and SAMClass architecture is conducted. Both models are trained using their respective optimal hyperparameters identified during the ablation phase. To enable direct comparison, both architectures are trained and evaluated at 1024×1024 resolution, because SAMClass is constrained to 1024×1024 due to its pre-trained architecture requirements.

4.1.4 EVALUATION METRICS

Intersection over Union (IoU) is employed for segmentation performance evaluation. It is advantageous due to its insensitivity to lesion size and its direct reflection of how closely model predictions align with manual annotations (Fu et al., 2018). After inferring lesion areas on the validation dataset called ‘post-epoch’, IoU is computed between the predicted mask and ground truth mask by using Eq. (18).

$$IoU = \frac{|A \cap B|}{|A \cup B|} \tag{18}$$

where A and B represent the pixel areas of the predicted mask and ground truth mask, respectively. The higher the IoU, the better the segmentation performance is.

For the SAMClass model, IoU is calculated per class, and the mean IoU (mIoU) reflects overall performance. Mathematically, mIoU is defined by Eq. (19).

$$mIoU = \frac{1}{N} \sum_{i=1}^{N} \frac{|A_c \cap B_c|}{|A_c \cup B_c|} \tag{19}$$

where N is the number of classes, A_i and B_i denote the predicted and ground truth regions for class $\boldsymbol{i}$, $|A_i \cap B_i|$ is their intersection area, and $|A_i \cup B_i|$ is their union area. For healthy video frames without lesions, IoU equals 1 if the prediction is entirely negative (no false positives) and 0 otherwise. Checkpoint files are saved on the basis of best IoU for the UNet model and the best mIoU for the SAMClass model. IoU defines spatial overlap accuracy for individual lesions, making it suitable for clinical evaluation. Besides, mIoU takes the average of IoU across all classes, generalizing this index into multi-class usage scenarios (Chahbar et al., 2025).

For classification, three metrics are employed, which include accuracy, precision and recall. Accuracy, defined as the ratio of correctly classified pixels to total pixels, serves as the evaluation metric for classification tasks (Kirimtat & Krejcar, 2023). It can be calculated by Eq. (20).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{20}$$

where TP is true positive, meaning the correctly predicted positive case, TN is true negative, which is the correctly predicted negative case. FP and FN stand for incorrectly predicted negative as positive, and positive as negative respectively.

Precision (PR), another common classification metric (Jurdi & Colliot, 2023), measures the ratio of true positives to all positive predictions. It can be calculated by Eq. (21).

$$PR = \frac{TP}{TP + FP} \tag{21}$$

Recall is also used as an evaluation metric, and it is defined as the capability of the model of detecting all relevant positive instances (Audeh et al., 2013). It can be calculated by Eq. (22).

$$Recall = \frac{TP}{TP + FN} \tag{22}$$

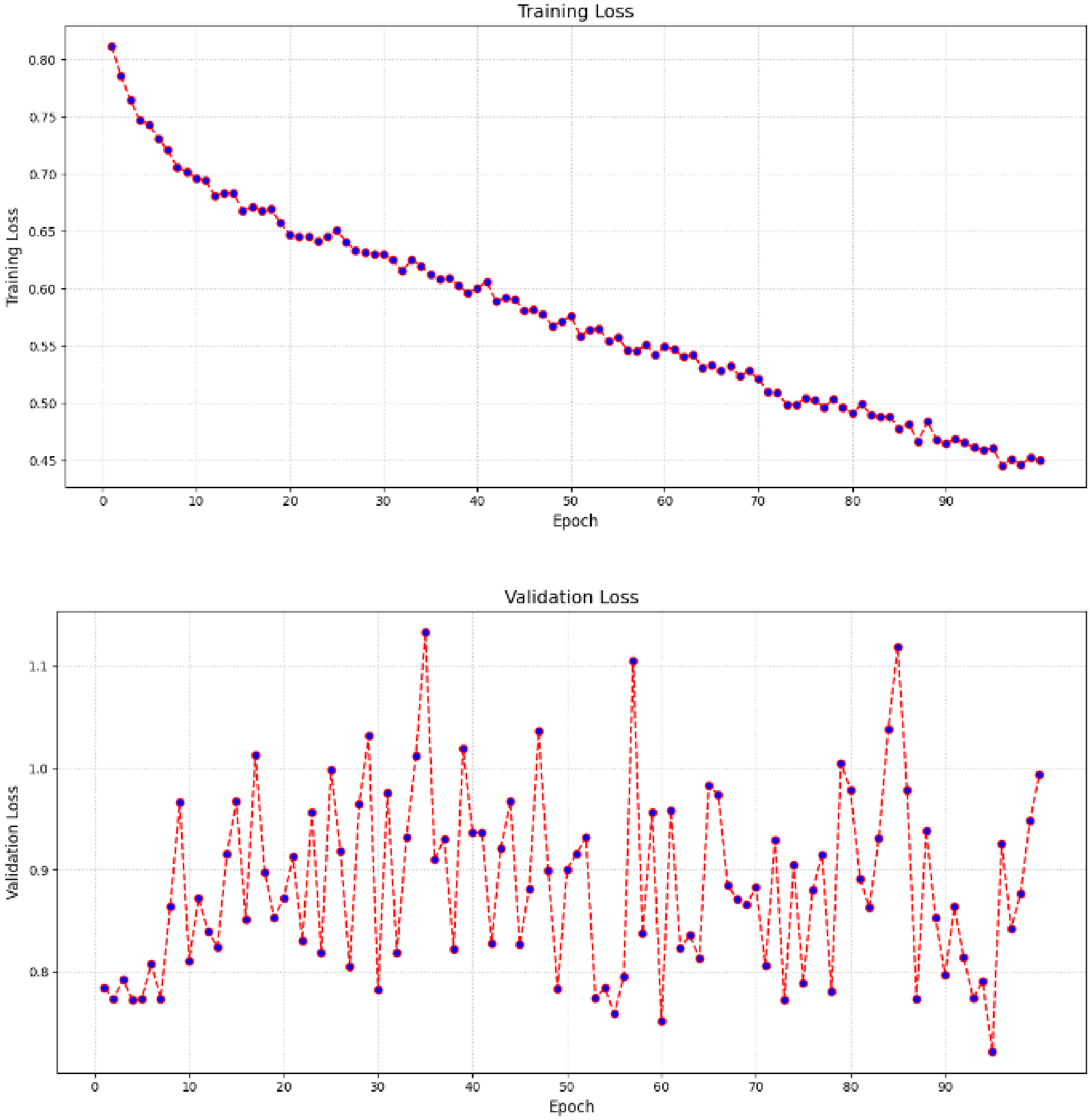
4.2 RESULTS

4.2.1 ABLATION STUDY OF USING DIFFERENT LEARNING RATES

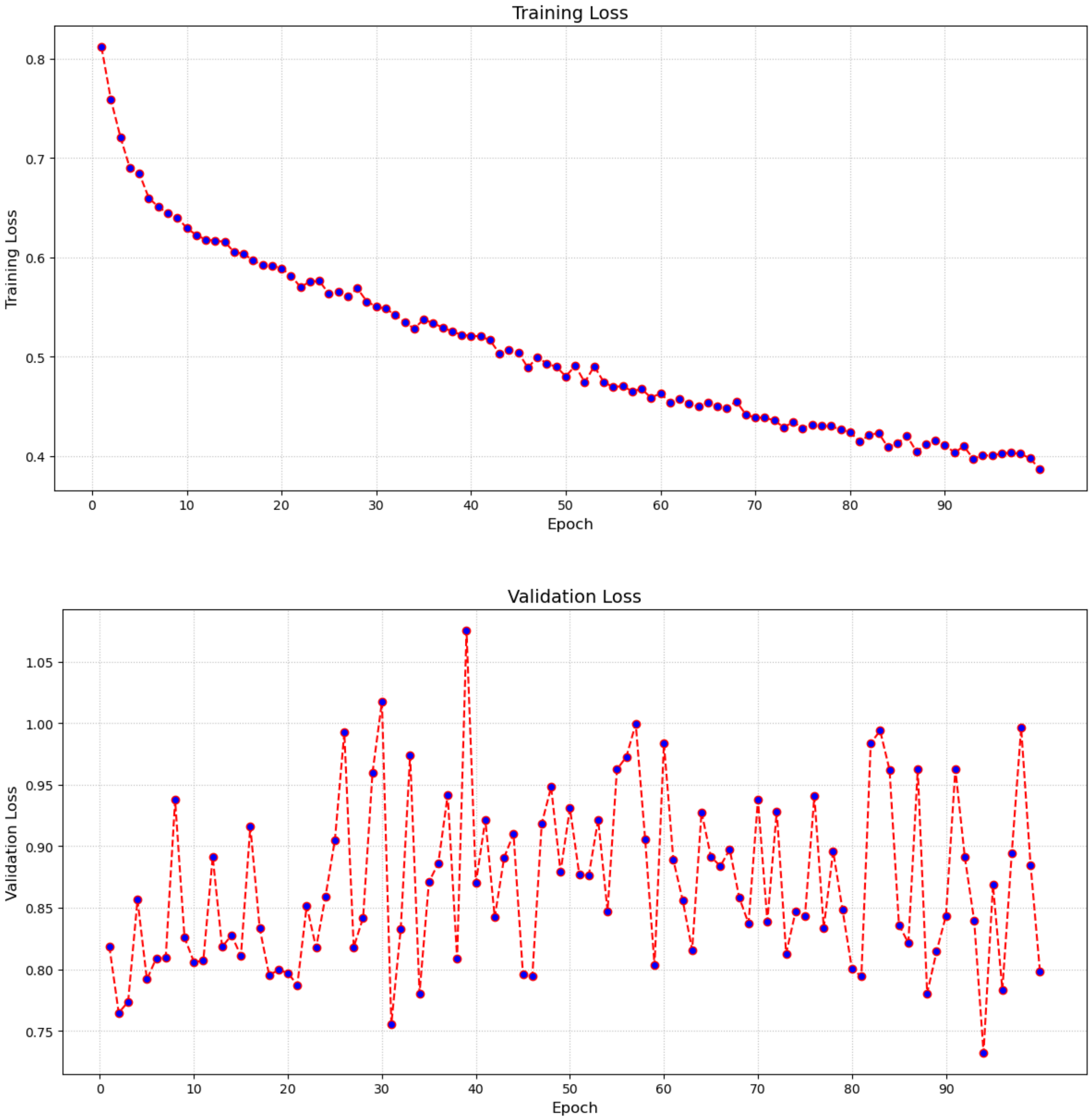
The training results of the effect of different learning rates are displayed in Table 4, and the training and validation losses for the UNet model under the three learning rates are shown in Figure 6.

Table 4: Ablation study on IoU with high and low learning rates

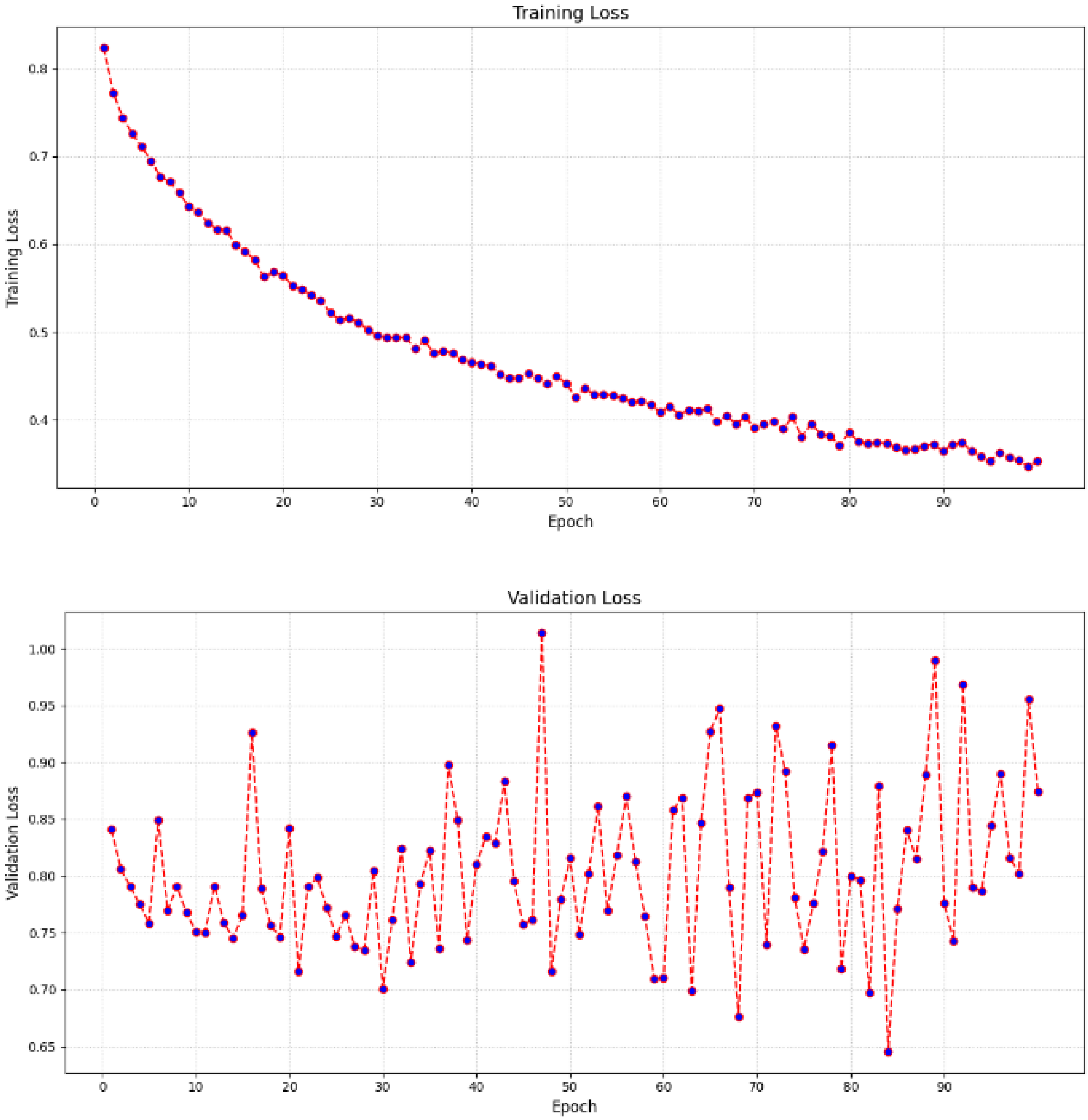
Model	Learning Rate (Ir)	IoU
UNet	1×10^{-3}	0.3738
	1×10^{-4}	0.3878
	1×10^{-5}	0.4122
SAMClass	1×10^{-3}	0.7642
	1×10^{-4}	0.7432
	1×10^{-5}	0.7027



