Promptable Pathology: Complete Project Documentation

Visual Foundation Models for Medical Image Analysis

CSCE 689 - Fall 2025, Texas A&M University

Table of Contents

1. Executive Summary
2. Problem Statement & Motivation
3. Dataset: BCSS
4. Technical Architecture
5. Model Components
6. Experiments & Results
7. Training Details
8. Technical Deep Dive: Concepts & Mechanisms
    - 8.1 Data Augmentation Pipeline
    - 8.2 Prompt Generation Mechanisms
    - 8.3 Test-Time Augmentation (TTA)
    - 8.4 Loss Functions
    - 8.5 LoRA (Low-Rank Adaptation)
    - 8.6 Mask Post-Processing
    - 8.7 Evaluation Metrics
9. Key Findings
10. Code Architecture
11. Reproduction Guide

Appendices

- Appendix A: Complete Experiment Catalog - All 17 experiment configurations
- Appendix B: Prompt Files - CLIP prompt definitions
- Appendix C: Complete Metrics - Detailed per-class results
- Appendix D: References - Academic citations
- Appendix E: Generated Figures - Qualitative & academic figures

1. Executive Summary

1.1 Project Goal

Develop a zero-shot pipeline for semantic segmentation and classification of breast cancer histopathology images using Visual Foundation Models (VFMs), specifically SAM2 for segmentation and CLIP for classification.

1.2 Key Results Summary

Task	Best Method	Performance	Key Insight
Segmentation	SAM2 + Box + Neg Points	**0.555 Dice**	Zero-shot outperforms finetuning
Classification	CLIP + LLM Few-shot Prompts	**44.4% Accuracy**	Prompt engineering > manual prompts
Finetuning	All methods	0.35-0.37 Dice	Failed due to small dataset

1.3 Main Contributions

1. **Comprehensive evaluation** of SAM2 prompting strategies for histopathology
2. **Systematic comparison** of CLIP prompt engineering approaches (manual vs LLM-generated)
3. **Evidence that zero-shot > finetuning** for small medical datasets (n < 100)
4. **Open-source pipeline** with reproducible experiments on BCSS dataset

---

2. Problem Statement & Motivation

2.1 Clinical Context

Breast cancer diagnosis requires pathologists to analyze H&E-stained tissue sections and identify:

- **Tumor regions** - malignant epithelial cells
- **Stroma** - supportive connective tissue
- **Lymphocytes** - immune infiltration (prognostic marker)
- **Necrosis** - dead tissue (indicator of aggressive tumors)
- **Blood vessels** - vascular invasion assessment

Manual analysis is:

- Time-consuming (15-30 minutes per slide)
- Subject to inter-observer variability
- Dependent on expert availability

2.2 Why Visual Foundation Models?

Traditional Approach Problems

1. **Requires large labeled datasets** - Medical annotation is expensive ($50-100/image)
2. **Domain-specific training** - Models don't generalize across cancer types
3. **Class imbalance** - Rare structures (blood vessels) are underrepresented

VFM Advantages

1. **Zero-shot capability** - No task-specific training required
2. **Promptable interface** - Flexible interaction via points, boxes, or text
3. **Massive pretraining** - Billions of natural images provide robust features

2.3 Research Questions

1. **RQ1**: Can SAM2 segment histopathology structures without finetuning?
2. **RQ2**: How do different prompting strategies affect SAM2 performance?
3. **RQ3**: Can CLIP classify tissue types using text descriptions alone?
4. **RQ4**: Does LLM-generated prompts outperform manual prompts?
5. **RQ5**: Does finetuning improve or hurt performance on small datasets?

---

3. Dataset: BCSS (Breast Cancer Semantic Segmentation)

3.1 Dataset Overview

Attribute	Value
Source	TCGA (The Cancer Genome Atlas)
Total Images	151
Resolution	1024 × 1024 pixels
Staining	H&E (Hematoxylin & Eosin)
Magnification	40x
Annotation Type	Pixel-wise semantic masks
Number of Classes	5 (excluding background)

3.2 Class Definitions

Class 1: Tumor (Invasive Carcinoma)

- **Appearance**: Densely packed cells with large, irregular purple nuclei
- **Characteristics**: High nuclear-to-cytoplasmic ratio, loss of normal architecture
- **Frequency**: 33.8% of annotated pixels
- **Clinical Significance**: Primary diagnostic target

Class 2: Stroma

- **Appearance**: Pink fibrous tissue with elongated spindle-shaped nuclei
- **Characteristics**: Collagen fibers, fibroblasts, loose connective tissue
- **Frequency**: 26.2% of annotated pixels
- **Clinical Significance**: Tumor microenvironment, desmoplastic reaction

Class 3: Lymphocyte (Tumor-Infiltrating Lymphocytes)

- **Appearance**: Small, dark, round nuclei densely clustered
- **Characteristics**: Uniform size, minimal cytoplasm, often in aggregates
- **Frequency**: 5.9% of annotated pixels
- **Clinical Significance**: Immune response indicator, prognostic marker

Class 4: Necrosis

- **Appearance**: Pale, washed-out pink areas with fragmented nuclei
- **Characteristics**: Loss of cellular structure, nuclear debris, ghostly outlines
- **Frequency**: 6.9% of annotated pixels
- **Clinical Significance**: Indicates aggressive tumor, poor prognosis

Class 18: Blood Vessel

- **Appearance**: Circular lumens with thin endothelial lining
- **Characteristics**: Hollow structures, smooth pink walls, may contain RBCs
- **Frequency**: 0.5% of annotated pixels
- **Clinical Significance**: Vascular invasion assessment, angiogenesis

3.3 Data Split Strategy

```
Total: 151 images
├── Train: 85 images (56%)
├── Validation: 21 images (14%)
└── Test: 45 images (30%)
```

Split Logic (from `src/dataset.py`):

```
# Test set defined by TCGA case prefixes (ensures patient-level separation)
test_prefixes = ['TCGA-OL-', 'TCGA-LL-', 'TCGA-E2-',
                 'TCGA-EW-', 'TCGA-GM-', 'TCGA-S3-']

# Remaining images split 80/20 for train/val with fixed seed
np.random.seed(42)
```

Rationale: Patient-level splitting prevents data leakage from same-patient patches appearing in train and test sets.

3.4 Class Imbalance Analysis

Class	Pixel %	Images with Class	Inverse Weight
Tumor	33.8%	45/45 (100%)	1.0
Stroma	26.2%	45/45 (100%)	1.3
Lymphocyte	5.9%	37/45 (82%)	5.7
Necrosis	6.9%	23/45 (51%)	4.9
Blood Vessel	0.5%	31/45 (69%)	67.6

Key Challenge: Blood vessels constitute only 0.5% of pixels but are clinically important for vascular invasion assessment.

