Medical Image Computing CAP5516 Assignment#03

Ashmal Vayani, UCF ID: 5669011

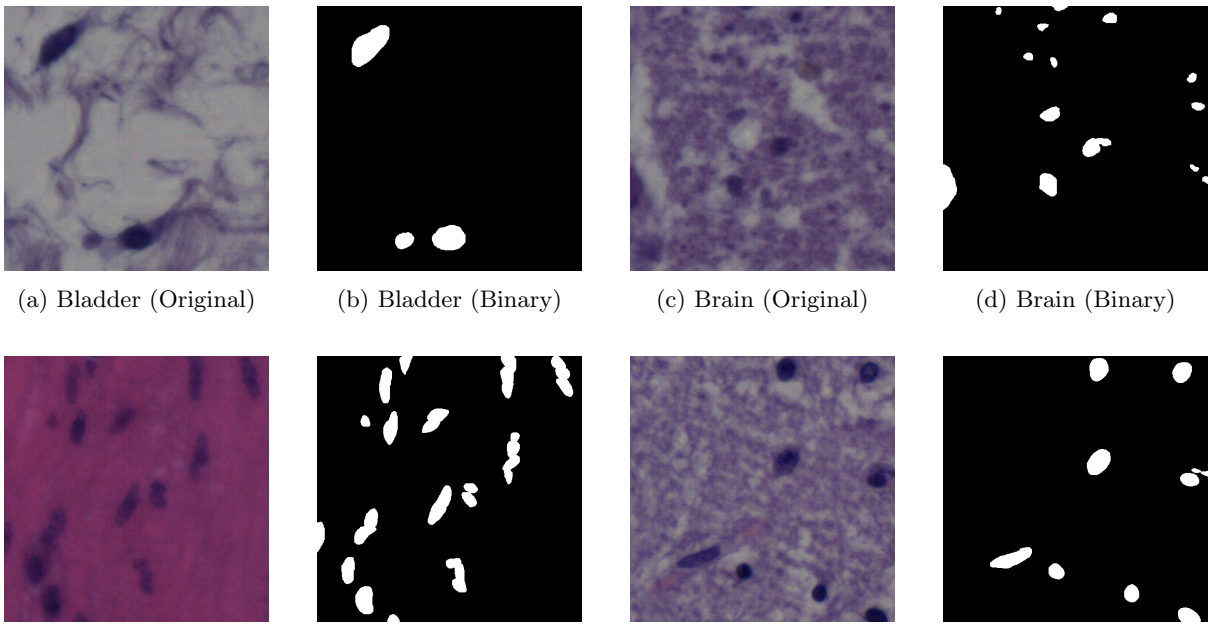
Parameter Efficient Fine-tuning Foundation Model for Nuclei Instance Segmentation

Training Code: The training and implementation code can be accessed **here**.

1 NuInsSeg Dataset

The NuInsSeg dataset is a fully manually annotated nuclei instance segmentation dataset, comprising 665 image tiles with 30,698 segmented nuclei, collected from 31 organs across human and mouse samples. It is one of the few datasets to include ambiguous area masks, marking regions too complex for even expert annotators to label deterministically. All nuclei were segmented manually using ImageJ to avoid bias from semi-automated tools. Additionally, the dataset provides various auxiliary segmentation masks—such as border-removed masks, distance maps, and weighted masks—to facilitate the development of advanced computer vision models. The dataset spans 23 human organs (e.g., liver, kidney, pancreas, cerebrum) and 8 mouse organs (e.g., thymus, spleen, muscle), offering a diverse distribution of nuclei densities (e.g., 236.7 nuclei/image in mouse spleen vs. 5.9 in mouse muscle).

Dataset Visualization



(a) Bladder (Original) (b) Bladder (Binary) (c) Brain (Original) (d) Brain (Binary)

(e) Cardia (Original) (f) Cardia (Binary) (g) Cerebellum (Orig.) (h) Cerebellum (Binary)

Figure 1: Examples of original and binary segmentation images from NuInsSeg.