(a) Training and validation losses for Ir = 1×10^{-3}



(b) Training and validation losses for lr $=1 \times 10^{-4}$

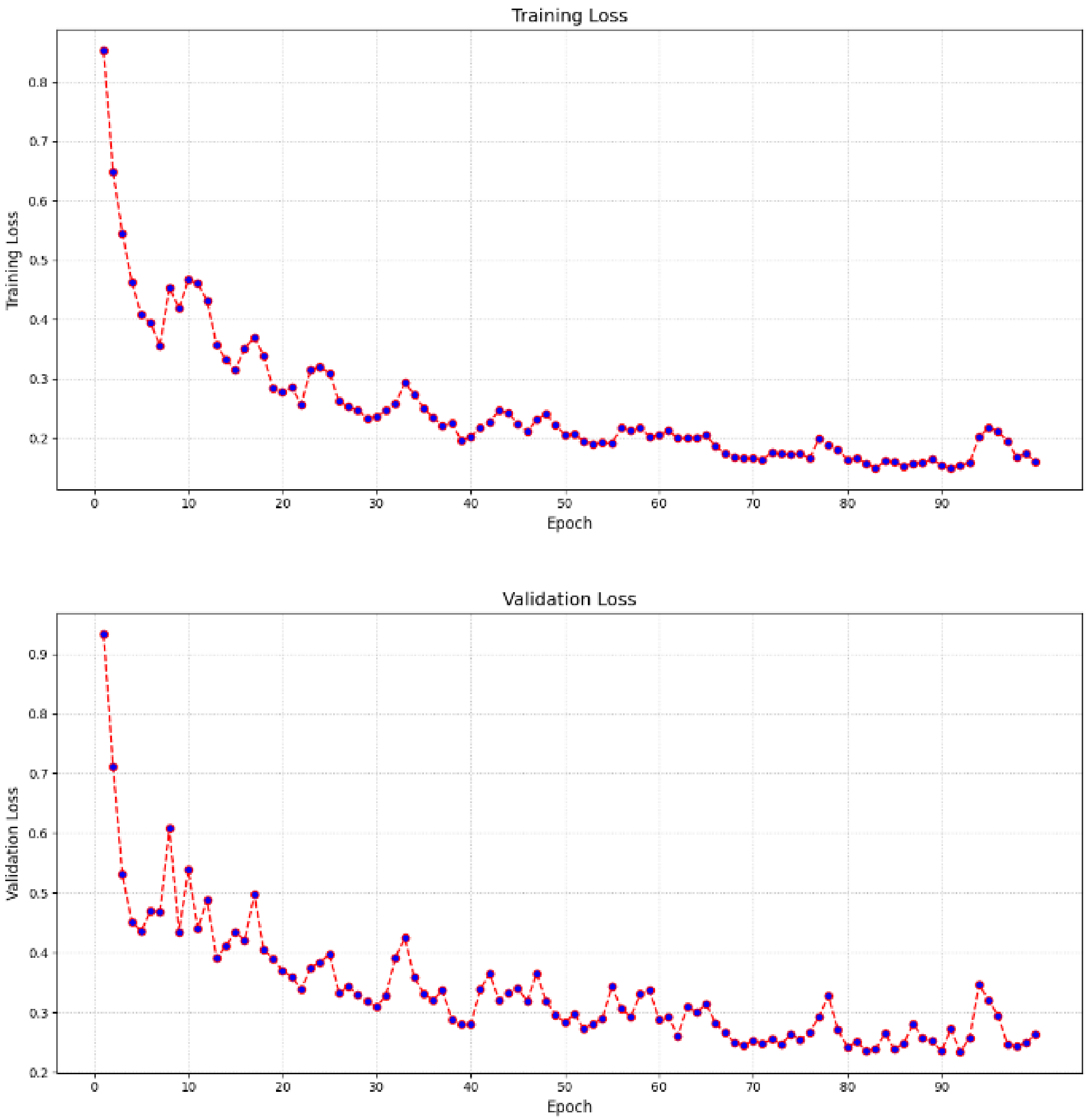


(c) Training and validation losses for Ir = 1×10^{-5}

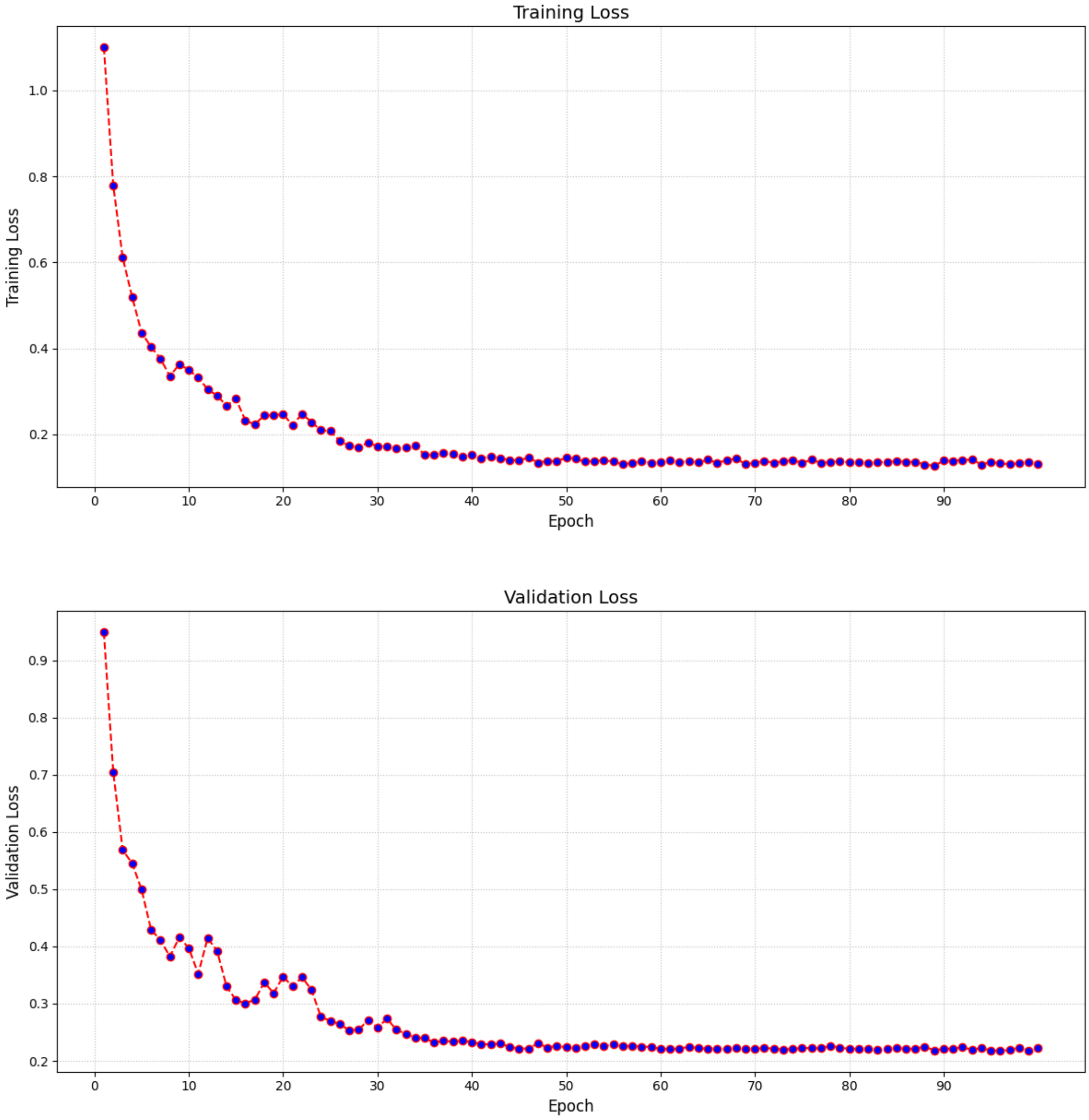
Figure 6: Training and validation losses for UNet model under three learning rates

Figure 6 reveals that the validation loss cannot converge and oscillates although the training loss is decreasing, which implies there is no improvement to the model.

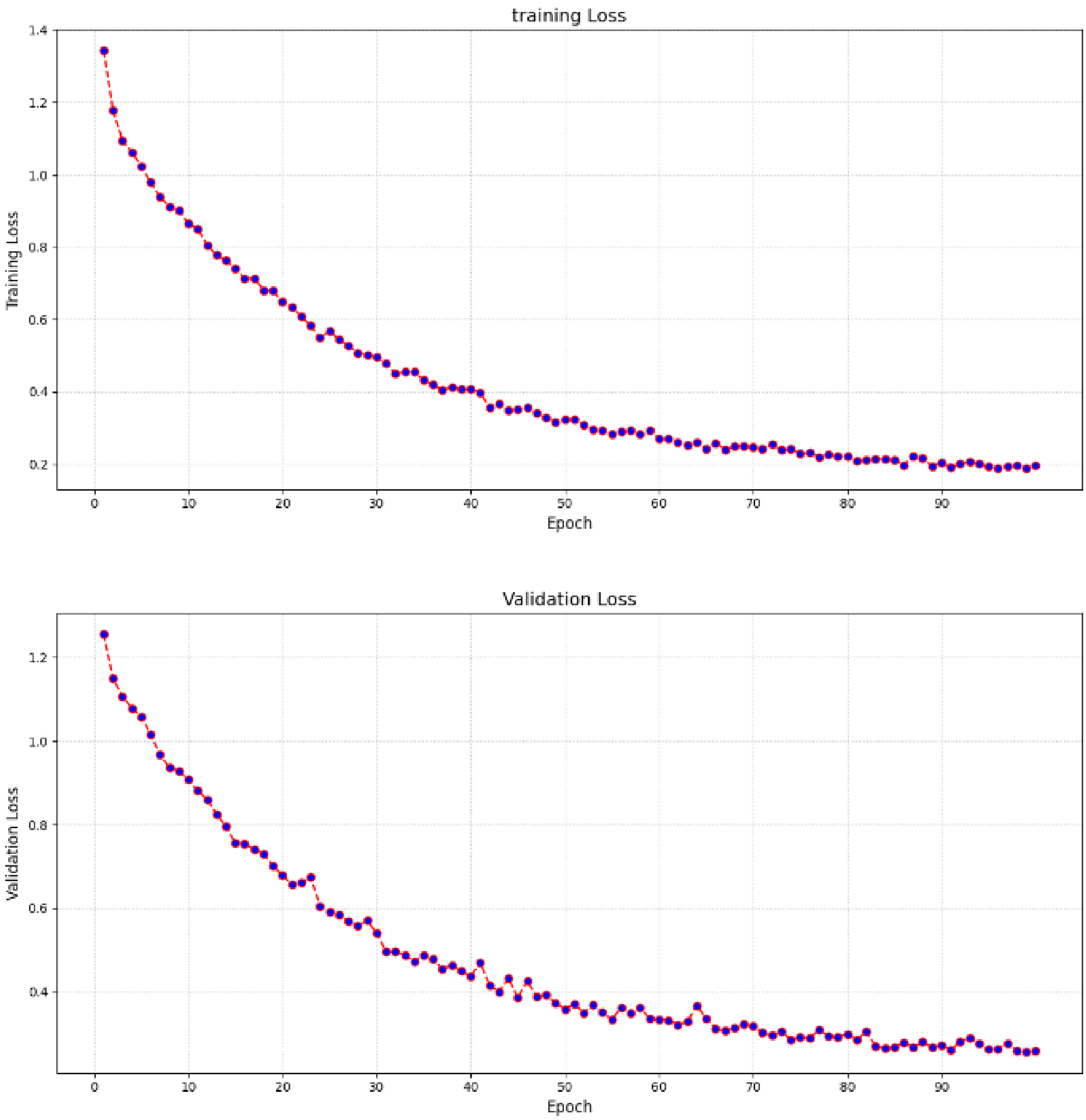
The training and validation loss for the SAMClass model is shown in Figure 7.



(a) Training and validation losses for Ir = 1×10^{-3}



(b) Training and validation losses for Ir $=1 \times 10^{-4}$



(c) Training and validation losses for Ir = 1×10^{-5}

Figure 7: Training and validation losses for SAMClass model under three learning rates

Figure 7 shows that both training and validation losses for SAMClass also decrease. Moreover, the difference between training and validation loss is minor. It means no overfitting occurs during training. (Trivedi et al., 2021)

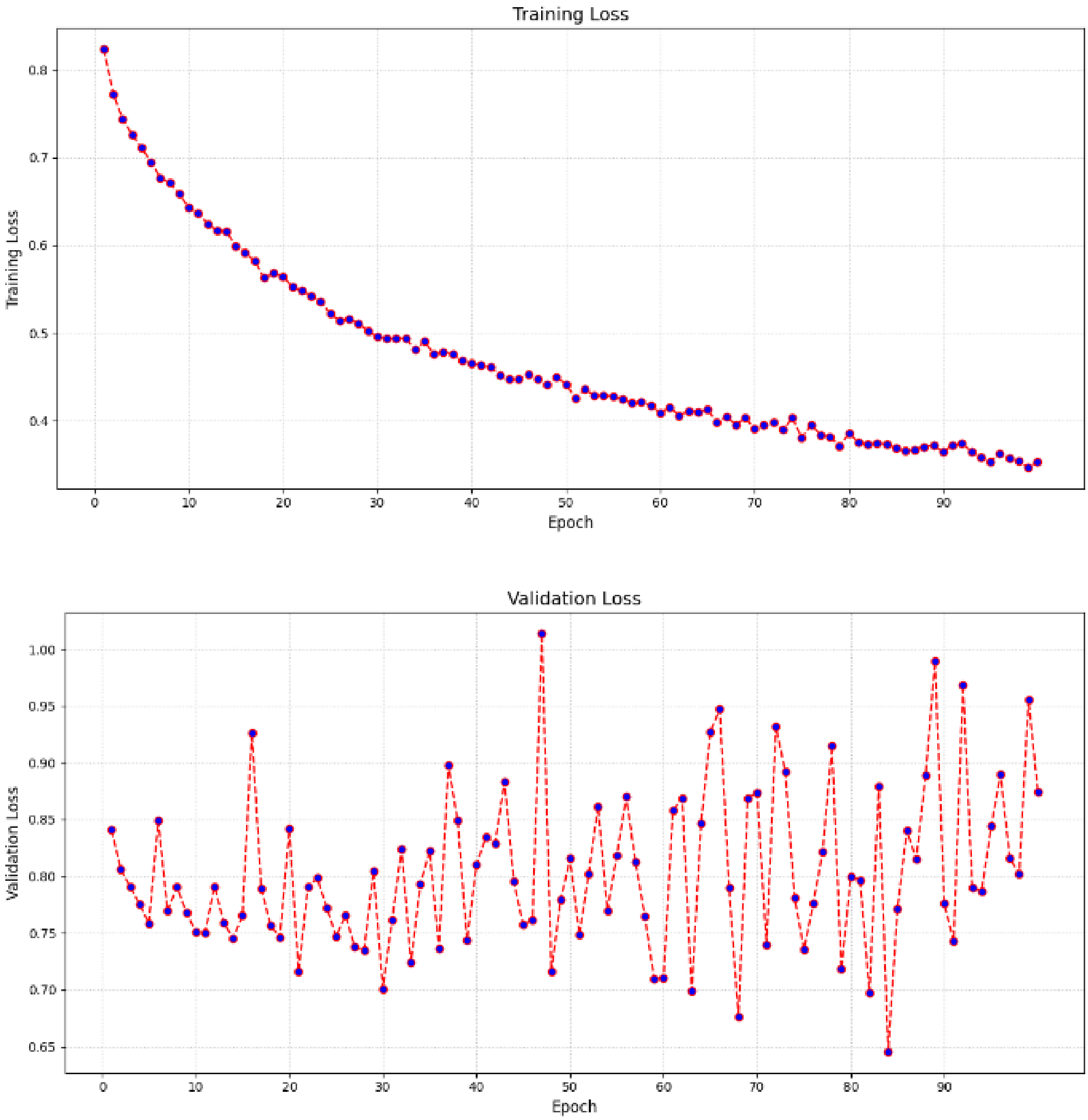
4.2.2 ABLATION STUDY ON USING DIFFERENT SCHEDULERS

Table 5 compares the effects of CosineAnnealingWarmRestarts and ReduceLROnPlateau schedulers under a fixed initial learning rate of 1×10^{-3} and 100 epochs. The learning rate and epoch settings are selected based on results in the last section, where higher initial rates demonstrate superior optimization convergence compared to lower rates.