4. Technical Architecture

4.1 Pipeline Overview

```
┌──────────────┐    ┌──────────────┐    ┌──────────────┐
│    Input     │    │     SAM2     │    │     CLIP     │
│    Image     │───▶│ Segmentation │───▶│Classification│
│   (1024²)    │    │   (Masks)    │    │   (Labels)   │
└──────────────┘    └──────────────┘    └──────────────┘
       │                   │                   │
       │                   ▼                   ▼
       │            ┌──────────────┐    ┌──────────────┐
       │            │  Per-Class   │    │    Tissue    │
       │            │    Binary    │    │     Type     │
       │            │    Masks     │    │  Prediction  │
       └───────────▶└──────────────┘    └──────────────┘
                           │                   │
                           ▼                   ▼
                    ┌─────────────────────────────┐
                    │     Evaluation Metrics      │
                    │  Dice, IoU, Accuracy, F1    │
                    └─────────────────────────────┘
```

4.2 Prompt Flow

SAM2 Prompting (Segmentation)

```
Ground Truth Mask → Prompt Generator → SAM2 Predictor → Predicted Mask
                           │
                           ├── Centroid Point
                           ├── Multi-Point (5 boundary points)
                           ├── Bounding Box
                           └── Box + Negative Points
```

CLIP Prompting (Classification)

```
Cropped Region → CLIP Image Encoder ─┐
                                     ├── Similarity → Class Prediction
Text Prompts → CLIP Text Encoder ────┘
```

4.3 Evaluation Protocol

1. **For each test image** (n=45):
2. **For each class present** in the ground truth: a. Extract binary mask for that class b. Generate prompts from mask (simulating user interaction) c. Run SAM2 to get predicted mask d. Crop predicted region from image e. Run CLIP classification on cropped region f. Record Dice, IoU for segmentation g. Record accuracy for classification

Note: This "oracle" setup uses ground truth to generate prompts, measuring upper-bound performance assuming perfect user interaction.

5. Model Components

5.1 SAM2 (Segment Anything Model 2)

Architecture

Component	Configuration
Backbone	Hiera-L (Hierarchical Vision Transformer)
Parameters	224 million
Input Resolution	1024 × 1024
Feature Pyramid	4 levels (144, 288, 576, 1152 channels)
Memory Attention	4 layers, 256 dim

Prompting Modes

```
# From src/sam_segmentation.py

def get_prompts_from_mask(binary_mask, num_points=5, neg_point_margin=10):
    """Generates various prompts from a binary mask."""
    prompts = {}

    # 1. Bounding Box - rectangle around largest contour
    x, y, w, h = cv2.boundingRect(largest_contour)
    prompts['box'] = np.array([[x, y], [x + w, y + h]])

    # 2. Centroid - center of mass
    M = cv2.moments(largest_contour)
    cx = int(M["m10"] / M["m00"])
    cy = int(M["m01"] / M["m00"])
    prompts['centroid'] = (np.array([[cx, cy]]), np.array([1]))

    # 3. Multi-point - evenly spaced boundary points
    step = len(largest_contour) // num_points
    points = largest_contour[::step, 0, :]
    prompts['multi_point'] = (points, np.ones(len(points)))

    # 4. Negative points - outside bounding box
    neg_points = []  # Points sampled outside bbox
    prompts['neg_points'] = (np.array(neg_points), np.zeros(len(neg_points)))

    return prompts
```

Inference Code

```
def get_predicted_mask_from_prompts(predictor, image, prompts,
                                    prompt_type='box', use_neg_points=False):
    predictor.set_image(image)

    # Combine positive and negative points if requested
    if use_neg_points and 'neg_points' in prompts:
        point_coords = np.concatenate([point_coords, neg_coords], axis=0)
        point_labels = np.concatenate([point_labels, neg_labels], axis=0)

    with torch.autocast("cuda", dtype=torch.bfloat16):
        masks, _, _ = predictor.predict(
            point_coords=point_coords,
            point_labels=point_labels,
            box=box,
            multimask_output=False
        )

    return masks[0]
```

5.2 CLIP (Contrastive Language-Image Pretraining)

Architecture

Component	Configuration
Model	ViT-B/32
Image Encoder	Vision Transformer, 32×32 patches
Text Encoder	Transformer, 77 token context
Embedding Dim	512
Parameters	151 million

Classification Logic

```python
# From src/clip_classification.py

class CLIPClassifier:
    def classify_region(self, image, prompts):
        """Classifies using text prompt ensemble."""

        # Create text embeddings for all prompts
        text_inputs = [prompt for class_name in class_names
                      for prompt in prompts[class_name]]

        # Get image-text similarities
        outputs = self.model(**inputs)
        logits_per_image = outputs.logits_per_image
        probs = logits_per_image.softmax(dim=1)

        # Average probabilities per class (ensemble of prompts)
        class_probs = []
        for num_prompts in num_prompts_per_class:
            class_probs.append(probs[:, start:start+num_prompts].mean(dim=1))

        # Return class with highest average probability
        predicted_class = class_names[avg_probs.argmax()]
        return predicted_class
```

5.3 MedSAM (Medical SAM)

Architecture Differences from SAM2

Attribute	SAM2 Hiera-L	MedSAM ViT-B
Backbone	Hiera (hierarchical)	ViT-B (vanilla)
Parameters	224M	86M
Training Data	SA-1B (natural images)	1.5M medical images
Specialization	General-purpose	Medical domain

Usage

```python
# MedSAM uses box prompts with optional TTA
from segment_anything import sam_model_registry

medsam = sam_model_registry["vit_b"](checkpoint="medsam_vit_b.pth")
predictor = SamPredictor(medsam)

# Box prompt format: [x0, y0, x1, y1]
masks, scores, _ = predictor.predict(box=box_prompt)
```

6. Experiments & Results

6.1 SAM2 Segmentation Experiments

Experiment 1: Prompt Type Ablation

Hypothesis: Bounding boxes provide more spatial context than point prompts.

Prompt Type	Dice Mean	Dice Std	IoU Mean	Samples
Centroid	0.338	0.263	0.236	180
Multi-Point (5)	0.418	0.209	0.287	170
Box	0.553	0.195	0.407	181
Box + Neg Points	**0.555**	0.193	**0.408**	181

Key Finding: Box prompts provide 64% improvement over centroid points.

Experiment 2: Per-Class Analysis (Box + Neg)

Class	Dice	Std	IoU	Count
Tumor	0.560	0.147	0.403	45
Stroma	0.538	0.166	0.385	45
Lymphocyte	0.532	0.218	0.391	37
Necrosis	**0.691**	0.194	**0.559**	23
Blood Vessel	0.497	0.208	0.357	31

Insight: Necrosis achieves highest Dice (0.691) due to distinct pale appearance. Blood vessels are hardest due to small size and thin walls.

Experiment 3: SAM2 vs MedSAM

Model	Prompt	TTA	Dice	IoU
SAM2 Hiera-L	Box+Neg	No	**0.555**	**0.408**
SAM2 Hiera-L	Box	No	0.553	0.407
MedSAM ViT-B	Box	Yes	0.536	0.389
MedSAM ViT-B	Box	No	0.522	0.375

Key Finding: SAM2 outperforms MedSAM despite MedSAM's medical-specific training. The larger Hiera backbone likely provides better feature representations.

6.2 CLIP Classification Experiments

Experiment 4: Prompt Engineering Comparison

Prompt Source	Strategy	Accuracy	F1 Macro
LLM (GPT-4)	**Text + Few-shot**	**44.4%**	**0.338**
Manual	Hardcoded v2	42.2%	0.311
LLM (GPT-4)	Text + CLIP-optimized	35.6%	0.270
LLM (Gemini)	Multimodal + Few-shot	29.4%	0.220
Manual	Hardcoded v1	23.3%	0.138
LLM (Gemini)	Multimodal + CLIP-opt	15.0%	0.100
LLM (GPT-4)	Text + Jargon	12.2%	0.097
LLM (Gemini)	Multimodal v1	8.3%	0.091

Key Findings:

1. LLM few-shot prompts outperform manual by 2.2%
2. Text-only prompts outperform multimodal by 15%
3. Medical jargon hurts performance (CLIP trained on natural language)

Experiment 5: Per-Class Classification (Best Config)

Class	Precision	Recall	F1	Support
Tumor	97.4%	82.2%	89.2%	45
Stroma	61.9%	28.9%	39.4%	45
Lymphocyte	0.0%	0.0%	0.0%	37
Necrosis	0.0%	0.0%	0.0%	23
Blood Vessel	25.4%	100%	40.5%	30

Critical Issue: CLIP fails completely on lymphocyte and necrosis classes. These require domain-specific features that CLIP's natural image pretraining doesn't capture.