2 Implementation Details

In this assignment, I referred from the codebase of Finetune-SAM repository and made adjustments as the code wasn't directly compatible with our use case. We used a SAM-Base architecture variant

Table 1: Summary of NuInsSeg dataset across selected human and mouse organs

Organ	Type	# Images	# Nuclei (Avg. per image)
Kidney	Human	11	1,222 (111.1)
Salivary gland	Human	44	3,129 (71.1)
Spleen	Human	34	3,286 (96.7)
Pancreas	Human	44	2,178 (49.5)
Brain (Cerebellum)	Human	12	549 (45.8)
Muscle	Human	9	127 (14.1)
Bone (Femur)	Mouse	6	757 (126.2)
Spleen	Mouse	7	1,657 (236.7)
Thymus	Mouse	6	1,342 (223.7)
Muscle	Mouse	28	165 (5.9)
Total (Human)		**472**	**23,247 (49.3)**
Total (Mouse)		**193**	**7,451 (38.6)**
Overall Total		**665**	**30,698 (46.2)**

and done data processing before starting the training. Next, we describe the detailed implementation approach that was used

2.1 Dataset Construction and Preprocessing

The dataset is managed through a custom Public_dataset class, which inherits from torch.utils.data.Dataset. This class is designed to flexibly handle multi-class and binary segmentation tasks, supporting several normalization and prompt strategies. The dataset is constructed using a pair of directories—one for images and another for masks—and a list of corresponding file paths. It offers functionality for filtering out samples with empty masks and can be configured to focus on specific anatomical targets or tissue types. For preprocessing, both intensity-based augmentations (like color jittering and histogram equalization) and spatial augmentations (such as random cropping and rotation) are applied during training. Additionally, two normalization schemes are supported: standard SAM normalization using mean and standard deviation, and MedSAM normalization which scales pixel intensities to [0, 1]. Masks can be converted into binary or multi-class labels depending on the specified task.

2.2 LoRA-Based SAM Model Modification

The SAM model is adapted using Low-Rank Adaptation (LoRA) to enable efficient fine-tuning with fewer trainable parameters. The LoRA_Sam class injects LoRA modules into the image encoder and mask decoder by modifying the QKV attention layers with low-rank matrices. This allows residual updates without changing the full weights. Both self-attention and cross-attention layers in the decoder are supported. Fine-tuning can be selectively applied using flags, and the model integrates seamlessly into the SAM pipeline.

2.3 Training Pipeline and Fine-Tuning Strategy

The training script supports multiple fine-tuning modes, including LoRA. In LoRA mode, only LoRA layers are updated, significantly reducing the number of trainable parameters. The model is trained using a combined Dice and CrossEntropy loss, with learning rate scheduling via warm-up or step decay. Dice coefficient is used for validation, and the best model is saved based on performance. Logging is handled by TensorBoard and wandb, and training stops early if no improvement is seen for 20 epochs.

2.4 Network Architecture

Below is the high-level architecture diagram for LoRA training during fine-tuning stage.