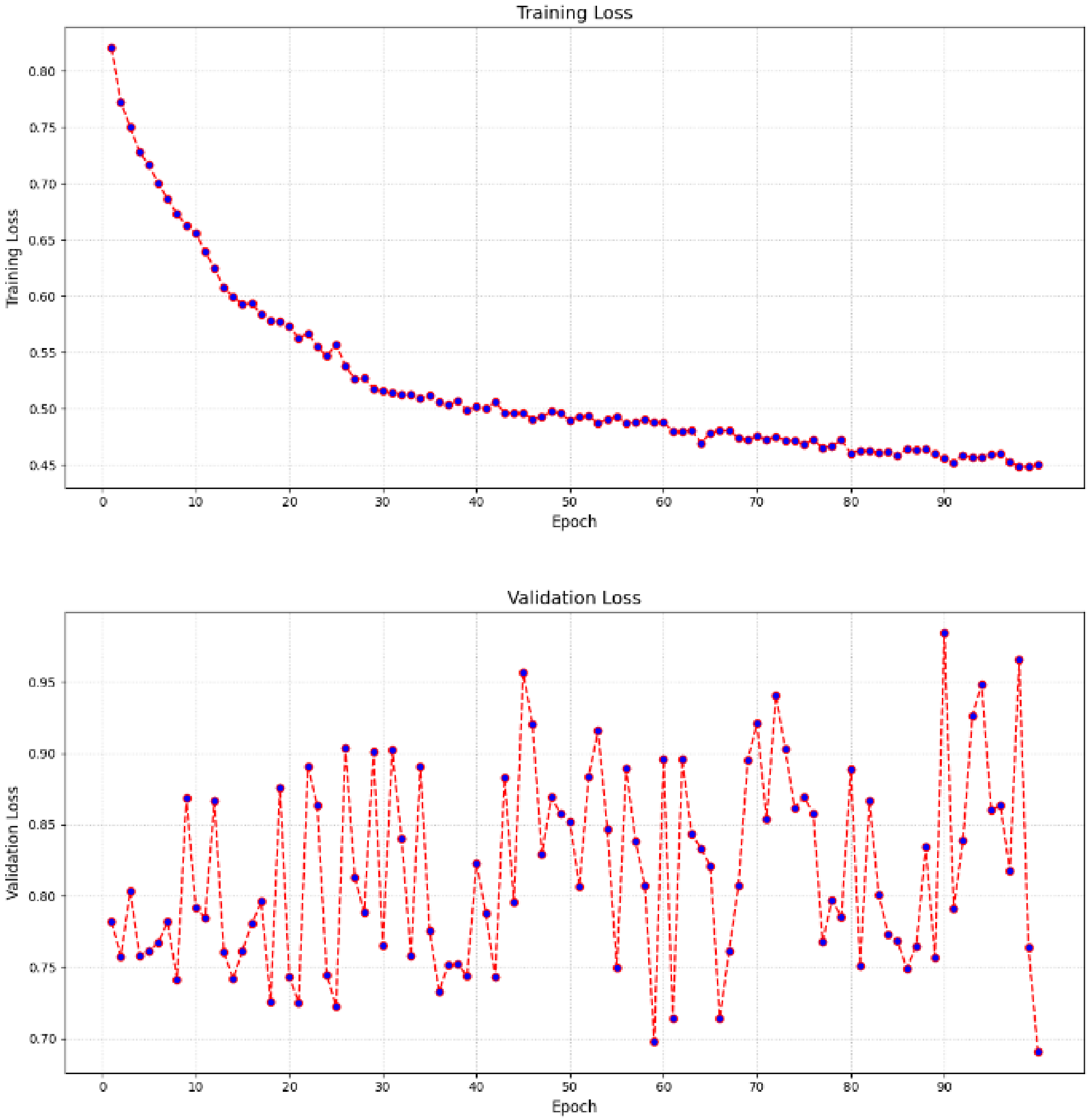
Table 5: Ablation study on different schedulers with a high learning rate

Model	Schedulers	IoU
UNet	ReduceLRonPlateau	0.4196
	CosineAnnealingWarmRestarts	0.4122
SAMClass	ReduceLRonPlateau	0.7597
	CosineAnnealingWarmRestarts	0.7027

Table 5 shows that both models using ReduceLRonPlateau achieve better IoU than using CosineAnnealingWarmRestats. The detailed training and validation loss can be found in Figure 8.



(a) Training and validation losses using CosineAnnealingWarmRestarts

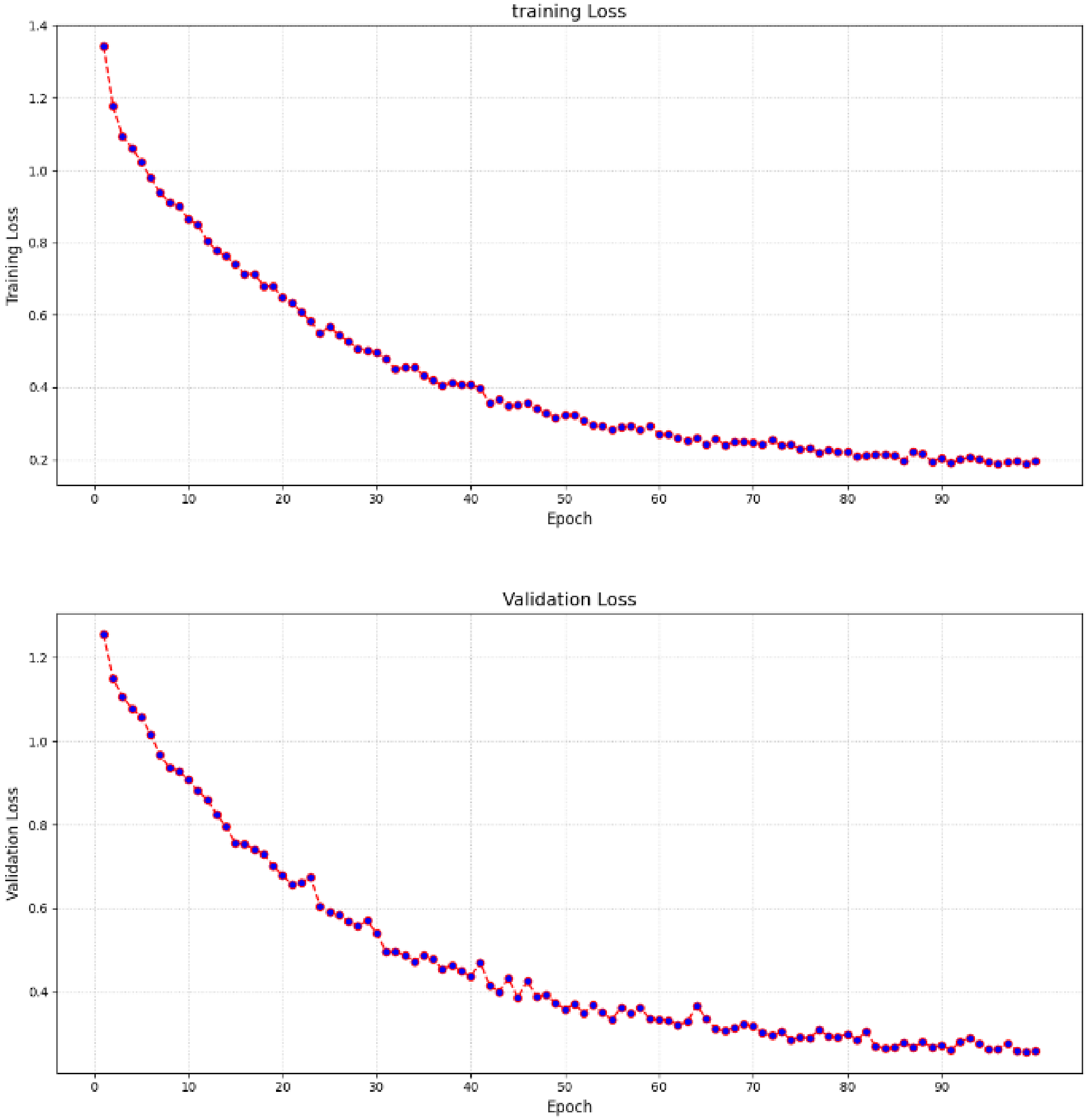


(b) Training and validation losses using ReduceLRonPlateau

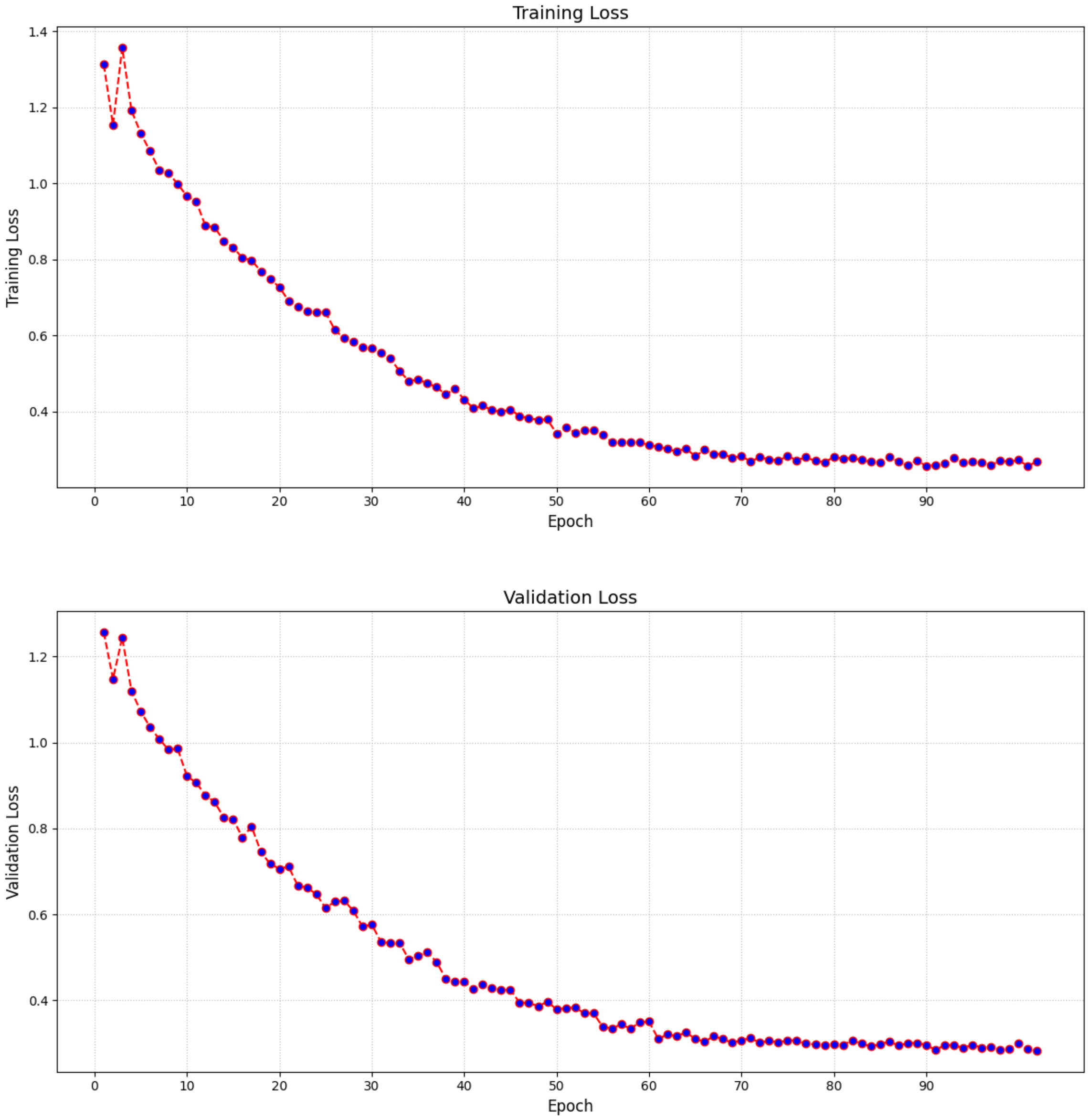
Figure 8: Training and validation loss of using different schedulers on UNet model

Figure 8 illustrates that although the training loss is declining, the validation loss is still unstable, indicting a potential overfitting or challenges in generalization to the validation data.

For the SAMClass model, the training loss and validation loss using different schedulers are shown in Figure 9.



(a) Training and validation losses using CosineAnnealingWarmRestarts



(b) Training and validation losses using ReduceLRonPlateau

Figure 9: Training and validation losses of using different schedulers on SAMClass model

Figure 9 demonstrates that the training and validation losses for SAMClass model using both schedulers keep decreasing during training. As the epoch number increases, both losses are stabilized. Moreover, the training and validation losses for SAMClass models using both schedulers are almost the same, indicating effective learning without overfitting.

4.2.3 ABLATION STUDY ON USING DIFFERENT WEIGHTS FOR MUTI-TASK LOSS FUNCTION IN SAMCLASS

From previous sections, it can be observed that SAMClass model with the learning rate of 1×10^{-3} and ReduceLRonPlateau has the best performance. The ablation study on different weighting factors to the loss of SAMCLASS model adopts these parameters, and the results after 100 epochs are listed in Table 6.

Table 6: Different weighting factors v.s. training results

Weights	mIoU	Accuracy	Precision	Recall
$w_d = w_f = w_c = 1$	0.5648	0.9808	0.9432	0.6099
$w_d = w_f = 1$ $w_c = 2$	0.4968	0.9911	0.2567	0.2201

After adjusting the weighting factor, the precision and recall of the model experience significant drop while mIoU and the accuracy keep almost the same.

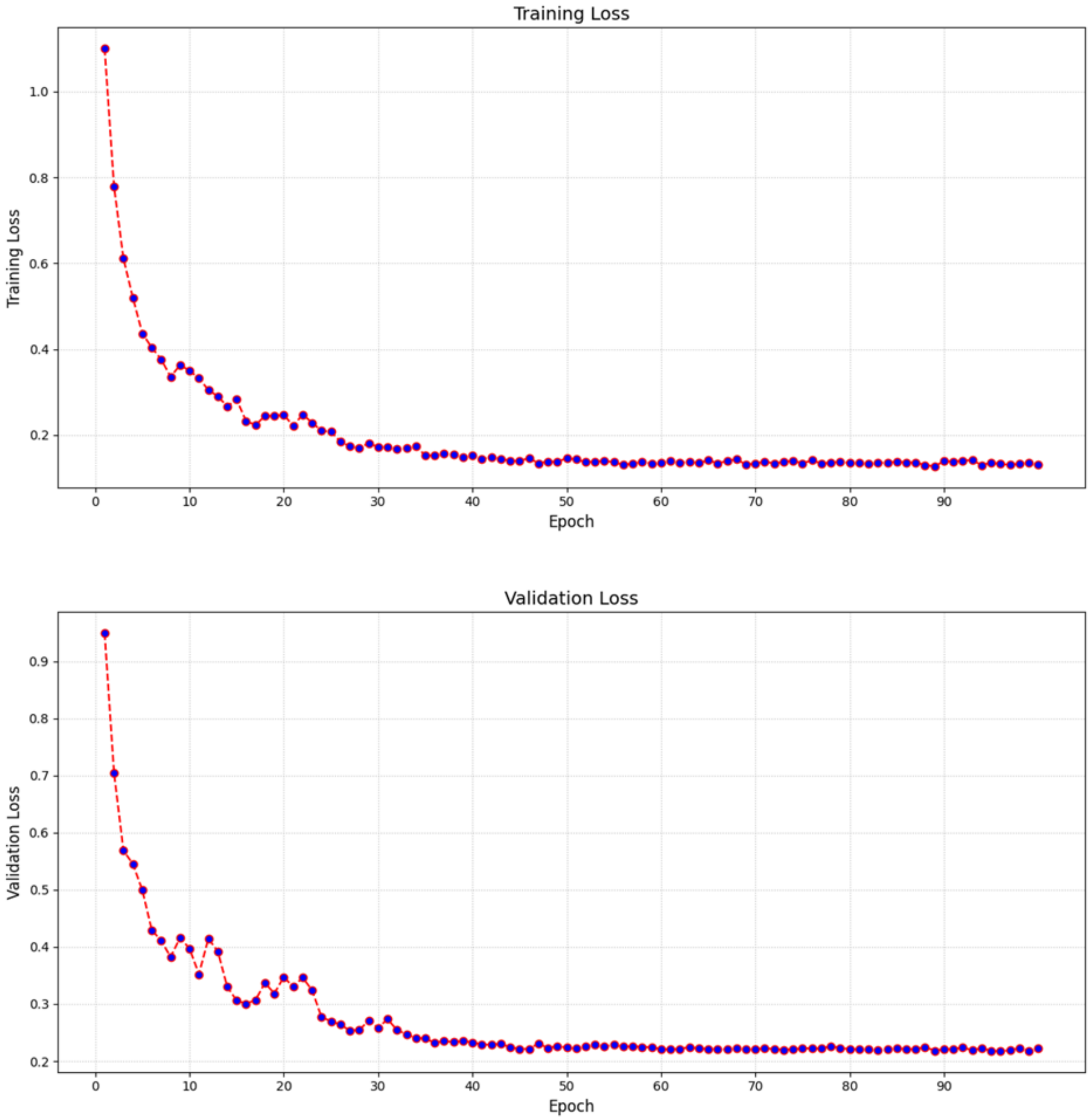
4.2.4 ABLATION STUDY ON USING DIFFERENT DROPOUT RATES FOR SAMCLASS MODEL

In this ablation study, the learning rate is 1×10^{-4}, weighting factors for different losses are equal to 1, and ReduceLRonPlateau is employed as scheduler. The training results for utilizing different dropout rates for SAMClass model are listed in Table 7.