---

7. Training Details

7.1 Finetuning Strategy Overview

We explored three finetuning approaches to adapt SAM2 to histopathology:

Approach	Parameters Trained	Epochs	Loss Function	Test Dice
Base Finetune	224M (full model)	100	BCE + Dice	0.371
Focal Loss	224M (decoder only)	50	Focal + Dice	0.372
LoRA Light	4.2M (adapters)	30	BCE + Dice + IoU	0.355

Baseline (Zero-Shot): 0.555 Dice

7.2 Training Configuration: SAM2 Focal Loss

Hyperparameters

```yaml
# From conf/experiment/sam2_box_focal.yaml

scratch:
  resolution: 1024
  train_batch_size: 4
  base_lr: 3.0e-05          # Moderate learning rate
  vision_lr: 0.0            # Frozen image encoder
  num_epochs: 50
  warmup_steps: 200

# Data augmentation
transforms:
  - RandomHorizontalFlip
  - RandomVerticalFlip
  - RandomAffine(degrees=90)
  - ColorJitter(brightness=0.15, contrast=0.15)
  - Normalize(ImageNet stats)

# Loss function
loss:
  _target_: src.focal_loss.FocalDiceLoss
  alpha:                    # Class weights (inverse frequency)
    1: 1.0                  # tumor
    2: 1.3                  # stroma
    3: 5.7                  # lymphocyte
    4: 4.9                  # necrosis
    18: 67.6                # blood_vessel (67x upweight!)
  gamma: 2.0                # Focal focusing parameter
  dice_weight: 3.0
  focal_weight: 1.0
```

Focal Loss Implementation

```python
# From src/focal_loss.py

class FocalDiceLoss(nn.Module):
    """
    Combines Focal Loss and Dice Loss for class imbalance.

    Focal Loss: FL(pt) = -αt(1-pt)^γ log(pt)
    - Reduces loss for well-classified examples
    - Focuses training on hard examples

    Dice Loss: DL = 1 - (2|P∩G|)/(|P|+|G|)
    - Overlap-based, handles class imbalance
    """

    def forward(self, outputs, targets):
        # Focal Loss
        bce_loss = F.binary_cross_entropy_with_logits(pred, target, reduction='none')
        pt = torch.exp(-bce_loss)
        focal_term = (1 - pt) ** self.gamma * bce_loss
        focal_loss = (self.alpha * focal_term).mean()

        # Dice Loss
        pred_probs = torch.sigmoid(pred)
        intersection = (pred_probs * target).sum()
        dice = 2 * intersection / (pred_probs.sum() + target.sum())
        dice_loss = 1 - dice

        return focal_loss + self.dice_weight * dice_loss
```

Training Curves (50 epochs)

```
Epoch 0:  Loss=45.4, Focal=42.6, Dice=0.94
Epoch 10: Loss=38.5, Focal=36.2, Dice=0.77
Epoch 20: Loss=35.1, Focal=32.9, Dice=0.73
Epoch 30: Loss=32.4, Focal=30.2, Dice=0.71
Epoch 40: Loss=30.1, Focal=28.0, Dice=0.70
Epoch 50: Loss=29.0, Focal=27.0, Dice=0.70
```

Observation: Loss decreases steadily but test Dice (0.372) is worse than zero-shot (0.555).

7.3 Training Configuration: LoRA (Low-Rank Adaptation)

Concept

LoRA adds small trainable matrices to frozen pretrained weights:

```
Output = Original_Output + (α/r) × B @ A @ x

where:
- A: (d_in, r) projects to low-rank space
- B: (r, d_out) projects back
- r << d_in, d_out (typically 4-16)
- α: scaling factor
```

Implementation

```python
# From src/lora_adapter.py

class LoRALinear(nn.Module):
    def __init__(self, original_linear, r=8, alpha=None):
        self.original_linear = original_linear
        self.r = r
        self.alpha = alpha if alpha else float(r)

        # Freeze original weights
        for param in original_linear.parameters():
            param.requires_grad = False

        # LoRA matrices (only these are trained)
        self.lora_A = nn.Parameter(torch.zeros(in_features, r))
        self.lora_B = nn.Parameter(torch.zeros(r, out_features))

        # Initialize A with Kaiming, B with zeros
        nn.init.kaiming_uniform_(self.lora_A)
        nn.init.zeros_(self.lora_B)

    def forward(self, x):
        original = self.original_linear(x)
        lora = x @ self.lora_A @ self.lora_B * (self.alpha / self.r)
        return original + lora
```

LoRA Configuration

```yaml
# From conf/experiment/sam2_lora_light.yaml

scratch:
  base_lr: 5.0e-06          # 10x smaller than Focal
  num_epochs: 30            # Shorter training
  warmup_steps: 300         # 25% warmup

# Minimal augmentation to preserve pretrained knowledge
transforms:
  - RandomHorizontalFlip
  - RandomVerticalFlip
  - ColorJitter(brightness=0.1)  # Half of Focal
```

Training Curves (30 epochs)

```
Epoch 0:  Loss=36.6, Mask=14.3, Dice=0.96, IoU=0.23
Epoch 5:  Loss=11.3, Mask=3.2,  Dice=0.87, IoU=0.11
Epoch 10: Loss=4.3,  Mask=0.1,  Dice=0.74, IoU=0.14
Epoch 20: Loss=4.0,  Mask=0.1,  Dice=0.72, IoU=0.14
Epoch 30: Loss=4.0,  Mask=0.1,  Dice=0.72, IoU=0.14
```

Observation: Very fast convergence (by epoch 7), but overfits to training set.

7.4 Why Finetuning Failed

Root Cause Analysis

1. **Insufficient Training Data**
    - Only 85 training images
    - SAM2 pretrained on 11M images
    - Ratio: 0.0008% of original training data
2. **Catastrophic Forgetting**
    - Finetuning destroys general-purpose features
    - Histopathology is a narrow domain
    - Model over-specializes to training distribution
3. **Distribution Shift**
    - Training set from specific TCGA centers
    - Test set from different centers (patient-level split)
    - Model learns center-specific artifacts
4. **Class Imbalance**
    - Blood vessels: 0.5% of pixels
    - Weighted loss helps training but not generalization

Evidence from Per-Class Results

Class	Zero-Shot	Focal FT	Δ
Tumor	0.560	0.386	-31%
Stroma	0.538	0.386	-28%
Lymphocyte	0.532	0.301	-43%
Necrosis	0.691	0.478	-31%
Blood Vessel	0.497	0.335	-33%

All classes degrade after finetuning, indicating global knowledge loss.

7.5 Test-Time Augmentation (TTA)

Implementation

```python
# From src/tta_utils.py

def predict_with_tta(predictor, image, prompts, num_augmentations=4):
    """
    Apply test-time augmentation for robust predictions.

    Augmentations:
    1. Original
    2. Horizontal flip
    3. Vertical flip
    4. 90° rotation
    """
    augmented_masks = []

    for aug_type in ['original', 'hflip', 'vflip', 'rot90']:
        # Augment image and prompts
        aug_image = apply_augmentation(image, aug_type)
        aug_prompts = transform_prompts(prompts, aug_type)

        # Predict
        mask = predictor.predict(aug_image, aug_prompts)

        # De-augment mask
        de_aug_mask = reverse_augmentation(mask, aug_type)
        augmented_masks.append(de_aug_mask)

    # Average predictions
    ensemble = np.mean(augmented_masks, axis=0)
    return (ensemble > 0.5).astype(np.uint8)
```

TTA Results

Model	Without TTA	With TTA	Improvement
MedSAM	0.522	0.536	+1.4%
SAM2	0.553	0.555	+0.2%

Conclusion: TTA provides marginal but consistent improvements.