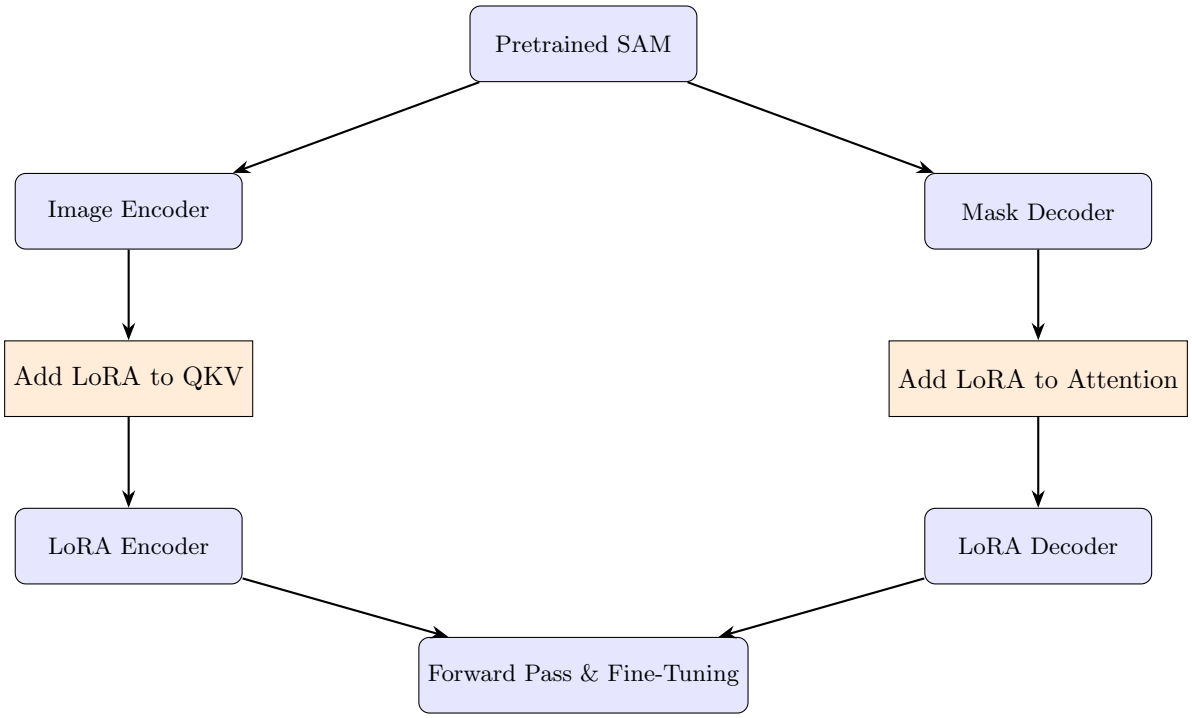


Figure 2: LoRA injection into the SAM model's encoder and decoder. Only LoRA weights are trainable during fine-tuning.

Table 2: Training hyperparameters and experimental configuration for LoRA-based SAM fine-tuning on NuInsSeg.

Training Hyperparameters	**Model, Dataset, and LoRA Configuration**
Batch Size: `10`	Architecture: `vit_b`
Epochs: `500`	Pretrained Checkpoint: `sam_vit_b_01ec64.pth`
Initial Learning Rate: `1e-3`	Dataset Name: `NuInsSeg`
Warm-up: `Enabled`	Segmentation Target: `combine_all` (binary)
Warm-up Period: `200 iterations`	Image Folder: `/datasets`
Loss Function: Dice + CrossEntropy	Train Image List: `train_fold_4.csv`
Optimizer: AdamW	Validation Image List: `val_fold_4.csv`
Scheduler: `StepLR (step=10, gamma=0.5)`	Output Directory: `NuInsSeg_Model_Outputs/lora`
Shuffle Dataset: `True`	Normalization Type: `sam`
Dataloader Workers: `4`	Number of Classes: `2`
GPU: `Enabled`	Input Size: `1024`, Output Size: `256`
	Patch Size: `2`, Dim: `512`, Depth: `64`
	Heads: `16`, MLP Dim: `1024`
LoRA Parameters	
Encoder LoRA: `Enabled`	LoRA Rank: `4`
Decoder LoRA: `Enabled`	Trainable Parameters: `LoRA only`
Encoder Layers: `All (default)`	Decoder Layers: `Full transformer + final attention`

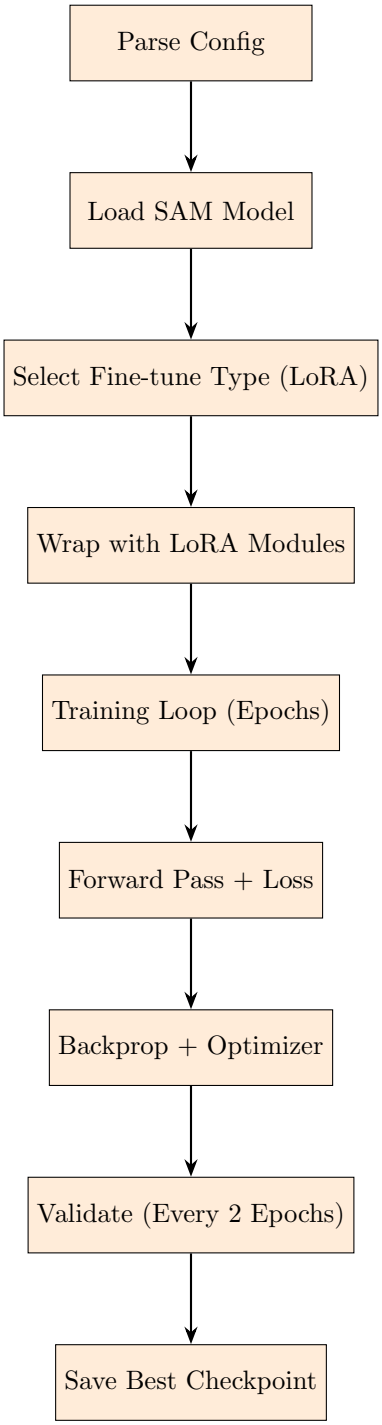


Figure 3: Training pipeline for LoRA-based SAM fine-tuning using hybrid Dice and CE loss.