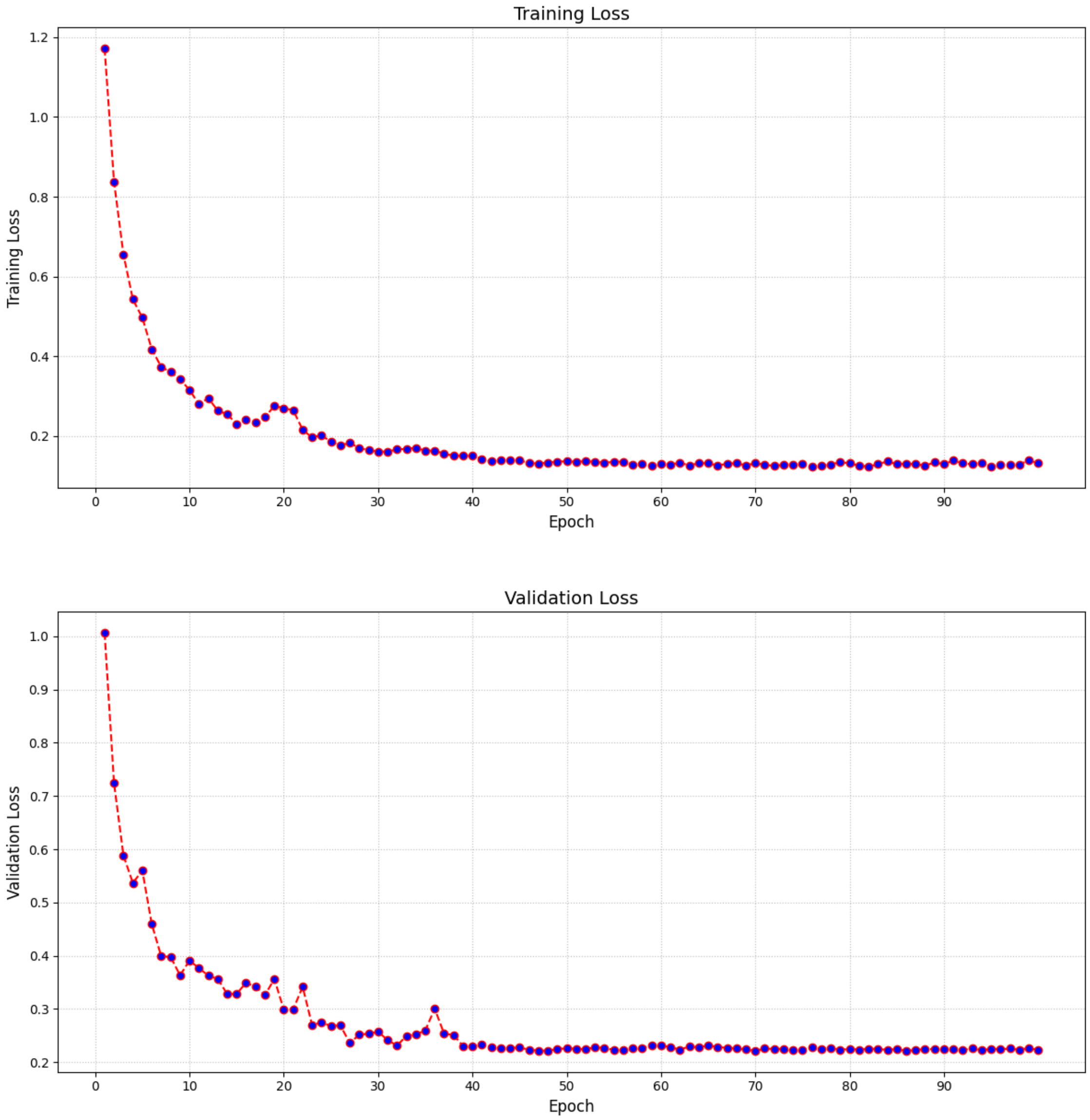
Table 7: Different dropout rates v.s. training results

Dropout Rate	IoU	Accuracy	Precision	Recall
0.6	0.7432	0.9966	0.8562	0.5946
0.4	0.7389	0.9991	0.9485	0.5271
0.2	0.6836	0.9803	0.8359	0.7681

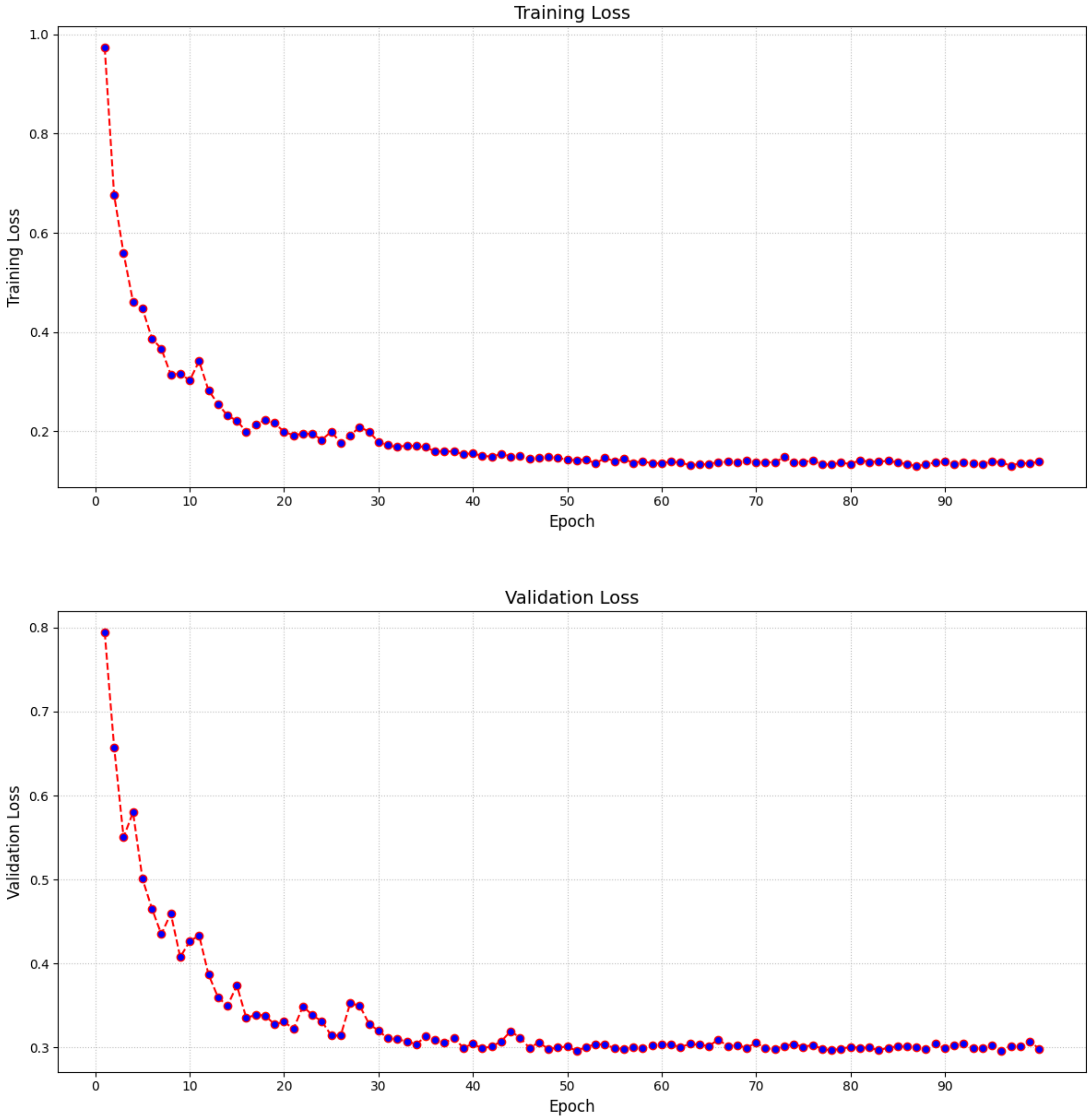
Figure 10 shows the training loss and validation loss for the three dropout rates.



(a) Training and validation losses for dropout rate equals to 0.6



(b) Training and validation losses for dropout rate equals to 0.4



(c) Training and validation losses for dropout rate equals 0.2

Figure 10: Training and validation losses for different dropout rates

4.2.5 QUANTITATIVE SEGMENTATION PERFORMANCE COMPARISON BETWEEN SAMCLASS AND UNET

After the ablation studies, the combination with best performance is obtained and listed in Table 8.

Table 8: Configurations of the best models

Model	Configurations					
	lr	Epochs	Image size	Scheduler	Weigting	Dropout
SAMClass	1e-3	100	1024*1024	CosineAnnealing	$w_f = w_d = w_c = 1$	0.6
UNet	1e-5	100	1024*1024	ReduceLRonPlateau	$w_f = w_d = 1$	0.6

Table 9 lists the best performance for both SAMClass and UNet models.

Table 9: Models with the best classification and segmentation performances

Model	IoU	Accuracy	Precision	Recall
SAMClass	0.7642	0.9808	0.9432	0.7819
UNet	0.4196	N/A	N/A	N/A

For the SAMClass model, class-wise mIoU is calculated. The result is listed in table 10.

Table 10: Class-wise mIoU of optimized SAMClass model

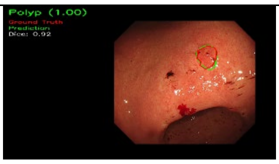
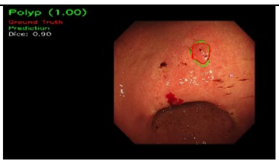
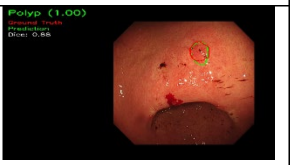
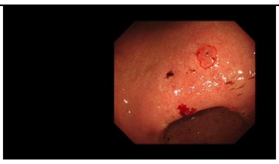
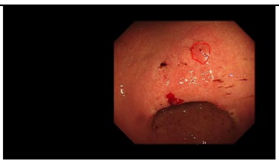
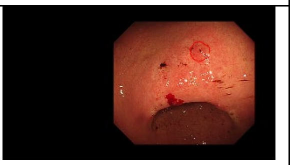
Model	Model

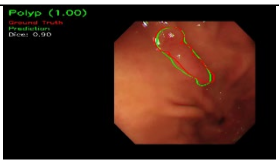
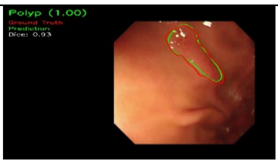
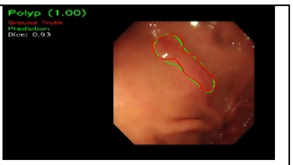
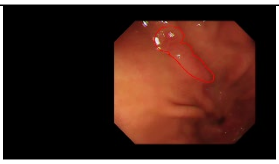
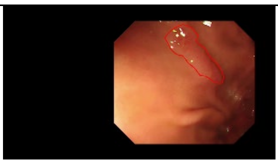
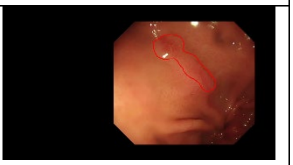
	Healthy	Polyp	GERD
SAMClass	0.6382	0.8347	0.8199

4.2.6 QUALITATIVE RESULTS OF SAMCLASS AND UNET

Due to design specification, SAMClass can produce disease classification results while UNet cannot do the classification. The visualized qualitative results for the best model can be found in Table 11. The red region represents the ground truth mask, while the green area indicates the predicted lesion area.

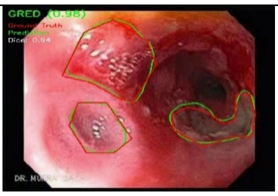
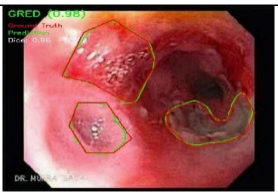
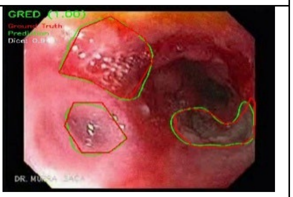
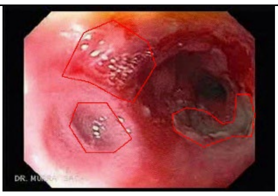
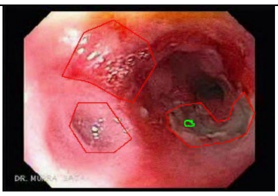
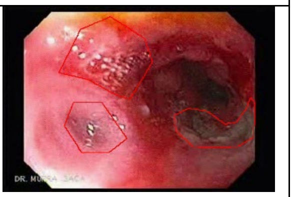
Table 11: Gastric polyp visualization results for both optimized models of continuous frames,

	Gastric polyp video 1		
Model	Frame 1	Frame 2	Frame 3
SAMClass Best	 	 	
UNet Best			
	Gastric polyp video 2		
Model	Frame 1	Frame 2	Frame 3

SAMClass Best			
UNet Best			

For the GERD disease, the visualization results are illustrated in Table 12.

Table 12: GERD visualization results for both optimized models of continuous frames

	GERD video 1		
Model	Frame 1	Frame 2	Frame 3
SAMClass Best			
UNet Best			
	GERD video 2		
Model	Frame 1	Frame 2	Frame 3

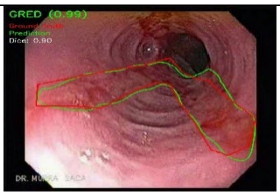
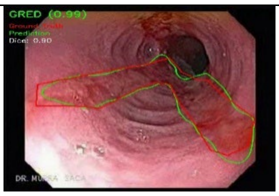
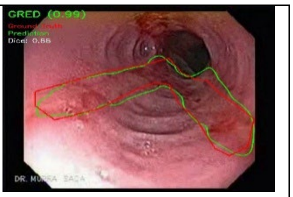
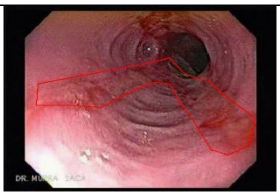
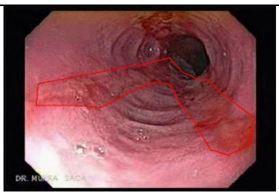
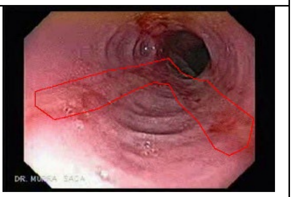
SAMClass Best			
UNet Best			

Table 11 and Table 12 show that the UNet model cannot perform the segmentation task. Lesions are not detected in many frames, resulting only in the red area, which represents the ground truth.

On the other hand, the SAMClass model can achieve visually satisfactory performance, where the predicted lesion areas can overlap with ground truth lesion areas with minor slight differences in the margin. As videos advance, the predicted lesion area can also move as the real-time ground truth area. The classification result of SAMClass can also predict the lesion type for all input videos with high precision.

4.3 DISCUSSION

In this study, both SAMClass and UNet are trained, and ablation studies are conducted to evaluate the impact of learning rates, schedulers, and loss function weightings for SAMClass. The best-performing models for both architectures are selected, and their validation results are reported.

Compared to existing UNet, our proposed models achieve successful lesion segmentation and classification. The best SAMClass model achieves an IoU exceeding 0.75. In addition to segmentation performance, lesion-type classification exhibits high accuracy, with key metrics of 0.9808 (Accuracy) and 0.9432 (Precision). The proposed model is trained and validated on continuous medical video datasets rather than individual images, enabling direct lesion detection and classification on raw endoscopy video outputs without requiring manual frame extraction.

CHAPTER 5: CONCLUSION AND FUTURE WORK

The widespread GI diseases are mainly caused by the highly diverse diet structure and the huge food supply in modern society. Many patients suffer from them and their complications today. As a result, a timely and accurate diagnosis is important and necessary in preventing early-stage GI deterioration from becoming dangerous and uncontrollable. Traditional GI detections are performed by well-trained professionals with the help of endoscopy and require significant educational and time investments.

This research is the first attempt at integrating the state-of-the-art SAM model with a classification head in order to outline lesions and clarify diseases, which include upper gastrointestinal polyp and gastroesophageal reflux disease (GERD). As the main objective of this study is to outline the lesions, the proposed SAMClass model compares with the classical segmentation model for video segmentation (UNet). Model optimization of the best models is also carried out by ablation studies on different parameters. The best performance of UNet and SAMClass models are then quantitatively compared. Experimental results show that SAMClass model outperforms UNet, achieving a mIoU of 0.7642, compared to the IoU of 0.4186 of UNet model. The proposed SAMClass model can identify lesion areas from the endoscopy video frames and can also classify the types of diseases successfully for all input videos, while UNet does not have classification ability. This work is the first implementation of a multi-task polyp and GERD segmentation in upper GI endoscopic videos. Besides, it addresses the dynamic GERD lesion analysis in endoscopic video data, making this study a pioneer in real-time GI disease diagnosis.