---

8. Technical Deep Dive: Concepts & Mechanisms

This section provides detailed explanations of all technical concepts, algorithms, and mechanisms used in this project.

8.1 Data Augmentation Pipeline

8.1.1 Geometric Augmentations

Histopathology images have **no natural orientation** - tissue can be rotated arbitrarily during slide preparation. Our geometric augmentations exploit this property:

Random Horizontal Flip

```python
RandomHorizontalFlip(consistent_transform=True)
```

- **Operation**: Mirror image left-to-right with 50% probability
- **Rationale**: Tissue structure is symmetric; cells don't have left/right preference
- **Consistent Transform**: Same flip applied to image AND mask

Random Vertical Flip

```
RandomVerticalFlip(consistent_transform=True)
```

- **Operation**: Mirror image top-to-bottom with 50% probability
- **Rationale**: No gravitational orientation in tissue sections

Random Affine (Rotation)

```
RandomAffine(
    degrees=90,                 # Rotate by multiples of 90°
    consistent_transform=True,
    num_tentatives=2            # Retry if mask becomes empty
)
```

- **Why 90°**: Larger rotations preserve structure better than arbitrary angles
- **Why no scale/translate**: Avoided to prevent mask area becoming zero
- **Num Tentatives**: If rotation erases the mask (rare edge case), retry with different angle

8.1.2 Color/Intensity Augmentations

H&E (Hematoxylin & Eosin) stained slides exhibit significant **inter-laboratory color variation** due to:

- Different staining protocols
- Scanner calibration differences
- Tissue preparation variations
- Fading over time

ColorJitter Transform

```
ColorJitter(
    brightness=0.2,    # ±20% brightness variation
    contrast=0.2,      # ±20% contrast variation
    saturation=0.2,    # ±20% saturation variation
    hue=0.05,          # ±5% hue shift (color tone)
    consistent_transform=True
)
```

Brightness: Simulates different illumination levels during scanning **Contrast**: Mimics different stain concentrations **Saturation**: Handles variation in stain intensity **Hue**: Small shifts to handle H&E color drift (kept conservative to avoid unrealistic colors)

8.1.3 Macenko Stain Normalization

The most sophisticated augmentation: decomposing and reconstructing H&E stains.

Theory: Optical Density (OD) Space

```
def convert_RGB_to_OD(I):
    """Convert RGB to Optical Density space."""
    mask = (I == 0)
    I[mask] = 1  # Avoid log(0)
    return np.maximum(-1 * np.log(I / 255.0), 1e-6)
```

Beer-Lambert Law: OD = $-\log_{10}(I/I_0)$

- Light absorbed by stain is proportional to concentration
- OD space linearizes stain concentration

Macenko Algorithm Steps

1. **Extract Stain Matrix:** Find the two principal stain vectors (H and E) using PCA on tissue pixels

```
def get_stain_matrix(I, luminosity_threshold=0.8, angular_percentile=99):
    # 1. Mask out background using luminosity
    tissue_mask = LuminosityThresholdTissueLocator.get_tissue_mask(I)

    # 2. Convert to OD and run PCA
    OD = convert_RGB_to_OD(I)[tissue_mask]
    _, V = np.linalg.eigh(np.cov(OD, rowvar=False))

    # 3. Project onto plane and find extreme angles
    That = np.dot(OD, V[:, [2, 1]])
    phi = np.arctan2(That[:, 1], That[:, 0])

    # 4. Extreme angles correspond to H and E vectors
    minPhi = np.percentile(phi, 100 - angular_percentile)
    maxPhi = np.percentile(phi, angular_percentile)

    return normalize_matrix_rows(HE_vectors)
```

2. **Decompose Concentrations:** Solve for stain concentrations using LASSO regression

```
def get_concentrations(I, stain_matrix, regularizer=0.01):
    OD = convert_RGB_to_OD(I).reshape((-1, 3))
    lasso = Lasso(alpha=regularizer, positive=True)
    lasso.fit(stain_matrix.T, OD.T)
    return lasso.coef_.T
```

3. **Normalize/Augment:** Rescale concentrations to target distribution

```
class StainAugmentor:
    def augment(self, concentrations):
        # Random scaling and shifting of stain concentrations
        alpha = 1 + np.random.uniform(-0.2, 0.2, size=2)  # Scale
        beta = np.random.uniform(-0.2, 0.2, size=2)       # Shift
        return concentrations * alpha + beta
```

4. **Reconstruct:** Convert back to RGB using target stain matrix

```
transformed_img = 255 * np.exp(-1 * np.dot(augmented_concentrations, target_stain_matrix))
```

Benefits:

- Creates realistic stain variations
- Preserves tissue structure while changing colors
- Better generalization to unseen laboratories

8.1.4 ImageNet Normalization

```
NormalizeAPI(
    mean=[0.485, 0.456, 0.406],
    std=[0.229, 0.224, 0.225]
)
```

- **Why ImageNet stats:** SAM2 and CLIP were pretrained on ImageNet-normalized images
- **Effect:** Shifts pixel values to have zero mean and unit variance per channel
- **Critical:** Must match pretraining normalization exactly

8.1.5 Augmentation Strength Comparison

Config	Brightness	Contrast	Saturation	Hue	Rotation
Minimal (LoRA)	0.1	0.1	0.1	0.02	None
Mild (v2_stable)	0.2	0.2	0.2	0.05	90°
Standard (base)	0.3	0.3	0.3	0.08	90°
Strong (strong_aug)	0.4	0.4	0.4	0.10	90°

Key Finding: Minimal augmentation preserved pretrained features best for LoRA.

8.2 Prompt Generation Mechanisms

8.2.1 SAM2 Prompt Types

SAM2 accepts three types of spatial prompts:

Centroid (Single Point)

```python
def get_centroid_prompt(binary_mask):
    M = cv2.moments(largest_contour)
    cx = int(M["m10"] / M["m00"])  # Center of mass X
    cy = int(M["m01"] / M["m00"])  # Center of mass Y
    return (np.array([[cx, cy]]), np.array([1]))  # Label 1 = foreground
```

- **Pros**: Minimal information, simulates quick user click
- **Cons**: Ambiguous for complex shapes, SAM may not know extent
- **Performance:** 0.338 Dice (lowest)

Multi-Point (Boundary Sampling)

```python
def get_multipoint_prompt(binary_mask, num_points=5):
    contours, _ = cv2.findContours(binary_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    largest_contour = max(contours, key=cv2.contourArea)

    # Sample points evenly along contour
    step = len(largest_contour) // num_points
    points = largest_contour[::step, 0, :]
    labels = np.ones(len(points))  # All foreground

    return (points, labels)
```

- **Pros**: Defines boundary better than single point
- **Cons**: May not fully constrain region
- **Performance:** 0.418 Dice (+24% vs centroid)

Bounding Box

```python
def get_box_prompt(binary_mask):
    contours, _ = cv2.findContours(binary_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    largest_contour = max(contours, key=cv2.contourArea)
    x, y, w, h = cv2.boundingRect(largest_contour)

    # SAM expects [[x0, y0], [x1, y1]]
    return np.array([[x, y], [x + w, y + h]])
```