2.5 Segment Anything Model (SAM) Configuration

We utilize the ViT-B variant of the Segment Anything Model (SAM) as the base architecture for our fine-tuning. The SAM model consists of three main components: an image encoder, a prompt encoder, and a mask decoder. The image encoder is a Vision Transformer (ViT-B) with 12 transformer blocks, an embedding dimension of 768, and 12 attention heads. Global attention is applied at layers 2, 5, 8, and 11. The patch size used is 16, producing a 64×64 image embedding for 1024×1024 input resolution. The encoder uses relative positional encodings and includes LayerNorm normalization with an MLP expansion ratio of 4.

The prompt encoder maps point- or box-based prompts into an embedding space with an output channel size of 256. The mask decoder uses a lightweight two-layer transformer with embedding dimension 256, MLP dimension 2048, and 8 attention heads. It includes an IoU prediction head with depth 3 and hidden dimension 256, and is configured to produce a single mask output per input (i.e., binary segmentation). The model is initialized using the official SAM checkpoint `sam_vit_b_01ec64.pth` and uses standard normalization statistics as described in the original SAM paper.

Table 3: Architecture details of the SAM ViT-B model used in our experiments.

Component	Configuration
Model Variant	SAM ViT-B
Image Encoder	ViT-B (12 layers, 768 dim, 12 heads)
Global Attention Indexes	`[2, 5, 8, 11]`
Patch Size	`16`
Input Image Size	`1024 × 1024`
Embedding Size	`64 × 64`
Prompt Encoder Output Channels	`256`
Mask Decoder	2-layer Transformer
Mask Decoder Heads	`8`
Mask Decoder Embedding Dim	`256`
Mask Decoder MLP Dim	`2048`
IoU Head Depth	`3`
IoU Head Hidden Dim	`256`
Number of Masks Output	`1`
Checkpoint Used	`sam_vit_b_01ec64.pth`
Pixel Mean	`[123.675, 116.28, 103.53]`
Pixel Std	`[58.395, 57.12, 57.375]`

3 Dataset Pre-processing:

3.1 5-fold Evaluation:

To ensure robust model evaluation, we implement a 5-fold cross-validation strategy to split our brain tumor data set.

- **Dataset Splitting**: The data set is divided into five subsets (folds), ensuring diverse training and testing samples.
- **Shuffle for Randomness**: Files are shuffled before splitting to prevent bias.
- **Training & Testing**: Each fold uses 4 subsets for training and 1 for testing, ensuring that all samples contribute to model evaluation.
- **Balanced Evaluation**: Reduce variance and provide a more reliable assessment of model performance.
- **Reproducibility**: A fixed random seed ensures consistent and repeatable splits across different runs.

3.2 Dataset Preprocessing and Augmentation

We implement a customized data preprocessing pipeline using the `Public_dataset` class, designed to handle medical segmentation datasets with flexibility for binary, multi-class, and class-specific settings. The input consists of paired image and mask paths, which are filtered based on relevance to target anatomy and non-empty masks. We support three segmentation target modes: (i) `combine_all` for binary segmentation, (ii) `multi_all` for multi-class, and (iii) class-specific ID matching.

For training, images undergo color augmentations including random histogram equalization and color jitter, and optionally spatial augmentations such as random resized cropping and rotation. All images are resized to a fixed input shape (`image_size`) and normalized using either SAM statistics or min-max scaling for MedSAM. Prompts for point- or box-based interactions can be generated during training or evaluation, enabling compatibility with prompt-based segmentation models. Cropping, padding, and mask class filtering are also handled as part of the pipeline to support clean supervision.

Table 4: Dataset preprocessing configuration used in training.