The future works include

1. expanding the dataset to incorporate more disease classes, which will enhance model generalizability.
2. Extending clinical applicability through transfer learning, which could enable adaptation to diverse anatomical regions and clinical scenarios.
3. Deploying the SAMClass on endoscopy devices to implement real-time endoscopic video segmentation and classification.

REFERENCES

Ahmadi, M., Nia, M. F., Asgarian, S., Danesh, K., Irankhah, E., Lonbar, A. G., & Sharifi, A. (2023). Comparative analysis of Segment Anything Model and U-Net for breast tumor detection in ultrasound and mammography images. arXiv.org. https://arxiv.org/abs/2306.12510

Audeh, B., Beaune, P., & Beigbeder, M. (2013). Recall-Oriented evaluation for information retrieval systems. In Lecture notes in computer science (pp. 29–32). https://doi.org/10.1007/978-3-642-41057-4_4

Bhattacharjee, P. K., & Chakraborty, S. (2023). Gastrointestinal Polyps with Atypical Presentations. Journal of Medical Sciences, 43(4), 190–194. https://doi.org/10.4103/jmedsci.jmedsci_218_22

Buslaev, A., Iglovikov, V. I., Khvedchenya, E., Parinov, A., Druzhinin, M., & Kalinin, A. A. (2020). Albumentations: fast and flexible image augmentations. Information, 11(2), 125. https://doi.org/10.3390/info11020125

Cao, W., Pomeroy, M. J., Zhang, S., Tan, J., Liang, Z., Gao, Y., Abbasi, A. F., & Pickhardt, P. J. (2022). An adaptive learning model for multiscale texture features in polyp classification via computed tomographic colonography. Sensors, 22(3), 907. https://doi.org/10.3390/s22030907

Castro-Pareja, C. R., Daly, B., & Shekhar, R. (2006). Elastic registration using 3D ChainMail: application to virtual colonoscopy. *Proceedings of SPIE, the International Society for Optical Engineering, 6144*, 61442Y. https://doi.org/10.1117/12.653644

Chahbar, F., Merati, M., & Mahmoudi, S. (2025). MPB-UNet: Multi-Parallel Blocks UNet for MRI Automated Brain Tumor Segmentation. Electronics, 14(1), 40. https://doi.org/10.3390/electronics14010040

Chang, Q., Ahmad, D., Toth, J., Bascom, R., & Higgins, W. E. (2024). ESFPNET: Efficient Stage-Wise Feature Pyramid on Mix Transformer for Deep Learning-Based cancer analysis in endoscopic video. Journal of Imaging, 10(8), 191. https://doi.org/10.3390/jimaging10080191

Chen, F., Tang, J., Wang, P., Wang, T., Li, S., & Deng, T. (2024). DEAP-3DSAM: Decoder Enhanced and Auto Prompt SAM for 3D medical image segmentation. *2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 1852–1859. https://doi.org/10.1109/bibm62325.2024.10822764

Chen, J. (2022). Optimized Hybrid Focal Margin Loss for Crack Segmentation. 1–7. https://doi.org/10.1109/dicta56598.2022.10034608

Cho, B., Bang, C. S., Park, S. W., Yang, Y. J., Seo, S. I., Lim, H., Shin, W. G., Hong, J. T., Yoo, Y. T., Hong, S. H., Choi, J. H., Lee, J. J., & Baik, G. H. (2019). Automated classification of gastric neoplasms in endoscopic images using a convolutional neural network. Endoscopy, 51(12), 1121–1129. https://doi.org/10.1055/a-0981-6133

Clavero, J. M., Topart, P., & Deschamps, C. (2006). Management of alkaline reflux. In Springer eBooks (pp. 139–149). https://doi.org/10.1007/1-84628-011-7_12

Cuccu, G., Jobin, J., Clement, J., Bhardwaj, A., Reischauer, C., Thony, H., & Cudre-Mauroux, P. (2020). Hydra: Cancer detection leveraging multiple heads and heterogeneous datasets. 2021 IEEE International Conference on Big Data, 4842–4849. https://doi.org/10.1109/bigdata50022.2020.9378042

De Santiago, E. R., Peñas, B., Mesonero, F., Parejo, S., & Albillos, A. (2016). Cáncer colorrectal. Medicine - Programa De Formación Médica Continuada Acreditado, 12(6), 297–307. https://doi.org/10.1016/j.med.2016.03.003

Esposito, F., & Cappabianca, P. (2013). Neuroendoscopy: General Aspects and Principles. World Neurosurgery, 79(2), S14.e7–S14.e9. https://doi.org/10.1016/j.wneu.2012.02.033

Fan, G., Wang, J., & Zhang, C. (2023). "SACA-UNet:Medical Image Segmentation Network Based on Self-Attention and ASPP," 2023 IEEE 36th International Symposium on Computer-Based Medical Systems (CBMS), L'Aquila, Italy, 2023, pp. 317-322. https://doi.org/10.1109/cbms58004.2023.00237

Feng, T., Wang, J., Shen, J., Jin, Q., Zhu, Z., & Wang, X. (2024). Retinal vessel segmentation via cross-attention feature fusion. 2022 IEEE International Conference on Multimedia and Expo (ICME), 1–6. https://doi.org/10.1109/icme57554.2024.10688098

Fu, G., Lu, H., Tan, J. K., Kim, H., Zhu, X., & Lu, J. (2018). "Segmentation of Spinal Canal Region in CT Images using 3D Region Growing Technique," 2018 International Conference on Information and Communication Technology Robotics (ICT-ROBOT), Busan, Korea (South), 2018, pp. 1-4. https://doi.org/10.1109/ict-robot.2018.8549913

Gao, H., Chen, Z., & Li, C. (2021). Shallow network based on depthwise overparameterized convolution for hyperspectral image classification. IEEE Geoscience and Remote Sensing Letters, 19, 1–5. https://doi.org/10.1109/lgrs.2021.3133598

Ge, H., Zhou, X., Wang, Y., Xu, J., Mo, F., Chao, C., Zhu, J., & Yu, W. (2023). Development and validation of deep learning models for the multiclassification of reflux esophagitis based on the Los Angeles classification. Journal of Healthcare Engineering, 2023(1). https://doi.org/10.1155/2023/7023731

Guarneri, F., Vaccaro, M., & Guarneri, C. (2008). Digital Image Compression in Dermatology: Format comparison. Telemedicine Journal and e-Health, 14(7), 666–670. https://doi.org/10.1089/tmj.2007.0119

Han, M., Shi, P., & Bao, X. (2019). "An Adaptive Pooling Model for Aesthetic Quality Assessment Based on Convolutional Neural Network," 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 2019, pp. 358-361. https://doi.org/10.1109/icsess47205.2019.9040714

He, D., Ma, Z., Li, C., & Li, Y. (2024). Dual-Branch fully convolutional segment anything model for lesion segmentation in endoscopic images. IEEE Access, 12, 125654–125667. https://doi.org/10.1109/access.2024.3449428

Hossain, M. S., Betts, J. M., & Paplinski, A. P. (2021). Dual Focal Loss to address class imbalance in semantic segmentation. Neurocomputing, 462, 69–87. https://doi.org/10.1016/j.neucom.2021.07.055

Hu, Y., Long, Z., & AlRegib, G. (2019). Multi-Level Texture Encoding and Representation (MULTER) based on deep neural networks. 2022 IEEE International Conference on Image Processing (ICIP), 4410–4414. https://doi.org/10.1109/icip.2019.8803640

Hughes, G. J. (2021). Gastrointestinal Endoscopy - Upper (pp. 144–148). Elsevier. https://doi.org/10.1016/B978-0-323-79007-9.00032

Huo, J., Li, W., & Wang, B. (2010). "Clustering Ensemble Based on a New Consensus Function with Hamming Distance," 2010 2nd International Symposium on Information Engineering and Electronic Commerce, Ternopil, Ukraine, 2010, pp. 1-4. https://doi.org/10.1109/ieec.2010.5533268

ImageNet. (n.d.). https://image-net.org/

Izzaturrahmah, A. N., Nhita, F., & Kurniawan, I. (2021). "Implementation of Support Vector Machine on Text-based GERD Detection by using Drug Review Content," 2021 International Conference Advancement in Data Science, E-learning and Information Systems (ICADEIS), Bali, Indonesia, 2021, pp. 1-6. https://doi.org/10.1109/icadeis52521.2021.9702073

Jiajun, C., Kaixiang, L., Renjian, L., Chunlei, S., Guiye, L., & Lingling, C. (2023). AtG-DeepLab V3+ Endoscopic Image Enhancement Algorithm Based on Self-attention Mechanism Optimization. ACTA Photonica Sinica, 52(8), 0817001. https://doi.org/10.3788/gzxb20235208.0817001

Jurdi, R. E., & Colliot, O. (2023). How Precise are Performance Estimates for Typical Medical Image Segmentation Tasks? 2022 IEEE 19th International Symposium on Biomedical Imaging (ISBI), 1–5. https://doi.org/10.1109/isbi53787.2023.10230401

Katzka, D. A., & Kahrilas, P. J. (2020). Advances in the diagnosis and management of gastroesophageal reflux disease. BMJ, 371. https://doi.org/10.1136/bmj.m3786

Kellerman, R., & Kintanar, T. (2017). Gastroesophageal reflux disease. Primary Care Clinics in Office Practice, 44(4), 561–573. https://doi.org/10.1016/j.pop.2017.07.001

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

Kirimtat, A., & Krejcar, O. (2023). A Guide and Mini-Review on the Performance Evaluation Metrics in binary Segmentation of magnetic resonance Images. In Lecture notes in computer science (pp. 428–440). https://doi.org/10.1007/978-3-031-34960-7_30

Kohli, M. D., Summers, R. M., & Geis, J. R. (2017). Medical Image Data and Datasets in the Era of Machine Learning—Whitepaper from the 2016 C-MIMI Meeting Dataset Session. Journal of Digital Imaging, 30(4), 392–399. https://doi.org/10.1007/s10278-017-9976-3

Lan, Y., Weng, Z., & Zhu, Y. (2018). Center-Adaptive weighted binary K-Means for image clustering. In Lecture notes in computer science (pp. 407–417). https://doi.org/10.1007/978-3-319-77383-4_40

Li, B. C. (2017). Fast Legendre moment computation for template matching. Proceedings of SPIE, the International Society for Optical Engineering, 10202, 102020J. https://doi.org/10.1117/12.2262783

Liu, X., Zhao, Y., Wang, S., & Wei, J. (2024). G-SAM: GMM-based segment anything model for medical image classification and segmentation. *Cluster Computing*, *27*(10), 14231–14245. https://doi.org/10.1007/s10586-024-04679-x

Liu, Y., Saber, E., Glading, A., & Helguera, M. (2014). Measurement of blood flow velocity forin vivovideo sequences with motion estimation methods. Proceedings of the International Society for Optical Engineering, 9034, 90342W. https://doi.org/10.1117/12.2043255

Lu, D., & Liu, S. (2024). "Real Time Performance Evaluation of Deep Learning Algorithms in Image Recognition under the Pytorch Framework," 2024 International Conference on Intelligent Algorithms for Computational Intelligence Systems (IACIS), Hassan, India, 2024, pp. 1-6. https://doi.org/10.1109/iacis61494.2024.10721731

Ma, M. X., & Bourke, M. J. (2017). Sessile Serrated Adenomas: How to Detect, Characterize and Resect. Gut and Liver, 11(6), 747–760. https://doi.org/10.5009/gnl16523

Martins, B. C., Moura, R. N., Kum, A. S. T., Matsubayashi, C. O., Marques, S. B., & Safatle-Ribeiro, A. V. (2023). Endoscopic imaging for the diagnosis of neoplastic and Pre-Neoplastic conditions of the stomach. Cancers, 15(9), 2445. https://doi.org/10.3390/cancers15092445

Mayo Clinic. (n.d.). *Upper endoscopy*. Upper Endoscopy - Mayo Clinic. https://www.mayoclinic.org/tests-procedures/endoscopy/about/pac-20395197

Milesial. (n.d.). GitHub - milesial/Pytorch-UNet: PyTorch implementation of the U-Net for image semantic segmentation with high quality images. GitHub. https://github.com/milesial/Pytorch-UNet

Nguyen, G.-V., & Huynh-The, T. (2024). Enhancing Aerial Semantic Segmentation With Feature Aggregation Network for DeepLabV3+, IEEE Geoscience and Remote Sensing Letters, vol. 21, pp. 1-5, 2024, no. 2504205. https://doi.org/10.1109/lgrs.2024.3432922

Nia, N. G., Kaplanoglu, E., & Nasab, A. (2023). Evaluation of artificial intelligence techniques in disease diagnosis and prediction. Discover Artificial Intelligence, 3(1). https://doi.org/10.1007/s44163-023-00049-5

Nie, Z., Xu, M., Wang, Z., Lu, X., & Song, W. (2024). A review of application of deep learning in endoscopic image processing. Journal of Imaging, 10(11), 275. https://doi.org/10.3390/jimaging10110275

Oh, C., Han, S., & Jeong, J. (2020). Time-Series Data Augmentation based on Interpolation. *Procedia Computer Science*, *175*, 64–71. https://doi.org/10.1016/j.procs.2020.07.012

Openmedlab. (n.d.). GitHub - openmedlab/Endo-FM: [MICCAI'23] Foundation Model for Endoscopy Video Analysis via Large-scale Self-supervised Pre-train. GitHub. https://github.com/openmedlab/Endo-FM

Pogorelov, K., Randel, K. R., Griwodz, C., Eskeland, S. L., de Lange, T., Johansen, D., ... & Halvorsen, P. (2017). Kvasir: A multi-class image dataset for computer aided gastrointestinal disease detection. In Proceedings of the 8th ACM on Multimedia Systems Conference pp. 164-169. https://doi.org/10.1145/3083187.3083212

Pytorch. (n.d.). GitHub - pytorch/pytorch: Tensors and Dynamic neural networks in Python with strong GPU acceleration. GitHub. https://github.com/pytorch/pytorch

Quandhiti. (n.d.). GitHub - quandhiti/ITI-GERD. GitHub. https://github.com/quandhiti/ITI-GERD

Rana, M., & Bhushan, M. (2022). Machine learning and deep learning approach for medical image analysis: diagnosis to detection. Multimedia Tools and Applications, 82(17), 26731–26769. https://doi.org/10.1007/s11042-022-14305-w

Redfish-Github. (n.d.). GitHub - redfish-github/tvi: Turn the video into image. GitHub. https://github.com/redfish-github/tvi

Sahu, S., Sahu, S. P., & Deepak Kumar Dewangan. (2022). Pedestrian Detection Using MobileNetV2 Based Mask R-CNN. Lecture Notes in Networks and Systems, 299–318. https://doi.org/10.1007/978-981-19-5845-8_22

Sengar, S. S., Meulengracht, C., Boesen, M. P., Overgaard, A. F., Gudbergsen, H., Nybing, J. D., Perslev, M., & Dam, E. B. (2022). Multi-planar 3D knee MRI segmentation via UNet inspired architectures. International Journal of Imaging Systems and Technology, 33(3), 985–998. https://doi.org/10.1002/ima.22836

Singh, J., Beeche, C., Shi, Z., Beale, O., Rosin, B., Leader, J., & Pu, J. (2023). Batch-balanced focal loss: a hybrid solution to class imbalance in deep learning. Journal of Medical Imaging, 10(05). https://doi.org/10.1117/1.jmi.10.5.051809

Sharma, A., Kumar, R., Yadav, G., & Garg, P. (2023). Artificial intelligence in intestinal polyp and colorectal cancer prediction. Cancer Letters, 565, 216238. https://doi.org/10.1016/j.canlet.2023.216238

Sharma, J. (2024). "Enhanced Rose Leaf Disease Classification Using Vision Transformer (ViT-B/16) Detecting Black Spot, Downy Mildew, and Healthy Leaves for Improved Plant Health Management," 2024 5th International Conference on Data Intelligence and Cognitive Informatics (ICDICI), Tirunelveli, India, 2024, pp. 52-56. https://doi.org/10.1109/icdici62993.2024.10810822

Shirokikh, B., Shevtsov, A., Kurmukov, A., Dalechina, A., Krivov, E., Kostjuchenko, V., Golanov, A., & Belyaev, M. (2020). Universal Loss Reweighting to balance lesion size inequality in 3D medical image segmentation. In Lecture notes in computer science (pp. 523–532). https://doi.org/10.1007/978-3-030-59719-1_51