- **Pros**: Defines spatial extent clearly, no ambiguity about region
- **Cons**: Includes background within box
- **Performance:** 0.553 Dice (+64% vs centroid)

Negative Points

```
def get_negative_points(bounding_box, num_points=5, margin=10):
    x, y, w, h = bounding_box
    neg_points = []

    for _ in range(num_points):
        side = np.random.randint(4)
        if side == 0:  # Top
            neg_points.append([np.random.randint(x, x+w), y - margin])
        elif side == 1:  # Bottom
            neg_points.append([np.random.randint(x, x+w), y + h + margin])
        elif side == 2:  # Left
            neg_points.append([x - margin, np.random.randint(y, y+h)])
        else:  # Right
            neg_points.append([x + w + margin, np.random.randint(y, y+h)])

    return (np.array(neg_points), np.zeros(len(neg_points)))  # Label 0 = background
```

- **Purpose**: Tell SAM what NOT to include
- **Combined with Box**: Provides clearer boundary signal
- **Performance:** 0.555 Dice (+0.4% vs box alone)

8.2.2 Training-Time Prompt Generation

During finetuning, prompts are generated on-the-fly:

```
class BCSSSegmentLoader:
    def __init__(self, prompt_type="mixed", use_neg_points=False, num_points=5):
        self.prompt_type = prompt_type
        self.use_neg_points = use_neg_points
        self.num_points = num_points

    def load(self, frame_id):
        # For each object in the mask
        for obj_id in object_ids:
            binary_mask = (masks == obj_id)
            prompt_dict = get_prompts_from_mask(binary_mask)

            # Mixed prompt: randomly choose box or centroid
            if self.prompt_type == 'mixed':
                current_type = 'box' if np.random.rand() > 0.5 else 'centroid'

            # SAM2 training uses special labels for box corners
            if current_type == 'box':
                point_coords = box_corners
                point_labels = np.array([2, 3])  # 2=top-left, 3=bottom-right
```

8.2.3 CLIP Text Prompt Engineering

Strategy 1: Manual Expert Prompts (Hardcoded)

```
{
    "tumor": [
        "densely crowded dark purple cells packed together",
        "large irregular purple nuclei in chaotic arrangement",
        "thick masses of deep purple tissue with high cell density"
    ]
}
```

- **Approach**: Domain expert writes visually descriptive phrases
- **Principle**: Describe colors, textures, patterns - NOT medical terminology
- **Result**: 42.2% accuracy

Strategy 2: LLM Text-Only Prompts (GPT-4)

```
Meta-prompt to GPT-4:
"You are an expert pathologist. Generate 7 short, visually descriptive
phrases for [class_name] tissue. Focus on colors, cell shapes, textures.
Do NOT use medical jargon. These will be used with CLIP which understands
natural images, not medical terminology."
```

- **Result**: Better prompts that match CLIP's vocabulary
- **Few-shot version**: Include 3-5 example prompts first

Strategy 3: LLM Multimodal Prompts (Gemini)

```
def generate_multimodal_prompts():
    # Get example images of each class
    example_images = get_example_images(class_name, n=5)

    # Send images + meta-prompt to Gemini
    api_prompt = example_images + [meta_prompt_text]
    response = model.generate_content(api_prompt)

    # Parse response into prompt list
    return [phrase.strip() for phrase in response.text.split('\n')]
```

- **Advantage**: LLM sees actual tissue images, not just class names
- **Disadvantage**: Gemini descriptions were too medical/jargon-heavy
- **Result**: 8.3% - 29.4% accuracy (worse than text-only)

Key Insight: CLIP was trained on natural language captions, not medical text. Simple visual descriptions ("dark purple dots") outperform medical terminology ("lymphocytic infiltrate").

8.2.4 CLIP Classification Logic

```
def classify_region(self, image, prompts):
    # 1. Flatten all prompts for all classes
    class_names = list(prompts.keys())
    all_text = [p for cls in class_names for p in prompts[cls]]

    # 2. Compute image-text similarity scores
    inputs = processor(text=all_text, images=image, return_tensors="pt")
    outputs = model(**inputs)
    logits = outputs.logits_per_image  # Shape: (1, num_prompts)
    probs = logits.softmax(dim=1)

    # 3. Average probabilities per class (prompt ensemble)
    class_probs = []
    idx = 0
    for cls in class_names:
        n = len(prompts[cls])
        class_probs.append(probs[:, idx:idx+n].mean())
        idx += n

    # 4. Return class with highest average probability
    return class_names[torch.stack(class_probs).argmax()]
```

Prompt Ensemble: Multiple prompts per class improves robustness. If one prompt fails to match, others may succeed.

8.3 Test-Time Augmentation (TTA)

8.3.1 Concept

TTA applies multiple augmentations at inference time and averages predictions:

```
Final Prediction = Average(
    Predict(Original),
    De-augment(Predict(Augment₁)),
    De-augment(Predict(Augment₂)),
    ...
)
```

8.3.2 Implementation Details

```python
def predict_with_tta(predictor, image, prompts, num_augmentations=4):
    h, w, _ = image.shape
    augmented_masks = []

    for aug_type in ['original', 'hflip', 'vflip', 'rot90']:
        # 1. Augment image
        if aug_type == 'hflip':
            aug_image = np.fliplr(image).copy()
        elif aug_type == 'vflip':
            aug_image = np.flipud(image).copy()
        elif aug_type == 'rot90':
            aug_image = np.rot90(image).copy()

        # 2. Transform prompts to match augmented image
        aug_prompts = transform_prompts(prompts, aug_type)

        # 3. Predict on augmented image
        mask = predictor.predict(aug_image, aug_prompts)

        # 4. De-augment mask back to original coordinates
        if aug_type == 'hflip':
            mask = np.fliplr(mask)
        elif aug_type == 'vflip':
            mask = np.flipud(mask)
        elif aug_type == 'rot90':
            mask = np.rot90(mask, k=-1)  # Rotate back

        augmented_masks.append(mask.astype(np.float32))

    # 5. Average and threshold
    ensemble = np.mean(augmented_masks, axis=0)
    return (ensemble > 0.5).astype(np.uint8)
```

8.3.3 Coordinate Transformation for Prompts

```python
def transform_coords(coords, transform_type, h, w):
    new_coords = coords.copy()

    if transform_type == 'hflip':
        new_coords[:, 0] = w - new_coords[:, 0]  # x → w - x
    elif transform_type == 'vflip':
        new_coords[:, 1] = h - new_coords[:, 1]  # y → h - y
    elif transform_type == 'rot90':
        new_coords = new_coords[:, [1, 0]]  # Swap x, y
        new_coords[:, 1] = w - new_coords[:, 1]  # Adjust for rotation

    return new_coords
```

8.3.4 When TTA Helps

Scenario	TTA Benefit
Symmetric structures	Marginal (already recognized)
Edge cases	Helpful (averaging smooths errors)
Ambiguous boundaries	Helpful (reduces noise)
Small objects	Minimal (often missed by all augmentations)

8.4 Loss Functions

8.4.1 Standard SAM2 Loss (MultiStepMultiMasksAndIous)

```
weight_dict:
  loss_mask: 20     # Binary cross-entropy on mask
  loss_dice: 1      # Dice loss for overlap
  loss_iou: 1       # IoU prediction loss
  loss_class: 1     # Object classification loss
```

- **loss_mask**: Per-pixel BCE, sensitive to class imbalance
- **loss_dice**: 1 - (2|P∩G|)/(|P|+|G|), naturally handles imbalance
- **loss_iou**: Trains IoU prediction head for quality estimation
- **loss_class**: Trains object presence/absence prediction