Parameter	**Value / Description**
Image Folder	`User-specified path`
Mask Folder	`Same as image folder`
Target Modes	`combine_all`, `multi_all`, or specific class ID
Input Image Size	`1024` $\times$ `1024` (resized)
Normalization	`sam` (mean/std) or `medsam` ([0,1] min-max)
Prompt Type	`point`, `box`, or `hybrid`
Crop	Optional random cropping (size = `1024`)
Delete Empty Masks	`True` (filter samples without annotations)
Spatial Augmentations	`RandomResizedCrop`, `RandomRotation`
Color Augmentations	`ColorJitter`, `RandomEqualize`
Label Mapping	Supports loading class IDs from pickle file
Supported Image Formats	PIL-based loading with RGB conversion
Train / Eval Phases	Separate transform configs for each phase

4 Evaluation Metrics

We evaluate the performance of our segmentation model using three primary metrics: the Aggregated Jaccard Index (AJI), Panoptic Quality (PQ), and Dice Loss. Each of these metrics captures different aspects of instance-level and pixel-level accuracy.

1. Aggregated Jaccard Index (AJI) AJI is an instance-aware extension of the Jaccard Index (IoU), which evaluates how well predicted instances align with ground truth instances. For each ground truth instance, it matches the predicted instance with the highest IoU (if any), then aggregates the intersection and union areas of all matched and unmatched instances. The AJI is defined as:

$$\text{AJI} = \frac{\sum_{i \in M} |G_i \cap P_i|}{\sum_{i \in M} |G_i \cup P_i| + \sum_{j \in U_G} |G_j| + \sum_{k \in U_P} |P_k|} \tag{1}$$

where:

- M is the set of matched ground truth and predicted instance pairs.
- U_G and U_P are unmatched ground truth and predicted instances.
- G_i and P_i denote ground truth and predicted masks.

2. Panoptic Quality (PQ) PQ combines segmentation quality and detection quality in a single metric. It is computed as the product of **Detection Quality (DQ)** and **Segmentation Quality (SQ)**, where matched instances have an IoU above a given threshold (typically 0.5):

$$\text{DQ} = \frac{TP}{TP + 0.5 \cdot FP + 0.5 \cdot FN}, \quad \text{SQ} = \frac{1}{TP} \sum_{(G_i, P_i) \in TP} \text{IoU}(G_i, P_i) \tag{2}$$

$$\text{PQ} = \text{DQ} \cdot \text{SQ} \tag{3}$$

where:

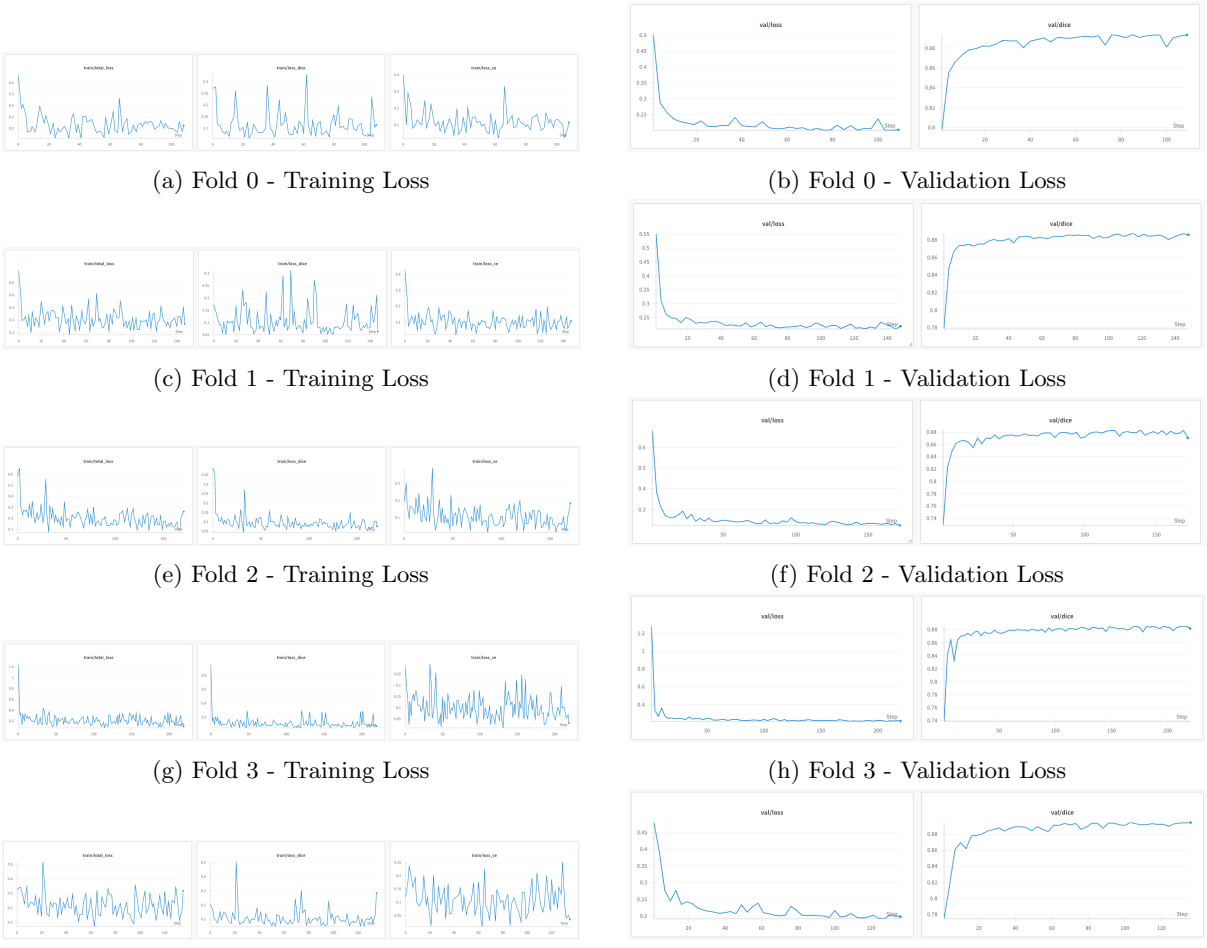
- TP, FP, and FN are the number of true positives, false positives, and false negatives.
- (G_i, P_i) are matched ground truth and predicted instance pairs.

3. Dice Loss The Dice coefficient measures pixel-wise overlap between the predicted mask and the ground truth. Dice Loss is derived by subtracting the Dice coefficient from 1 and is commonly used as a loss function for segmentation tasks:

$$\text{Dice}(G, P) = \frac{2 \cdot |G \cap P|}{|G| + |P|}, \quad \text{Dice Loss} = 1 - \text{Dice}(G, P) \tag{4}$$

where G is the ground truth mask and P is the predicted mask.

5 Training/ Fine-Tuning (Train-Val Loss Curves)



(a) Fold 0 - Training Loss

(b) Fold 0 - Validation Loss

(c) Fold 1 - Training Loss

(d) Fold 1 - Validation Loss

(e) Fold 2 - Training Loss

(f) Fold 2 - Validation Loss

(g) Fold 3 - Training Loss

(h) Fold 3 - Validation Loss

(i) Fold 4 - Training Loss

(j) Fold 4 - Validation Loss

Figure 4: Training and validation losses across all 5 folds on 500 epochs. Please note that there was early stopping applied in the training so it may have stopped earlier than 500 epochs.

6 Evaluation Results

Table 5: Fold-wise segmentation results and average scores for our LoRA-SAM model.

Fold	Dice Score (%)	AJI Score (%)	PQ Score (%)
Fold 0	89.00	69.64	66.89
Fold 1	89.38	70.18	67.94
Fold 2	88.85	69.19	66.39
Fold 3	89.64	70.62	68.50
Fold 4	88.67	68.90	66.03
Average	**89.11**	**69.71**	**67.15**

Table 6: Comparison of segmentation performance with existing U-Net variants. All values are reported as percentages.

Model	# Parameters	Avg. Dice (%)	Avg. AJI / PQ (%)
Shallow U-Net	1.9 million	78.8	50.5 / 42.7
Deep U-Net	7.7 million	79.7	49.4 / 40.4
Attention U-Net	2.3 million	80.5	45.7 / 36.4
Residual Attention U-Net	2.4 million	81.4	46.2 / 36.9
Two-stage U-Net	3.9 million	76.6	52.8 / 47.2
Dual decoder U-Net	3.5 million	79.4	55.9 / 51.3
Ours (LoRA-SAM)	**705k trainable**	**89.1**	**69.7 / 67.1**

As shown in Table 5, our LoRA-SAM model achieved consistently high performance across all five cross-validation folds. The average Dice score was **89.11%**, indicating strong pixel-wise agreement between the predicted and ground truth masks. Instance-level metrics also showed robust performance, with an average Aggregated Jaccard Index (AJI) of **69.71%** and a Panoptic Quality (PQ) score of **67.15%**, demonstrating the model's effectiveness in both segmentation accuracy and instance delineation.