Sloan, J., & Katz, P. O. (2019). Gastroesophageal reflux disease. In Elsevier eBooks (pp. 197–203). https://doi.org/10.1016/b978-0-323-40232-3.00015-7

Spadaccini, M., Menini, M., Massimi, D., Rizkala, T., De Sire, R., Alfarone, L., Capogreco, A., Colombo, M., Maselli, R., Fugazza, A., Brandaleone, L., Di Martino, A., Ramai, D., Repici, A., & Hassan, C. (2025). AI and Polyp Detection During Colonoscopy. Cancers, 17(5), 797. https://doi.org/10.3390/cancers17050797

Tang, G., & Fan, N. (2022). A Survey of Solution Path Algorithms for Regression and Classification Models. Annals of Data Science, 9(4), 749–789. https://doi.org/10.1007/s40745-022-00386-9

Tang, S., Yu, X., Cheang, C. F., Liang, Y., Zhao, P., Yu, H. H., & Choi, I. C. (2023). Transformer-based multi-task learning for classification and segmentation of gastrointestinal tract endoscopic images. Computers in Biology and Medicine, 157, 106723. https://doi.org/10.1016/j.compbiomed.2023.106723

Tran, T., Vu, D. H., Tran, D. H., Linh, K., DO, Nguyen, P. T., Nguyen, V. T., Nguyen, L. T., Ho, N. K., Vu, H., & Dao, V. H. (2022). DCS-UNet: Dual-Path Framework for Segmentation of Reflux Esophagitis Lesions from Endoscopic Images with U-Net-Based Segmentation and Color/Texture Analysis. Vietnam Journal of Computer Science, 10(02), 217–242. https://doi.org/10.1142/s2196888822500385

Trivedi, U. B., Bhatt, M., & Srivastava, P. (2021). Prevent overfitting problem in machine learning: a case focus on linear regression and logistics regression. In Advances in Science, Technology & Innovation/Advances in science, technology & innovation (pp. 345–349). https://doi.org/10.1007/978-3-030-66218-9_40

Vietanhdev. (n.d.). Releases · vietanhdev/anylabeling. GitHub. https://github.com/vietanhdev/anylabeling/releases

Wang, L., Wang, C., Sun, Z., & Chen, S. (2020). An improved dice loss for pneumothorax segmentation by mining the information of negative areas. IEEE Access, 8, 167939–167949. https://doi.org/10.1109/access.2020.3020475

Wzh. (n.d.). GitHub - WZH0120/SAM2-UNET: SAM2-UNET: Segment Anything 2 makes strong encoder for natural and medical image segmentation. GitHub. https://github.com/WZH0120/SAM2-UNet

Xu, M., Zhou, W., Wu, L., Zhang, J., Wang, J., Mu, G., Huang, X., Li, Y., Yuan, J., Zeng, Z., Wang, Y., Huang, L., Liu, J., & Yu, H. (2021). Artificial intelligence in the diagnosis of gastric precancerous conditions by image-enhanced endoscopy: a multicenter, diagnostic study (with video). Gastrointestinal Endoscopy, 94(3), 540-548.e4. https://doi.org/10.1016/j.gie.2021.03.013

Xuanhao, Q., & Min, Z. (2023). A review of attention mechanisms in Computer Vision. 2022 7th International Conference on Image, Vision and Computing (ICIVC), 36, 577–583. https://doi.org/10.1109/icivc58118.2023.10270435

Yang, X., Feng, L., Lu, T., & Dong, Q. (2022). Application of image hash algorithm in copyright protection system. Third International Conference on Electronics and Communication; Network and Computer Technology (ECNCT 2021), 32, 87. https://doi.org/10.1117/12.2628802

Zhang, X., Hu, Y., Liu, J., Zhang, X., & Wu, B. (2022). Robust rotating machinery diagnosis using a dynamic-weighted graph updating strategy. Measurement, 202, 111895. https://doi.org/10.1016/j.measurement.2022.111895

Zhang, Y., Liu, S., Li, C., & Wang, J. (2021). Rethinking the dice loss for deep learning lesion segmentation in medical images. *Journal of Shanghai Jiaotong University (Science)*, *26*(1), 93–102. https://doi.org/10.1007/s12204-021-2264-x

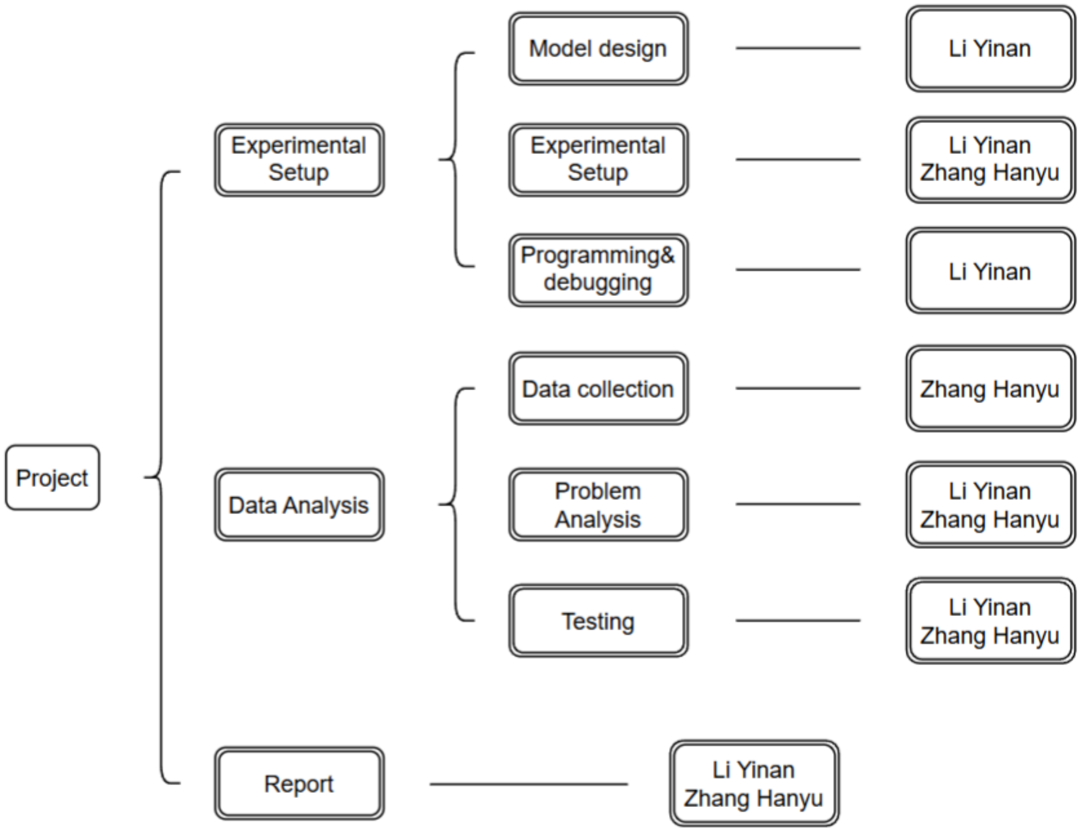
Zhao, J., Sun, L., Fan, D., Wang, K., Si, H., Fu, H., & Zhang, D. (2025). "Uncertainty-Driven Edge Prompt Generation Network for Medical Image Segmentation," in IEEE Transactions on Medical Imaging. https://doi.org/10.1109/tmi.2025.3535478

Zhao, R., Qian, B., Zhang, X., Li, Y., Wei, R., Liu, Y., & Pan, Y. (2020). Rethinking dice loss for medical image segmentation. 2021 IEEE International Conference on Data Mining (ICDM), 851–860. https://doi.org/10.1109/icdm50108.2020.00094

Zhao, X., & Liu, Y. (2022). False awareness stock market prediction by LightGBM with focal loss. 13, 147–152. https://doi.org/10.1145/3538641.3561502

Zhou, Z., Siddiquee, M. M. R., Tajbakhsh, N., & Liang, J. (2020). UNet++: Redesigning Skip Connections to Exploit Multiscale Features in Image Segmentation, IEEE Transactions on Medical Imaging, vol. 39, no. 6, pp. 1856-1867. https://doi.org/10.1109/tmi.2019.2959609

APPENDIX I: WORK BREAKDOWN



APPENDIX II: PYTHON CODE FOR MASK GENERATION

```
import os
import json
import numpy as np
from PIL import Image, ImageDraw

def generate_mask_and_frame(input_dir, frame_output_dir, mask_output_dir,
videolist_path):
    os.makedirs(frame_output_dir, exist_ok=True)
    os.makedirs(mask_output_dir, exist_ok=True)
    os.makedirs(os.path.dirname(videolist_path), exist_ok=True)

    existing_frames = []
    for f in os.listdir(frame_output_dir):
        if f.endswith('.tif'):
            try:
                num = int(os.path.splitext(f)[0])
                existing_frames.append(num)
            except ValueError:
                pass

    max_current = max(existing_frames) if existing_frames else 0

    png_files = [f for f in os.listdir(input_dir) if f.endswith('.png')]
    json_files = [f for f in os.listdir(input_dir) if f.endswith('.json')]

    png_numbers = [int(os.path.splitext(f)[0]) for f in png_files]
    json_numbers = [int(os.path.splitext(f)[0]) for f in json_files]

    if sorted(png_numbers) != sorted(json_numbers):
        raise ValueError("PNG和JSON文件编号不匹配")

    sorted_png_files = [f for _, f in sorted(zip(png_numbers, png_files))]
    sorted_json_files = [f for _, f in sorted(zip(json_numbers, json_files))]

    frame_count = len(sorted_png_files)
    for index, (png_file, json_file) in enumerate(zip(sorted_png_files,
sorted_json_files)):
        png_path = os.path.join(input_dir, png_file)
        json_path = os.path.join(input_dir, json_file)

        with open(json_path, 'r') as f:
            json_data = json.load(f)

        if json_data["imagePath"] != png_file:
            raise ValueError(f"JSON文件中的imagePath与PNG文件名不匹配: {json_file}")

        frame_image = Image.open(png_path)
        mask_image = Image.new('L', frame_image.size, 0)
        draw = ImageDraw.Draw(mask_image)

        for shape in json_data["shapes"]:
            if shape["shape_type"] == "polygon":
                points = [(int(p[0]), int(p[1])) for p in shape["points"]]
                draw.polygon(points, fill=255)

        frame_number = max_current + index + 1
        frame_output_path = os.path.join(frame_output_dir, f"{frame_number}.tif")
        mask_output_path = os.path.join(mask_output_dir, f"{frame_number}.tif")

        frame_image.save(frame_output_path, format='TIFF')
        mask_image.save(mask_output_path, format='TIFF')

    with open(videolist_path, 'a') as f:
        current_start = max_current + 1 if existing_frames else 1
        f.write(f"{current_start} {frame_count}\n")

    first_tif = f"{max_current + 1}.tif" if existing_frames else "1.tif"
    print(f"First tif created: {first_tif}")
    print(f"Total tifs created: {frame_count}")
    return first_tif, frame_count

if __name__ == "__main__":
    input_dir = r"C:\Users\leiya\OneDrive\桌面\out"
    frame_output_dir = r"D:\Intelligent_Medical_System\System\dataPrep\json_et_png\frame"
```

```
mask_output_dir = r"D:\Intelligent_Medical_System\System\dataPrep\json_et_png\mask"
videolist_path = r"D:\Intelligent_Medical_System\System\json_et_png\videolist.txt"

first_tif_name, total_tifs_created = generate_mask_and_frame(input_dir, frame_output_dir, mask_output_dir, videolist_path)
```

APPENDIX III: PYTHON CODE FOR VIDEO TEXT REMOVAL

```
import os
import cv2
import numpy as np

def remove_left_text(img):
  h, w = img.shape[:2]
  left_width = w // 3
  img[:, :left_width] = [0, 0, 0] if len(img.shape) == 3 else 0
  return img

# directories
original_dir = "/media/ubuntu-
user/KESU/Intelligent_Medical_System/System/dataPrep/json_et_png/frame"
mask_dir = "/media/ubuntu-
user/KESU/Intelligent_Medical_System/System/dataPrep/json_et_png/mask"
output_dir_frame = "/media/ubuntu-
user/KESU/Intelligent_Medical_System/System/dataPrep/json_et_png/frameVid/unet
"
output_dir_masks = "/media/ubuntu-
user/KESU/Intelligent_Medical_System/System/dataPrep/json_et_png/maskVid/unet
"

os.makedirs(output_dir_frame, exist_ok=True)
os.makedirs(output_dir_masks, exist_ok=True)

with open("/media/ubuntu-
user/KESU/Intelligent_Medical_System/System/dataPrep/json_et_png/videolist.txt",
"r") as file:
  lines = file.readlines()

for line_number, line in enumerate(lines, start=1):
  parts = line.strip().split()
  if len(parts) < 2:
    print(f"Skipping invalid line {line_number}")
    continue

  start_frame = int(parts[0])
```

```
num_frames = int(parts[1])

for dir_type, input_dir, output_dir in [('frame', original_dir, output_dir_frame),
                                        ('mask', mask_dir, output_dir_masks)]:

    video_name = f"{line_number}.avi"
    output_path = os.path.join(output_dir, video_name)

    frame_files = [f"{start_frame + i}.tif" for i in range(num_frames)]

    first_frame_path = os.path.join(input_dir, frame_files[0])
    if not os.path.exists(first_frame_path):
        print(f"Missing first frame {first_frame_path}")
        continue

    sample_frame = cv2.imread(first_frame_path)
    if sample_frame is None:
        print(f"Failed to read sample frame {first_frame_path}")
        continue

    processed_frame = remove_left_text(sample_frame)
    h, w = processed_frame.shape[:2]

    fourcc = cv2.VideoWriter_fourcc('F', 'F', 'V', '1')
    writer = cv2.VideoWriter(output_path, fourcc, 30, (w, h), isColor=True)

    for fname in frame_files:
        img_path = os.path.join(input_dir, fname)
        if not os.path.exists(img_path):
            break

        if dir_type == 'frame':
            img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        else:
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            if img is not None:
                _, img = cv2.threshold(img, 1, 255, cv2.THRESH_BINARY)
                img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)

        if img is None:
            print(f"Failed to read {img_path}")
```

```
                continue

            processed_img = remove_left_text(img)

            writer.write(processed_img)

        writer.release()
        print(f"Created {dir_type} video: {video_name}")

print("All videos generated successfully.")
```

APPENDIX IV: PYTHON CODE FOR CONTINUOUS FRAME DETECTION

```
import cv2
import numpy as np
import os

def load_images(folder):
    images = []
    filenames = []
    file_list = os.listdir(folder)
    file_list.sort(key=lambda x: int(os.path.splitext(x)[0]))
    for filename in file_list:
        img_path = os.path.join(folder, filename)
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        if img is not None:
            images.append(img)
            filenames.append(filename)
    return images, filenames

def compute_features(images):
    orb = cv2.ORB_create()
    descriptors = []
    for img in images:
        _, des = orb.detectAndCompute(img, None)
        descriptors.append(des)
    return descriptors

def build_similarity_matrix(descriptors):
    n = len(descriptors)
    similarity_matrix = np.zeros((n, n), dtype=int)
    bf = cv2.BFMatcher(cv2.NORM_HAMMING)

    for i in range(n):
        for j in range(n):
            if i == j or descriptors[i] is None or descriptors[j] is None:
                continue
            matches = bf.knnMatch(descriptors[i], descriptors[j], k=2)
        good = []
        for m, neighbor in matches:
            # Adjust ratio test to 0.6 to include more matches
            if m.distance < 0.6 * neighbor.distance:
                good.append(m)
        similarity_matrix[i][j] = len(good)
    return similarity_matrix

def determine_order(similarity_matrix):
    n = similarity_matrix.shape[0]
    next_frame = {}
    prev_frame = {}

    for i in range(n):
        max_sim = -1
        max_idx = -1
        for j in range(i+1, min(i+6, n)):
            if similarity_matrix[i][j] > max_sim:
                max_sim = similarity_matrix[i][j]
                max_idx = j
        next_frame[i] = max_idx if max_sim > 10 else -1

    ordered_sequence = []
    current = 0
    visited = set()

    while current != -1 and current not in visited and len(ordered_sequence) < n:
        visited.add(current)
        ordered_sequence.append(current)
        current = next_frame.get(current, -1)

    remaining = [i for i in range(n) if i not in ordered_sequence]
    ordered_sequence += remaining

    return ordered_sequence

folder_path = "/home/ubuntu-user/Downloads/ITI-GERD-main/Images"
images, filenames = load_images(folder_path)
descriptors = compute_features(images)
similarity_matrix = build_similarity_matrix(descriptors)
ordered_indices = determine_order(similarity_matrix)
```

```
ordered_files = [filenames[i] for i in ordered_indices]
print("Ordered frames:", ordered_files)
```