8.4.2 Focal Loss

Addresses class imbalance by down-weighting easy examples:

```
FL(pt) = -αt(1 - pt)^γ log(pt)

where:
- pt = model's probability for the correct class
- γ = focusing parameter (typically 2.0)
- αt = class weight
```

```
class FocalDiceLoss(nn.Module):
    def forward(self, outputs, targets):
        # Focal component
        bce_loss = F.binary_cross_entropy_with_logits(pred, target, reduction='none')
        pt = torch.exp(-bce_loss)  # Probability of correct class
        focal_term = (1 - pt) ** self.gamma * bce_loss  # Down-weight easy examples

        # Apply class weights
        alpha_t = self.alpha[class_ids]  # e.g., blood_vessel gets 67.6x weight
        focal_loss = (alpha_t * focal_term).mean()

        # Dice component
        pred_probs = torch.sigmoid(pred)
        intersection = (pred_probs * target).sum(dim=(-2, -1))
        union = pred_probs.sum(dim=(-2, -1)) + target.sum(dim=(-2, -1))
        dice_loss = 1 - (2 * intersection + eps) / (union + eps)

        return focal_weight * focal_loss + dice_weight * dice_loss.mean()
```

8.4.3 Class Weights (Inverse Frequency)

```
# Computed from training set pixel counts
class_weights = {
    1: 1.0,     # tumor (33.8% of pixels) - baseline
    2: 1.3,     # stroma (26.2%)
    3: 5.7,     # lymphocyte (5.9%)
    4: 4.9,     # necrosis (6.9%)
    18: 67.6,   # blood_vessel (0.5%) - 67x upweight!
}
```

Calculation: weight = 1 / (class_frequency * num_classes)

8.5 LoRA (Low-Rank Adaptation)

8.5.1 Mathematical Foundation

Original linear layer: y = Wx + b

LoRA adds low-rank decomposition: y = Wx + b + (α/r) × BA × x

Where:

- A ∈ ℝ^(d_in × r) - projects to low-rank space
- B ∈ ℝ^(r × d_out) - projects back
- r << d_in, d_out (typically 4, 8, or 16)
- α - scaling factor

```
class LoRALinear(nn.Module):
    def __init__(self, original_linear, r=8, alpha=None):
        # Freeze original weights
        for param in original_linear.parameters():
            param.requires_grad = False

        # Add trainable LoRA matrices
        self.lora_A = nn.Parameter(torch.zeros(in_features, r))
        self.lora_B = nn.Parameter(torch.zeros(r, out_features))
        self.scaling = alpha / r

        # Initialize: A with Kaiming, B with zeros (starts as identity)
        nn.init.kaiming_uniform_(self.lora_A)
        nn.init.zeros_(self.lora_B)

    def forward(self, x):
        original = self.original_linear(x)
        lora = x @ self.lora_A @ self.lora_B * self.scaling
        return original + lora
```

8.5.2 Why LoRA for Medical Imaging?

1. **Prevents Catastrophic Forgetting**: Original weights frozen, can't be destroyed
2. **Parameter Efficient**: ~0.2% new parameters vs 100% for full finetuning
3. **Composable**: Can merge multiple LoRA adapters
4. **Reversible**: Remove adapter to restore original model

8.5.3 Where to Apply LoRA in SAM2

```
target_modules = {
    'image_encoder': Apply to Hiera attention QKV and projection layers,
    'mask_decoder': Apply to TwoWayTransformer attention layers,
    'memory_attention': Apply to memory attention layers,
}
```

Our Choice: image_encoder only (where most features are learned)

8.5.4 LoRA Configuration Comparison

Rank ®	Trainable Params	% of Total	Capacity
4	~1.1M	0.5%	Low
8	~2.2M	1.0%	Medium
16	~4.4M	2.0%	High

Our Setting: r=8, α=8 (balanced)

8.6 Mask Post-Processing

8.6.1 Morphological Operations

```
def postprocess_mask(mask, kernel_size=7, min_area_ratio=0.1):
    # 1. Closing: Fill small holes
    kernel = np.ones((kernel_size, kernel_size), np.uint8)
    closed = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

    # 2. Keep only largest connected component
    contours, _ = cv2.findContours(closed, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    largest = max(contours, key=cv2.contourArea)

    # 3. Create clean mask
    final = np.zeros_like(mask)
    cv2.drawContours(final, [largest], -1, 1, cv2.FILLED)

    return final
```

8.6.2 Operations Explained

Closing (Dilation → Erosion):

- Fills small holes and gaps
- Connects nearby regions
- Kernel size controls gap-filling range

Largest Component Filtering:

- Removes spurious predictions
- Assumes single object per class
- May discard valid but small structures

8.7 Evaluation Metrics

8.7.1 Dice Similarity Coefficient (DSC)

```
Dice = 2|P ∩ G| / (|P| + |G|)

where:
- P = predicted mask pixels
- G = ground truth mask pixels
- Range: [0, 1], higher is better
```

Interpretation:

- 0.0: No overlap
- 0.5: Partial overlap
- 0.7+: Good segmentation
- 0.9+: Excellent segmentation

Code:

```
def dice(pred, gt):
    intersection = np.logical_and(pred, gt).sum()
    return (2.0 * intersection) / (pred.sum() + gt.sum() + 1e-6)
```

8.7.2 Intersection over Union (IoU / Jaccard)

```
IoU = |P ∩ G| / |P ∪ G|
    = |P ∩ G| / (|P| + |G| - |P ∩ G|)
```

Relationship to Dice: Dice = 2 * IoU / (1 + IoU)

IoU is stricter than Dice (always ≤ Dice).

8.7.3 Classification Metrics

Precision: Of all predictions for class C, how many were correct?

```
Precision_C = TP_C / (TP_C + FP_C)
```

Recall: Of all actual class C samples, how many were predicted?

```
Recall_C = TP_C / (TP_C + FN_C)
```

F1 Score: Harmonic mean of precision and recall

```
F1_C = 2 * Precision_C * Recall_C / (Precision_C + Recall_C)
```

Macro F1: Average F1 across classes (treats all classes equally)

```
Macro_F1 = (1/C) * Σ F1_c
```

---

9. Key Findings

8.1 Main Conclusions

Finding 1: Zero-Shot Outperforms Finetuning

```
Zero-Shot SAM2:  0.555 Dice
Finetuned SAM2:  0.372 Dice (-33%)
```

Implication: For small medical datasets (n < 100), invest in prompt engineering rather than finetuning.

Finding 2: Prompt Type is Critical

```
Centroid:     0.338 Dice (baseline)
Multi-Point:  0.418 Dice (+24%)
Box:          0.553 Dice (+64%)
Box + Neg:    0.555 Dice (+64%)
```

Implication: Bounding boxes provide the most reliable spatial information for SAM2.

Finding 3: LLM Prompts Beat Manual Prompts

```
Manual Prompts:    42.2% Accuracy
LLM Few-Shot:      44.4% Accuracy (+2.2%)
```

Implication: LLMs can generate better CLIP prompts than domain experts.

Finding 4: CLIP Struggles with Medical Images

```
Overall Accuracy:  44.4%
Random Baseline:   20.0% (5 classes)
```

Implication: CLIP provides above-random classification but fails on subtle tissue types (lymphocytes, necrosis).

Finding 5: MedSAM Underperforms SAM2

```
SAM2 Hiera-L:  0.555 Dice
MedSAM ViT-B:  0.536 Dice (-3.4%)
```

Implication: Model capacity (224M vs 86M params) matters more than domain-specific pretraining.