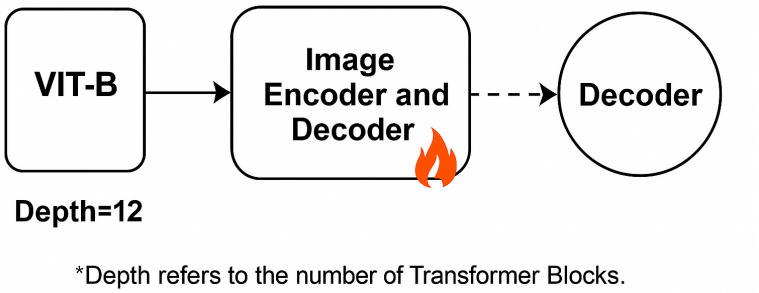
In comparison with existing U-Net variants, as presented in Table 6, our LoRA-SAM model outperforms all baselines across all three metrics. Despite using only **705k trainable parameters**, our model surpasses even the most complex U-Net designs, including the Dual Decoder U-Net, which achieved a Dice score of 79.4% and a PQ score of 51.3%. Notably, our method achieves a relative improvement of **+9.7% in Dice**, **+13.8% in AJI**, and **+15.8% in PQ** compared to the best-performing U-Net variant. These results highlight the efficiency and effectiveness of LoRA-based fine-tuning when applied to high-capacity foundation models like SAM.

7 LoRA - Number of Tunable Parameters

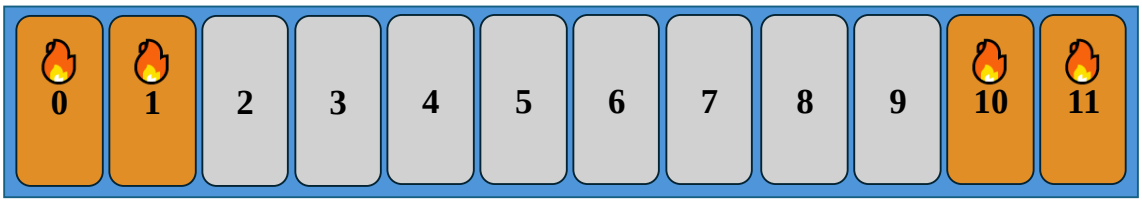
Table 7 summarizes the distribution of trainable parameters across the major components of our LoRA-SAM model. Notably, only a small fraction of the full SAM model is fine-tuned—specifically, LoRA-injected layers in the image encoder and mask decoder, as well as the prompt encoder. The image encoder contributes just 49k parameters, while the prompt encoder accounts for approximately 6.2k. The majority of trainable parameters (over 650k) reside in the mask decoder, particularly within the output upscaling and hypernetwork MLP modules. Overall, our fine-tuning involves only **705,679 parameters**, making the approach highly parameter-efficient while preserving strong segmentation performance.

Table 7: Breakdown of trainable parameters in LoRA-SAM fine-tuning.

Component / Layer	Trainable Parameters
Image Encoder	
Blocks [0, 1, 10, 11] – LoRA-Q and LoRA-V (QKV)	49,152
Prompt Encoder	
Point Embeddings (4 + not_a_point + no_mask)	1,536
Mask Downscaling Layers	4,684
Subtotal – Prompt Encoder	6,220
Mask Decoder	
LoRA – Self-Attn + Cross-Attn (2 Layers + Final)	24,576
Token Embeddings (iou_token, mask_tokens)	1,024
Output Upscaling Convs (All layers)	74,048
Hypernetwork MLPs (3 heads × 3 layers)	410,880
IoU Prediction Head (3 layers)	139,779
Subtotal – Mask Decoder	650,307
Total Trainable Parameters	**705,679**



(a) ViT Base architecture used for assignment purposes.