APPENDIX V: PYTHON CODE FOR THE UNET MODEL

```
import torch
import torch.nn as nn
import torch.nn.functional as F


class ConvBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, 3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, 3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

    def forward(self, x):
        return self.conv(x)


class EnhancedUNet(nn.Module):
    def __init__(self, in_channels=3):
```

```
super().__init__()

# Encoder
self.enc1 = ConvBlock(in_channels, 64)
self.enc2 = ConvBlock(64, 128)
self.enc3 = ConvBlock(128, 256)
self.enc4 = ConvBlock(256, 512)
self.pool = nn.MaxPool2d(2)

# Bottleneck
self.bottleneck = ConvBlock(512, 1024)

# Decoder
self.upconv4 = nn.ConvTranspose2d(1024, 512, 2, stride=2)
self.dec4 = ConvBlock(1024, 512)
self.upconv3 = nn.ConvTranspose2d(512, 256, 2, stride=2)
self.dec3 = ConvBlock(512, 256)
self.upconv2 = nn.ConvTranspose2d(256, 128, 2, stride=2)
self.dec2 = ConvBlock(256, 128)
self.upconv1 = nn.ConvTranspose2d(128, 64, 2, stride=2)
self.dec1 = ConvBlock(128, 64)

# Output layer
```

```
        self.final_conv = nn.Conv2d(64, 1, 1)

    def forward(self, x):
        # Encoder
        e1 = self.enc1(x)
        e2 = self.enc2(self.pool(e1))
        e3 = self.enc3(self.pool(e2))
        e4 = self.enc4(self.pool(e3))

        # Bottleneck
        b = self.bottleneck(self.pool(e4))

        # Decoder
        d4 = self.upconv4(b)
        d4 = torch.cat([d4, e4], dim=1)
        d4 = self.dec4(d4)

        d3 = self.upconv3(d4)
        d3 = torch.cat([d3, e3], dim=1)
        d3 = self.dec3(d3)

        d2 = self.upconv2(d3)
        d2 = torch.cat([d2, e2], dim=1)
```

```
d2 = self.dec2(d2)


d1 = self.upconv1(d2)

d1 = torch.cat([d1, e1], dim=1)

d1 = self.dec1(d1)


return self.final_conv(d1)
```

APPENDIX VI: PYTHON CODE FOR TRAINING THE UNET MODEL

```
import os
import cv2
import glob
import torch
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from tqdm import tqdm
from torch import nn, optim
from torch.utils.data import Dataset, DataLoader, random_split
import torchvision.transforms as T
import torch.nn.functional as F
import albumentations as A
from model import EnhancedUNet  # Modified model architecture

IMG_SIZE = 256  # Increased input resolution
BATCH_SIZE = 16  # Optimized batch size
EPOCHS = 200
LEARNING_RATE = 0.001  # Higher initial learning rate
PATIENCE = 10  # For early stopping
CHECKPOINT_DIR = "/media/ubuntu-
user/KESU/Intelligent_Medical_System/System/pth"
FRAME_DIR = "/media/ubuntu-
user/KESU/Intelligent_Medical_System/System/dataPrep/json_et_png/frameVid"
MASK_DIR = "/media/ubuntu-
user/KESU/Intelligent_Medical_System/System/dataPrep/json_et_png/maskVid"

os.makedirs(CHECKPOINT_DIR, exist_ok=True)

class MedicalDataset(Dataset):
    def __init__(self, frame_dir, mask_dir, transform=None):
        self.frame_files = sorted(glob.glob(os.path.join(frame_dir, "*.avi")))
        self.mask_files = sorted(glob.glob(os.path.join(mask_dir, "*.avi")))
        self.transform = transform
        self.samples = []
        for f_path, m_path in zip(self.frame_files, self.mask_files):
            cap = cv2.VideoCapture(f_path)
            frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
            cap.release()
            self.samples.extend([(f_path, m_path, i) for i in range(frame_count)])

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, idx):
        f_path, m_path, frame_idx = self.samples[idx]

        cap = cv2.VideoCapture(f_path)
        cap.set(cv2.CAP_PROP_POS_FRAMES, frame_idx)
        ret, frame = cap.read()
        if not ret:
            raise ValueError(f"Failed to read frame {frame_idx} from {f_path}")
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        cap.release()

        cap = cv2.VideoCapture(m_path)
        cap.set(cv2.CAP_PROP_POS_FRAMES, frame_idx)
        ret, mask = cap.read()
        if not ret:
            raise ValueError(f"Failed to read mask {frame_idx} from {m_path}")
        mask = cv2.cvtColor(mask, cv2.COLOR_BGR2GRAY)
        cap.release()

        if self.transform:
            transformed = self.transform(image=frame, mask=mask)
            frame = transformed["image"]
            mask = transformed["mask"]

        mask = (mask > 127).float()
        mask = mask.unsqueeze(0)

        return frame, mask

class EnhancedLoss(nn.Module):
    def __init__(self, dice_weight=1.0, focal_weight=1.0, gamma=2):
        super().__init__()
        self.dice_weight = dice_weight
        self.focal_weight = focal_weight
        self.gamma = gamma

    def forward(self, inputs, targets):
        if targets.shape[1] != 1:
            targets = targets[:, :1, :, :]

        probs = torch.sigmoid(inputs)

        intersection = (probs * targets).sum()
        dice_loss = 1 - (2. * intersection + 1e-6) / (probs.sum() + targets.sum() + 1e-6)

        bce = F.binary_cross_entropy_with_logits(inputs, targets, reduction='none')
        p_t = torch.exp(-bce)
        focal_loss = ((1 - p_t) ** self.gamma * bce).mean()

        return self.dice_weight * dice_loss + self.focal_weight * focal_loss

def main():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    transform = A.Compose([
        A.Resize(IMG_SIZE, IMG_SIZE),
        A.OneOf([
            A.HorizontalFlip(p=0.5),
            A.VerticalFlip(p=0.5),
            A.RandomRotate90(p=0.5)
        ], p=0.8),
        A.GaussianBlur(p=0.3),
        A.RandomBrightnessContrast(p=0.3),
        A.CoarseDropout(
            max_holes=8,
            max_height=16,
            max_width=16,
            min_holes=4,
            fill_value=0,
            p=0.5
        ),
        A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
        A.ToTensorV2()
    ])

    full_dataset = MedicalDataset(FRAME_DIR, MASK_DIR, transform=transform)
    train_size = int(0.9 * len(full_dataset))
    val_size = len(full_dataset) - train_size
    train_set, val_set = random_split(full_dataset, [train_size, val_size])

    train_loader = DataLoader(train_set, batch_size=BATCH_SIZE,
                              shuffle=True, pin_memory=True, num_workers=4)
    val_loader = DataLoader(val_set, batch_size=BATCH_SIZE,
                            pin_memory=True, num_workers=2)

    model = EnhancedUNet(in_channels=3).to(device)
    optimizer = optim.AdamW(model.parameters(), lr=1e-4, weight_decay=1e-5)
    scheduler = optim.lr_scheduler.OneCycleLR(
        optimizer,
        max_lr=LEARNING_RATE,
        total_steps=EPOCHS * len(train_loader),
        pct_start=0.2
    )
    # optim.lr_scheduler.CosineAnnealingWarmRestarts(optimizer,
    #     T_0=10,
    #     T_mult=2
    # )

    torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
    criterion = EnhancedLoss(dice_weight=1.0, focal_weight=0.5)
    early_stopping_counter = 0
    best_iou = 0.0

    for epoch in range(EPOCHS):
        model.train()
        epoch_loss = 0.0
        progress_bar = tqdm(train_loader, desc=f"Epoch {epoch + 1}/{EPOCHS}")

        for x, y in progress_bar:
            x, y = x.to(device), y.to(device)
            optimizer.zero_grad()

            pred = model(x)
            loss = criterion(pred, y)
            loss.backward()

            torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
            optimizer.step()

            epoch_loss += loss.item()
            progress_bar.set_postfix({"Loss": f"{loss.item():.4f}"})

        model.eval()
        val_loss = 0.0
        total_intersection = 0
        total_union = 0
        with torch.no_grad():
            for x, y in val_loader:
                x, y = x.to(device), y.to(device)
                pred = model(x)
                val_loss += criterion(pred, y).item()

                pred_mask = (torch.sigmoid(pred) > 0.5)
                y_mask = (y > 0.5)

                total_intersection += (pred_mask & y_mask).sum().item()
                total_union += (pred_mask | y_mask).sum().item()

        avg_loss = epoch_loss / len(train_loader)
        val_loss = val_loss / len(val_loader)
        iou = total_intersection / (total_union + 1e-7)

        scheduler.step(iou)

        print(f"Epoch {epoch + 1} | Train Loss: {avg_loss:.4f} | Val Loss: {val_loss:.4f} | IoU: {iou:.4f}")

        if iou > best_iou:
            best_iou = iou
            prev_best = glob.glob(os.path.join(CHECKPOINT_DIR, "best_model.pth"))
            for pb in prev_best:
                os.remove(pb)
            torch.save(model.state_dict(), os.path.join(CHECKPOINT_DIR, "best_iou_model.pth"))
```

```
            print(f"New best model saved with IoU: {best_iou:.4f}")

    print("Training completed successfully!")

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

APPENDIX VII: PYTHON CODE FOR VALIDATING UNET MODEL

```
import os
import cv2
import torch
import numpy as np
from albumentations import Compose, Resize, Normalize
from albumentations.pytorch import ToTensorV2
from model import EnhancedUNet

TEST_VIDEO_DIR = "/media/ubuntu-user/KESU/Intelligent_Medical_System/System/validVid/"
MASK_DIR = "/media/ubuntu-user/KESU/Intelligent_Medical_System/System/dataPrep/json_et_png/maskVid"
OUTPUT_DIR = "/media/ubuntu-user/KESU/Intelligent_Medical_System/System/infRes/unet"
CHECKPOINT_PATH = "/media/ubuntu-user/KESU/Intelligent_Medical_System/System/pth/best_iou_model.pth"
CLASS_SUBDIRS = ['Healthy', 'Polyp', 'GERD']
IMG_SIZE = 256
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")

os.makedirs(OUTPUT_DIR, exist_ok=True)

transform = Compose([
    Resize(IMG_SIZE, IMG_SIZE),
    Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

def process_video(input_path, output_path, model):
    cap = cv2.VideoCapture(input_path)
    if not cap.isOpened():
        print(f"无法打开视频: {input_path}")
        return

    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = cap.get(cv2.CAP_PROP_FPS)
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

    video_name = os.path.basename(input_path)
    mask_path = None
    for cls in CLASS_SUBDIRS:
        possible_path = os.path.join(MASK_DIR, cls, video_name)
        if os.path.exists(possible_path):
            mask_path = possible_path
            break

    mask_cap = cv2.VideoCapture(mask_path) if mask_path else None

    with torch.no_grad():
        while cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                break

            orig_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            transformed = transform(image=orig_image)
            img_tensor = transformed["image"].unsqueeze(0).to(DEVICE)

            pred = model(img_tensor)
            pred_mask = torch.sigmoid(pred).squeeze().cpu().numpy()
            pred_mask = (pred_mask > 0.5).astype(np.uint8)
            pred_mask = cv2.resize(pred_mask, (width, height),
interpolation=cv2.INTER_NEAREST)

            vis_image = cv2.cvtColor(orig_image, cv2.COLOR_RGB2BGR)

            contours, _ = cv2.findContours(pred_mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
            cv2.drawContours(vis_image, contours, -1, (0, 255, 0), 3)

            if mask_cap and mask_cap.isOpened():
                m_ret, mask_frame = mask_cap.read()
                if m_ret:
                    gray_mask = cv2.cvtColor(mask_frame, cv2.COLOR_BGR2GRAY)
                    gray_mask = cv2.resize(gray_mask, (width, height),
interpolation=cv2.INTER_NEAREST)
                    _, gt_mask = cv2.threshold(gray_mask, 127, 255, cv2.THRESH_BINARY)
                    gt_contours, _ = cv2.findContours(gt_mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
                    cv2.drawContours(vis_image, gt_contours, -1, (0, 0, 255), 2)

            out.write(vis_image)

    cap.release()
    if mask_cap: mask_cap.release()
    out.release()
    print(f"已保存结果视频: {output_path}")

def main():
    model = EnhancedUNet(in_channels=3).to(DEVICE)
    model.load_state_dict(torch.load(CHECKPOINT_PATH, map_location=DEVICE))
    model.eval()

    for root, dirs, files in os.walk(TEST_VIDEO_DIR):
        for file in files:
            if file.endswith(('.avi', '.mp4')):
                input_path = os.path.join(root, file)

                rel_path = os.path.relpath(input_path, TEST_VIDEO_DIR)
                rel_path = os.path.splitext(rel_path)[0] + '.mp4'
                output_path = os.path.join(OUTPUT_DIR, rel_path)
                os.makedirs(os.path.dirname(output_path), exist_ok=True)

                process_video(input_path, output_path, model)

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

APPENDIX VIII: PYTHON CODE FOR SAMCLASS MODEL

```
import torch
import torch.nn as nn
from segment_anything import sam_model_registry


class ResidualBlock(nn.Module):
    def __init__(self, channels):
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(channels, channels, 3, padding=1),
            nn.BatchNorm2d(channels),
            nn.ReLU(),
            nn.Conv2d(channels, channels, 3, padding=1),
            nn.BatchNorm2d(channels)
        )

    def forward(self, x):
        return x + self.conv(x)


class SAMClass(nn.Module):
    def __init__(self, num_classes=5, checkpoint="sam_vit_b_01ec64.pth"):
        super().__init__()
        self.sam = sam_model_registry['vit_b'](
            checkpoint='/media/ubuntu-user/KESU/Intelligent_Medical_System/System/pth/sam_vit_b_01ec64.pth'
        )

        for name, param in self.sam.named_parameters():
            if 'image_encoder' in name:
                if 'blocks.6' in name or 'blocks.7' in name or 'blocks.8' in name or 'blocks.9' in name or 'blocks.10' in name or 'blocks.11' in name:
                    param.requires_grad = True
                else:
                    param.requires_grad = False

        self.decoder = nn.Sequential(
            nn.Conv2d(256, 256, 3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.Upsample(scale_factor=2, mode='bilinear', align_corners=False),

            nn.Conv2d(256, 128, 3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.Upsample(scale_factor=2, mode='bilinear', align_corners=False),

            nn.Conv2d(128, 64, 3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.Upsample(scale_factor=2, mode='bilinear', align_corners=False),

            nn.Conv2d(64, 32, 3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.Upsample(scale_factor=2, mode='bilinear', align_corners=False),

            nn.Conv2d(32, 1, 1)
        )

        self.cls_head = nn.Sequential(
            nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Linear(256, 512),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.6),
            nn.Linear(512, num_classes)
        )

    def forward(self, x):
        image_embedding = self.sam.image_encoder(x)
        seg_features = self.decoder(image_embedding)
```

```
cls_pred = self.cls_head(image_embedding)

return seg_features, cls_pred
```