8.2 Recommendations

For Practitioners

1. **Use SAM2 with box prompts** for histopathology segmentation
2. **Don't finetune** unless you have >1000 training images
3. **Use LLM-generated prompts** for CLIP classification
4. **Avoid medical jargon** in text prompts (CLIP trained on natural language)

For Researchers

1. **Develop medical-specific VFMs** with larger architectures
2. **Create prompt optimization methods** for medical domains
3. **Investigate hybrid approaches** combining VFMs with traditional methods
4. **Build larger annotated datasets** for finetuning experiments

10. Code Architecture

10.1 Repository Structure

```
vfm_project/
├── src/                          # Core source code
│   ├── dataset.py                # BCSS data loading and splits
│   ├── sam_segmentation.py       # SAM2 inference and prompting
│   ├── clip_classification.py    # CLIP inference
│   ├── evaluation.py             # Main evaluation pipeline
│   ├── focal_loss.py             # Custom loss functions
│   ├── lora_adapter.py           # LoRA implementation
│   ├── tta_utils.py              # Test-time augmentation
│   ├── finetune_dataset.py       # Training data loader
│   ├── evaluators/               # Evaluation scripts
│   ├── trainers/                 # Training scripts
│   └── prompt_generators/        # LLM prompt generation
│
├── conf/                         # Hydra configuration
│   ├── config.yaml               # Base config
│   └── experiment/               # Experiment-specific configs
│
├── configs/prompts/              # CLIP prompt files
│   ├── hard_coded_prompts_v2.json
│   └── llm_text_prompts_v3_fewshot.json
│
├── scripts/                      # Utility scripts
│   ├── analysis/                 # Figure generation
│   ├── slurm/                    # HPC job scripts
│   └── utils/                    # Download helpers
│
├── results/                      # Experiment outputs
│   ├── figures/                  # Generated plots
│   └── complete_metrics/         # JSON metrics
│
├── finetune_logs/                # Training outputs
│   ├── sam2_focal_50ep/
│   ├── sam2_lora_30ep/
│   └── sam2_base_100ep/
│
├── sam2/                         # SAM2 submodule
├── MedSAM/                       # MedSAM submodule
└── models/                       # Checkpoints
```

10.2 Key Classes and Functions

BCSSDataset (src/dataset.py)

```python
class BCSSDataset(Dataset):
    """
    PyTorch Dataset for BCSS images.

    Attributes:
        class_names: {0: 'background', 1: 'tumor', ...}
        target_class_ids: {1, 2, 3, 4, 18}

    Returns:
        {
            'image': torch.Tensor (C, H, W),
            'mask': torch.Tensor (H, W),
            'unique_classes': np.array,
            'filename': str,
            'image_np': np.array (H, W, C)
        }
    """
```

SAM2 Predictor (src/sam_segmentation.py)

```python
def get_sam2_predictor(model_cfg, checkpoint, device):
    """Initialize SAM2 predictor."""

def get_prompts_from_mask(binary_mask, num_points=5):
    """Generate prompts from ground truth mask."""

def get_predicted_mask_from_prompts(predictor, image, prompts,
                                     prompt_type, use_neg_points):
    """Run SAM2 inference with specified prompts."""

def calculate_metrics(pred_mask, gt_mask):
    """Calculate Dice and IoU scores."""
```

CLIP Classifier (src/clip_classification.py)

```python
class CLIPClassifier:
    def __init__(self, model_name=None, device=None):
        """Load CLIP model (local or HuggingFace)."""

    def classify_region(self, image, prompts):
        """Classify cropped region using text prompts."""
```

Evaluation Pipeline (src/evaluation.py)

```python
def run_evaluation(args):
    """
    Main evaluation loop.

    For each test image:
        For each class in image:
            1. Generate prompts from GT mask
            2. Run SAM2 segmentation
            3. Crop predicted region
            4. Run CLIP classification
            5. Record metrics

    Outputs:
        - metrics.json
        - confusion_matrix.png
    """
```

10.3 Configuration System (Hydra)

Base Config (conf/config.yaml)

```yaml
defaults:
  - experiment: base_finetune
  - _self_

data_root: data/bcss
output_dir: finetune_logs

hydra:
  run:
    dir: ${output_dir}/${hydra.job.name}
  job:
    name: ${experiment.name}-${now:%Y-%m-%d_%H-%M-%S}
```

Experiment Override (conf/experiment/sam2_box_focal.yaml)

```yaml
# @package _global_

experiment:
  name: sam2_box_focal

scratch:
  base_lr: 3.0e-05
  num_epochs: 50

trainer:
  model:
    freeze_image_encoder: true
    prob_to_use_box_input_for_train: 1.0

loss:
  all:
    _target_: src.focal_loss.FocalDiceLoss
```

Running Experiments

```bash
# Default config
python src/trainers/run_finetuning.py

# With experiment override
python src/trainers/run_finetuning.py experiment=sam2_box_focal

# With additional overrides
python src/trainers/run_finetuning.py experiment=sam2_lora_light \
    scratch.base_lr=1e-5 scratch.num_epochs=20
```

---

11. Reproduction Guide

11.1 Environment Setup

Prerequisites

- Python 3.10+
- CUDA 11.8+ (for GPU training)
- 16GB+ GPU memory (for SAM2 Hiera-L)

Installation

```bash
# Clone repository with submodules
git clone --recursive https://github.com/shubham-mhaske/vfm_project.git
cd vfm_project

# Create conda environment
conda create -n vfm python=3.10
conda activate vfm

# Install dependencies
pip install -r requirements.txt

# Download SAM2 checkpoints
bash sam2/checkpoints/download_ckpts.sh
```

Verify Installation

```python
import torch
from sam2.build_sam import build_sam2
from transformers import CLIPModel

# Check CUDA
print(f"CUDA available: {torch.cuda.is_available()}")

# Check SAM2
model = build_sam2("configs/sam2.1/sam2.1_hiera_l.yaml",
                   "sam2/checkpoints/sam2.1_hiera_large.pt")
print(f"SAM2 loaded: {sum(p.numel() for p in model.parameters())/1e6:.1f}M params")

# Check CLIP
clip = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
print(f"CLIP loaded: {sum(p.numel() for p in clip.parameters())/1e6:.1f}M params")
```

11.2 Running Experiments

Zero-Shot Evaluation

```bash
# SAM2 + CLIP evaluation
python src/evaluation.py \
    --sam_model_cfg configs/sam2.1/sam2.1_hiera_l.yaml \
    --sam_checkpoint sam2/checkpoints/sam2.1_hiera_large.pt \
    --clip_prompts configs/prompts/llm_text_prompts_v3_fewshot.json \
    --output_dir results/my_experiment

# MedSAM evaluation
python src/evaluators/evaluate_medsam.py \
    --checkpoint models/medsam_checkpoints/medsam_vit_b.pth \
    --use_tta \
    --output_dir results/medsam_eval
```

Finetuning

```bash
# Focal Loss finetuning
python src/trainers/run_finetuning.py experiment=sam2_box_focal

# LoRA finetuning
python src/trainers/run_finetuning.py experiment=sam2_lora_light

# Resume from checkpoint
python src/trainers/run_finetuning.py experiment=sam2_box_focal \
    resume_checkpoint=finetune_logs/sam2_focal_50ep/checkpoints/checkpoint_25.pt
```

Generate Figures

```bash
# Comparison figures
python scripts/analysis/generate_comparison_figures.py

# Training analysis
python scripts/analysis/generate_training_analysis.py

# Output: results/figures/
```