(b) LoRA layers integrated into the ViT Base model.

Figure 5: Visual comparison of the ViT Base model and its LoRA-integrated variant.

8 Prediction Visualization

Below, I have added a few qualitative examples for the model's prediction, ground truth, and the original image.

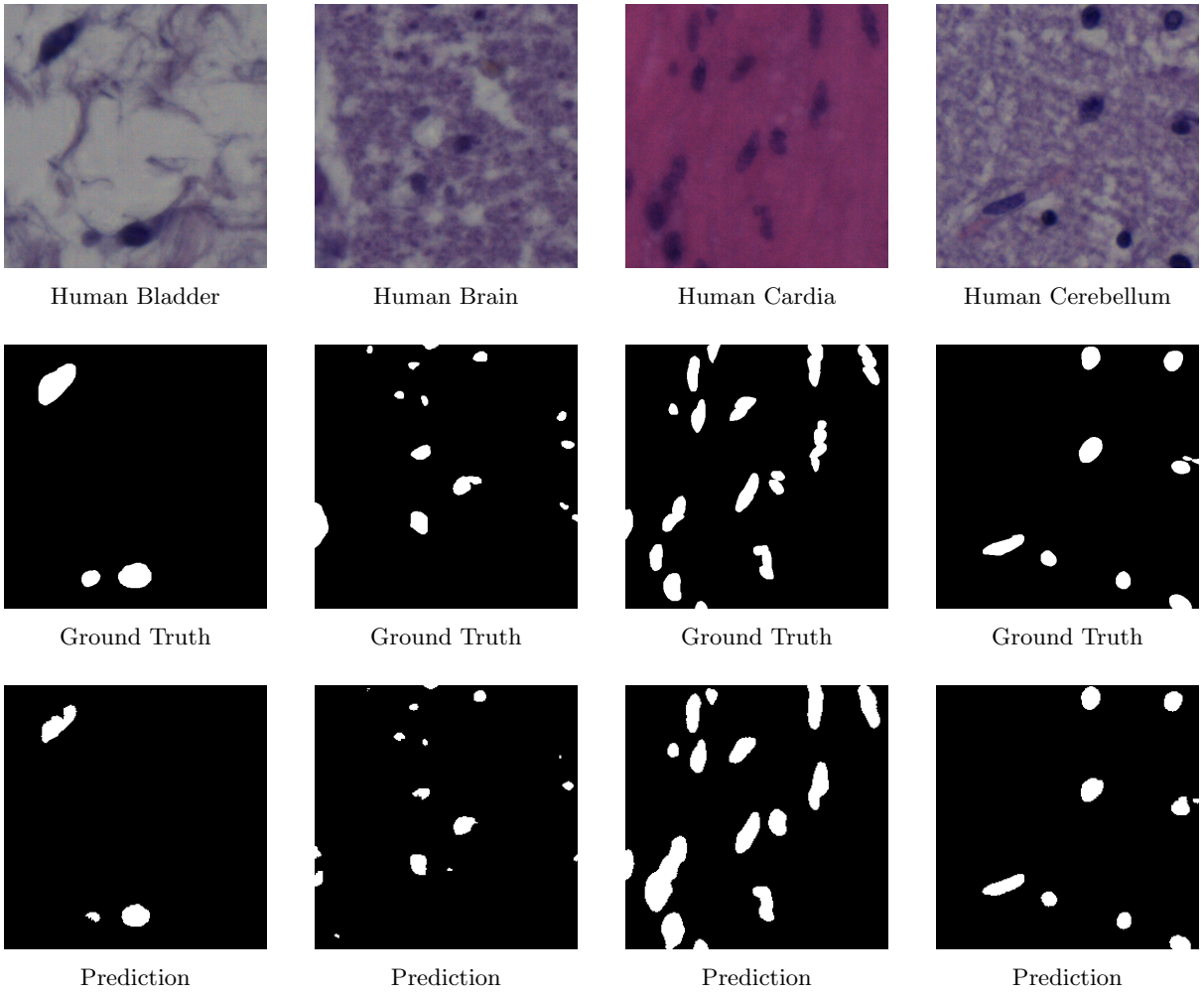


Figure 6: Qualitative results showing original images (top), ground truth masks (middle), and model predictions (bottom) for various organ tissues.