APPENDIX IX: PYTHON CODE FOR TRAINING SAMCLASS MODEL

```
 import os
import cv2
import glob
import torch
import numpy as np
from PIL import Image
from tqdm import tqdm
from torch import nn, optim
from torch.utils.data import Dataset, DataLoader, random_split,
WeightedRandomSampler
import torchvision.transforms as T
import albumentations as A
import torch.nn.functional as F
import decord
from sam_model import SAMClass


IMG_SIZE = 1024
BATCH_SIZE = 2
EPOCHS = 200
#LEARNING_RATE = 0.001
CHECKPOINT_DIR = "/media/ubuntu-
user/KESU/Intelligent_Medical_System/System/pth"
```

```
FRAME_DIR = "/media/ubuntu-user/KESU/Intelligent_Medical_System/System/dataPrep/json_et_png/frameVid"

MASK_DIR = "/media/ubuntu-user/KESU/Intelligent_Medical_System/System/dataPrep/json_et_png/maskVid"

CLASS_NAMES = ['Healthy', 'Polyp', 'GERD']

LOG_FILE = "/media/ubuntu-user/KESU/Intelligent_Medical_System/System/sam_200_1e-5.txt"



class FocalLoss(nn.Module):
    def __init__(self, weight=None, gamma=2):
        super().__init__()
        self.weight = weight
        self.gamma = gamma

    def forward(self, inputs, targets):
        weight = self.weight.to(inputs.device) if self.weight is not None else None
        ce_loss = F.cross_entropy(inputs, targets, reduction='none', weight=weight)
        pt = torch.exp(-ce_loss)
        return ((1 - pt) ** self.gamma * ce_loss).mean()


class MedicalDataset(Dataset):
    def __init__(self, frame_dir, mask_dir, transform=None):
        self.samples = []
        self.class_counts = np.zeros(len(CLASS_NAMES))
        self.class_to_idx = {cls: i for i, cls in enumerate(CLASS_NAMES)}
self.transform = transform
self.vr_cache = {}

for class_name in CLASS_NAMES:
    class_idx = self.class_to_idx[class_name]
    frame_class_dir = os.path.join(frame_dir, class_name)
    mask_class_dir = os.path.join(mask_dir, class_name)

    if not os.path.exists(frame_class_dir) or not os.path.exists(mask_class_dir):
        continue

    video_pairs = []
    for vid_name in os.listdir(frame_class_dir):
        frame_vid = os.path.join(frame_class_dir, vid_name)
        mask_vid = os.path.join(mask_class_dir, vid_name)

        if os.path.exists(mask_vid):
            video_pairs.append((frame_vid, mask_vid))

    for f_vid, m_vid in video_pairs:
        vr = decord.VideoReader(f_vid)
        num_frames = len(vr)
        self.samples.extend([(f_vid, m_vid, i, class_idx) for i in range(num_frames)])
        self.class_counts[class_idx] += num_frames
    print(f"Class distribution: {self.class_counts}")

  def __len__(self):
    return len(self.samples)

  def __getitem__(self, idx):
    f_path, m_path, frame_idx, cls_idx = self.samples[idx]

    if f_path not in self.vr_cache:
      self.vr_cache[f_path] = decord.VideoReader(f_path)
    frame = self.vr_cache[f_path][frame_idx].asnumpy()

    if m_path not in self.vr_cache:
      self.vr_cache[m_path] = decord.VideoReader(m_path)
    mask = self.vr_cache[m_path][frame_idx].asnumpy()
    mask = cv2.cvtColor(mask, cv2.COLOR_RGB2GRAY)
    mask = (mask > 127).astype(np.float32)

    if self.transform:
      transformed = self.transform(image=frame, mask=mask)
      frame = transformed["image"]
      mask = transformed["mask"]
```

```
        return frame, mask.unsqueeze(0), torch.tensor(cls_idx)


class MultiTaskLoss(nn.Module):
    def __init__(self, class_counts):
        super().__init__()
        self.class_weights = 1.0 / (torch.sqrt(torch.tensor(class_counts).float() + 1e-6))
        self.class_weights = self.class_weights / self.class_weights.sum()

        self.focal_loss = FocalLoss(weight=self.class_weights, gamma=2)

    def forward(self, seg_pred, cls_pred, seg_target, cls_target):
        seg_pred = seg_pred.squeeze(1)
        seg_target = seg_target.squeeze(1).float()

        with torch.no_grad():
            edges = F.max_pool2d(seg_target, 3, 1, 1) - F.avg_pool2d(seg_target, 3, 1, 1)
            edge_weight = 1.0 + 2.0 * (edges > 0.1).float()

        sum_target = (seg_target * edge_weight).sum()
        sum_pred = (seg_pred.sigmoid() * edge_weight).sum()
        intersection = (seg_pred.sigmoid() * seg_target * edge_weight).sum()

        dice = (2. * intersection + 1e-6) / (sum_pred + sum_target + 1e-6)
        dice = torch.where(sum_target > 0, dice, 1.0 - (sum_pred > 0).float())
        dice_loss = 1 - dice.mean()

        pos_weight = torch.exp(-5 * seg_target.mean())
        focal_loss = F.binary_cross_entropy_with_logits(
            seg_pred, seg_target,
            reduction='mean',
            pos_weight=pos_weight + 1.0
        )

        cls_loss = self.focal_loss(cls_pred, cls_target)

        return 1.0 * dice_loss + 1.0 * focal_loss + 3.0 * cls_loss

def main():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    transform = A.Compose([
        A.Resize(1024, 1024),
        A.HorizontalFlip(p=0.5),
        A.VerticalFlip(p=0.5),
        A.Rotate(limit=30, p=0.5),
        A.ElasticTransform(
            sigma=15,
            alpha=30,
            p=0.5
        ),
        A.OneOf([
            A.CoarseDropout(
                max_holes=8,
                max_height=32,
                max_width=32,
                fill_value=180,
                p=0.5
            ),
        ], p=0.7),
        A.GridDistortion(p=0.3),
        A.RandomBrightnessContrast(p=0.4),
        A.CLAHE(p=0.3),
        A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
        A.ToTensorV2()
    ], is_check_shapes=False)


    full_dataset = MedicalDataset(FRAME_DIR, MASK_DIR, transform=transform)


    train_size = int(0.9 * len(full_dataset))
    val_size = len(full_dataset) - train_size
    train_set, val_set = random_split(full_dataset, [train_size, val_size])
```

```
    class_weights = 1.0 / (full_dataset.class_counts + 1e-6)
    sample_weights = class_weights[[s[3] for s in full_dataset.samples]]
    train_sampler = WeightedRandomSampler(sample_weights[:len(train_set)], len(train_set))

    train_loader = DataLoader(
        train_set, batch_size=BATCH_SIZE,
        sampler=train_sampler,
        num_workers=4, pin_memory=True, drop_last=True
    )
    val_loader = DataLoader(val_set, batch_size=BATCH_SIZE, num_workers=2, drop_last=True)

    model = SAMClass(num_classes=3).to(device)

    optimizer = optim.AdamW([
        {'params': model.decoder.parameters(), 'lr': 0.001},
        {'params': model.cls_head.parameters(), 'lr': 0.001},
        {'params': model.sam.image_encoder.parameters(), 'lr': 0.001 / 5}
    ], weight_decay=1e-4)

    criterion = MultiTaskLoss(full_dataset.class_counts)
    scheduler = optim.lr_scheduler.CosineAnnealingWarmRestarts(
        optimizer,
        T_0=15,
    T_mult=2,
    eta_min=1e-6,
    last_epoch=-1
)
scaler = torch.amp.GradScaler()

best_iou = 0.0
for epoch in range(EPOCHS):
    model.train()
    epoch_loss = 0.0
    for x, y, cls in tqdm(train_loader, desc=f"Epoch {epoch + 1}/{EPOCHS}"):
        x, y, cls = x.to(device), y.to(device), cls.to(device)

        optimizer.zero_grad()

        with torch.amp.autocast(device_type='cuda', dtype=torch.float16):
            seg_pred, cls_pred = model(x)
            loss = criterion(seg_pred, cls_pred, y, cls)

        scaler.scale(loss).backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        scaler.step(optimizer)
        scaler.update()
        epoch_loss += loss.item()

    model.eval()
    val_loss = 0.0
    class_intersection = torch.zeros(len(CLASS_NAMES), device=device)
    class_union = torch.zeros(len(CLASS_NAMES), device=device)
    cls_correct = torch.zeros(len(CLASS_NAMES), device=device)
    cls_total = torch.zeros(len(CLASS_NAMES), device=device)

    with torch.no_grad():
        for x, y, cls in val_loader:
            x, y, cls = x.to(device), y.to(device), cls.to(device)

            seg_pred, cls_pred = model(x)
            val_loss += criterion(seg_pred, cls_pred, y, cls).item()

            seg_pred = F.interpolate(seg_pred, size=(1024, 1024), mode='bilinear')

            _, predicted = torch.max(cls_pred, 1)
            pred_mask = (seg_pred.sigmoid() > 0.5).float()
            target_mask = y.float()

            for c in range(len(CLASS_NAMES)):
                class_mask = (cls == c)
```

```
          if class_mask.any():
            c_pred = pred_mask[class_mask]
            c_target = target_mask[class_mask]

            if c_target.sum() == 0:
              correct = (c_pred.sum() == 0).float()
              class_intersection[c] += correct * c_target.numel()
              class_union[c] += c_target.numel()
            else:
              intersection = (c_pred * c_target).sum()
              union = (c_pred + c_target).sum() - intersection
              class_intersection[c] += intersection
              class_union[c] += union

    iou_per_class = class_intersection / (class_union + 1e-7)
    dice_per_class = (2 * class_intersection) / (class_intersection + class_union + 1e-7)
    cls_accuracy = cls_correct / (cls_total + 1e-7)
    mean_iou = iou_per_class.mean()

    train_loss = epoch_loss / len(train_loader)
    val_loss = val_loss / len(val_loader)
    scheduler.step(mean_iou)

# for saving training output
```

```
    log_str = f"\nEpoch {epoch + 1}/{EPOCHS}\n"
    log_str += f"Train Loss: {train_loss:.4f} | Val Loss: {val_loss:.4f}\n"
    log_str += f"Val mIoU: {mean_iou:.4f} | Val mDice: {dice_per_class.mean():.4f}\n"
    log_str += "\nClass-wise Metrics:\n"
     for c, name in enumerate(CLASS_NAMES):
        log_str += f"{name}:\n"
        log_str += f"  IoU: {iou_per_class[c]:.4f} | Dice: {dice_per_class[c]:.4f}\n"
        log_str += f"  Cls Acc: {cls_accuracy[c]:.4f} | Samples: {cls_total[c]:.0f}\n"

    print(log_str)
    with open(LOG_FILE, 'a') as f:
        f.write(log_str)

    if mean_iou > best_iou:
        best_iou = mean_iou
        torch.save(model.state_dict(), os.path.join(CHECKPOINT_DIR, "sam_best_other.pth"))
        #print(f"New best model saved with mIoU: {mean_iou:.4f}")

        best_msg = f"New best model saved with mIoU: {mean_iou:.4f}\n"
        print(best_msg)
        with open(LOG_FILE, 'a') as f:
            f.write(best_msg)

    print(f"\nEpoch {epoch + 1}/{EPOCHS}")
```

```
        print(f"Train Loss: {train_loss:.4f} | Val Loss: {val_loss:.4f}")
        print(f"Val mIoU: {mean_iou:.4f} | Val mDice: {dice_per_class.mean():.4f}")

        print("\nClass-wise Metrics:")
        for c, name in enumerate(CLASS_NAMES):
            print(f"{name}:")
            print(f"  IoU: {iou_per_class[c]:.4f} | Dice: {dice_per_class[c]:.4f}")
            print(f"  Cls Acc: {cls_accuracy[c]:.4f} | Samples: {cls_total[c]:.0f}")

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

APPENDIX X: PYTHON CODE FOR VALIDATING SAMCLASS MODEL

```
import os
import glob
import decord
import numpy as np
import cv2
import torch
import torch.nn.functional as F
from albumentations import Compose, Resize, Normalize, ToTensorV2
from sam_model import SAMClass


TEST_VIDEO_DIR = "/media/ubuntu-user/KESU/Intelligent_Medical_System/System/validVid/"
OUTPUT_DIR = "/media/ubuntu-user/KESU/Intelligent_Medical_System/System/infRes/sam/unknown"
CHECKPOINT_PATH = "/media/ubuntu-user/KESU/Intelligent_Medical_System/System/pth/sam_best_1e-5_200.pth"
MASK_DIR = "/media/ubuntu-user/KESU/Intelligent_Medical_System/System/dataPrep/json_et_png/maskVid"
FRAME_DIR = "/media/ubuntu-user/KESU/Intelligent_Medical_System/System/dataPrep/json_et_png/frameVid"
IMG_SIZE = 1024
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
CLASS_SUBDIRS = ['Healthy', 'Polyp', 'GERD']
```

```
transform = Compose([
    Resize(IMG_SIZE, IMG_SIZE),
    Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

model = SAMClass(num_classes=3)
model.load_state_dict(torch.load(CHECKPOINT_PATH, map_location=device))
model.to(device)
model.eval()

os.makedirs(OUTPUT_DIR, exist_ok=True)

for video_path in glob.glob(os.path.join(TEST_VIDEO_DIR, '**', '*.avi'), recursive=True):
    rel_path = os.path.relpath(video_path, TEST_VIDEO_DIR)
    output_path = os.path.join(OUTPUT_DIR, rel_path)
    os.makedirs(os.path.dirname(output_path), exist_ok=True)

    video_name = os.path.basename(video_path)
    mask_reader = None

    frame_found = any(os.path.exists(os.path.join(FRAME_DIR, cls, video_name)) for cls in CLASS_SUBDIRS)
    mask_matches = []

    if frame_found:
        for cls in CLASS_SUBDIRS:
            mask_path = os.path.join(MASK_DIR, cls, video_name)
            if os.path.exists(mask_path):
                mask_matches.append(mask_path)
                break

    if mask_matches:
        try:
            mask_reader = decord.VideoReader(mask_matches[0])
            print(f"Found matching mask video in class subdirectory: {mask_matches[0]}")
        except Exception as e:
            print(f"Error loading mask video: {e}")
            mask_reader = None

    vr = decord.VideoReader(video_path)
    fps = vr.get_avg_fps()

    fourcc = cv2.VideoWriter_fourcc(*'XVID')
    out = cv2.VideoWriter(output_path, fourcc, fps, (IMG_SIZE, IMG_SIZE), isColor=True)

    for i in range(len(vr)):
```

```
    frame = vr[i].asnumpy()

    transformed = transform(image=frame)
    img_tensor = transformed["image"].unsqueeze(0).to(device)

    with torch.no_grad():
        seg_pred, cls_pred = model(img_tensor)
        seg_pred = F.interpolate(seg_pred, size=(1024, 1024), mode='bilinear', align_corners=False)

    pred_class = torch.argmax(cls_pred, dim=1).item()
    prob_map = seg_pred.sigmoid().squeeze().cpu().numpy()

    mean = torch.tensor([0.485, 0.456, 0.406]).to(device).view(1, 3, 1, 1)
    std = torch.tensor([0.229, 0.224, 0.225]).to(device).view(1, 3, 1, 1)
    denorm_img = img_tensor * std + mean
    denorm_img = denorm_img.squeeze(0).permute(1, 2, 0).cpu().numpy()
    denorm_img = (denorm_img * 255).astype(np.uint8)
    denorm_img_bgr = cv2.cvtColor(denorm_img, cv2.COLOR_RGB2BGR)
    overlay = denorm_img_bgr.copy()
    threshold = 0.3
    binary_mask = (prob_map > threshold).astype(np.uint8)
    contours, _ = cv2.findContours(binary_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cv2.drawContours(overlay, contours, -1, (0, 255, 0), 3)
```

```
        if mask_reader and i < len(mask_reader):
            try:
                mask_frame = mask_reader[i].asnumpy()
                mask_gray = cv2.cvtColor(mask_frame, cv2.COLOR_RGB2GRAY)


                mask_gray = cv2.resize(mask_gray, (IMG_SIZE, IMG_SIZE),
interpolation=cv2.INTER_NEAREST)


                _, gt_mask = cv2.threshold(mask_gray, 127, 255, cv2.THRESH_BINARY)
                gt_contours, _ = cv2.findContours(gt_mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)


                cv2.drawContours(overlay, gt_contours, -1, (0, 0, 255), 3)
                cv2.putText(overlay, "Ground Truth", (20, 120),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)


                y_true = gt_mask.astype(np.float32) / 255.0
                y_pred = binary_mask.astype(np.float32)
                intersection = np.sum(y_true * y_pred)
                dice = (2. * intersection + 1e-6) / (np.sum(y_true) + np.sum(y_pred) + 1e-6)
                cv2.putText(overlay, f"Dice: {dice:.2f}", (20, 200),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
            except Exception as e:
                print(f"Error processing mask frame {i}: {e}")
        cls_prob = F.softmax(cls_pred, dim=1).squeeze().cpu().numpy()

        class_label = f"{CLASS_SUBDIRS[pred_class]} ({cls_prob[pred_class]:.2f})"

        cv2.putText(overlay, class_label, (20, 60), cv2.FONT_HERSHEY_SIMPLEX, 1.5, (0,
255, 0), 3, cv2.LINE_AA)

        cv2.putText(overlay, "Prediction", (20, 160), cv2.FONT_HERSHEY_SIMPLEX, 1,
(0, 255, 0), 2)


        out.write(overlay)


    out.release()

    print(f"Processed: {video_path} -> {output_path}")


print("Inference completed!")
```