11.3 SLURM (HPC) Submission

Example SLURM Script

```
#!/bin/bash
#SBATCH --job-name=sam2_eval
#SBATCH --partition=gpu
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --gres=gpu:a100:1
#SBATCH --mem=64G
#SBATCH --time=4:00:00

module load Python/3.10
source activate vfm

python src/evaluation.py \
    --sam_model_cfg configs/sam2.1/sam2.1_hiera_l.yaml \
    --sam_checkpoint sam2/checkpoints/sam2.1_hiera_large.pt \
    --clip_prompts configs/prompts/llm_text_prompts_v3_fewshot.json \
    --output_dir results/hprc_eval
```

Submit Job

```
sbatch scripts/slurm/run_evaluation.slurm
```

11.4 Expected Results

After running evaluation, you should see:

```
results/my_experiment/
├── metrics.json          # All metrics
├── confusion_matrix.png  # Classification confusion matrix
└── per_image_results/    # Optional per-image outputs
```

metrics.json Structure

```
{
    "config": {
        "sam_prompt_type": "box",
        "use_neg_points": true,
        "clip_prompts": "llm_text_prompts_v3_fewshot.json"
    },
    "segmentation": {
        "avg_dice": 0.555,
        "dice_std": 0.193,
        "avg_iou": 0.408,
        "total_samples": 181
    },
    "classification": {
        "accuracy": 0.444,
        "per_class": {
            "tumor": {"precision": 0.974, "recall": 0.822, "f1_score": 0.892},
            ...
        }
    }
}
```

Appendix A: Complete Experiment Catalog

This appendix documents **ALL 17 experiment configurations** ever created for this project, including their complete hyperparameters, rationale, and outcomes.

A.1 Base Configurations

A.1.1 base_finetune.yaml

Purpose: Initial finetuning configuration - full model training baseline

Parameter	Value	Rationale
Batch Size	4	Smaller batch = more steps per epoch
Learning Rate	2e-5	Scaled down for smaller batch
Vision LR	2e-6	0.1× base_lr for backbone
Epochs	150	More epochs for small dataset (~3150 steps)
Freeze Encoder	❌ No	Full model training
Loss	MultiStepMultiMasksAndIous	Standard SAM2 loss
Prompt Type	Mixed	Points + Boxes

Data Augmentation:

```
RandomHorizontalFlip: true
RandomVerticalFlip: true
RandomAffine: 90° rotation
ColorJitter:
  brightness: 0.3
  contrast: 0.3
  saturation: 0.3
  hue: 0.08
```

Loss Weights:

```
loss_mask: 20
loss_dice: 1
loss_iou: 1
loss_class: 1
```

LR Schedule: Cosine decay from 2e-5 → 1e-6

Outcome: Baseline for comparison. Full model training led to overfitting.

A.1.2 base_finetune_v2_stable.yaml

Purpose: Stabilized version with warmup and frozen encoder

Parameter	Value	Change from v1
Batch Size	6	↑ from 4
Learning Rate	6e-5	Scaled for batch 6 ($\sqrt{(6/4)} \times 5e\text{-}5$)
Epochs	100	↓ from 150 (best was at epoch 100)
Warmup	12%	NEW - Linear warmup
Freeze Encoder	✅ Yes	NEW - Only train decoder
Weight Decay	0.01	↓ from 0.05
Drop Last	true	Changed for consistent batch sizes

Data Augmentation Changes:

```
ColorJitter:
  brightness: 0.2  # ↓ from 0.3 (milder)
  contrast: 0.2    # ↓ from 0.3
  saturation: 0.2  # ↓ from 0.3
  hue: 0.05        # ↓ from 0.08
```

Loss Weight Changes:

```
loss_mask: 5   # ↓ from 20 (better balance)
loss_dice: 2   # ↑ from 1 (emphasize boundary)
loss_iou: 1
loss_class: 1
```

LR Schedule: Composite scheduler

- 12% Linear warmup: 0 → 6e-5
- 88% Cosine decay: 6e-5 → 6e-6

Outcome: More stable training, but still underperformed zero-shot.

A.1.3 base_finetune_v3_perclass.yaml

Purpose: Per-class validation tracking with gradual unfreezing

Parameter	Value	Change from v2
Learning Rate	8e-5	↑ for gradual unfreezing
Epochs	40	↓ (early stopping expected)
Warmup	20%	↑ from 12% (longer for stability)
Weight Decay	0.02	↑ for better generalization

Data Augmentation Changes:

```
ColorJitter:
  brightness: 0.3  # ↑ back to original
  contrast: 0.3
  saturation: 0.3
  hue: 0.1         # ↑ wider range for stain artifacts
```

LR Schedule: Composite scheduler

- 20% Linear warmup: 0 → 8e-5
- 80% Cosine decay: 8e-5 → 1e-6

Outcome: Designed for per-class tracking. Requires custom validation loop.

A.2 Box Prompting Strategies

A.2.1 sam2_box_focal.yaml

Purpose: Box-only prompts with class-weighted Focal+Dice loss

Hypothesis: Point prompts are ambiguous for histopathology. Bounding boxes provide clearer region definitions.

Parameter	Value	Rationale
Batch Size	4	
Learning Rate	3e-5	Half of v2 (moderate)
Vision LR	0.0	Completely frozen encoder
Epochs	50	
Warmup	15%	
Prompt Type	box	Box prompts ONLY
Negative Points	❌	Not needed for box

Loss Function: `src.focal_loss.FocalDiceLoss`

```
alpha:  # Class weights (inverse frequency)
  1: 1.0    # tumor (baseline)
  2: 1.3    # stroma
  3: 5.7    # lymphocyte
  4: 4.9    # necrosis
  18: 67.6  # blood_vessel (67× upweight!)
gamma: 2.0        # Focal focusing parameter
dice_weight: 3.0
focal_weight: 1.0
```

Key Model Settings:

```
prob_to_use_pt_input_for_train: 0.0   # NO point prompts
prob_to_use_box_input_for_train: 1.0  # ALWAYS box prompts
```

Result: Test Dice 0.372 (vs 0.555 zero-shot)

A.2.2 sam2_box_simple.yaml

Purpose: Box prompts with standard loss (isolate prompt type effect)

Inherits from: `base_finetune_v2_stable`

Parameter	Value	Override
Prompt Type	box	Changed from mixed
Negative Points	❌	Disabled
Loss	MultiStepMultiMasksAndIous	Standard (not Focal)

Outcome: Tests whether box prompts alone help without Focal loss.

A.3 LoRA (Low-Rank Adaptation)

A.3.1 sam2_lora.yaml

Purpose: Full LoRA configuration for parameter-efficient finetuning

Concept: Add small trainable matrices to frozen pretrained weights:

```
Output = Original + (α/r) × B @ A @ x
```

Parameter	Value	Rationale
LoRA Rank ®	8	Balance between capacity and efficiency
LoRA Alpha	8.0	Typically same as rank
LoRA Dropout	0.1	Regularization
Target Modules	image_encoder	Apply LoRA to encoder
Trainable Head	✅ Yes	Keep final MLP heads trainable

Training Config:

```
batch_size: 4
lr: 1e-4         # Higher LR OK for fewer params
weight_decay: 0.01
epochs: 20
warmup_steps: 100  # Shorter warmup
```

Class Weights:

```
tumor: 1.0
stroma: 1.3
lymphocyte: 5.7
necrosis: 4.9
blood_vessel: 67.6
```

A.3.2 sam2_lora_light.yaml ⭐ (Key Experiment)

Purpose: Minimal adaptation to preserve pretrained knowledge

Philosophy: "Less is more" - extremely gentle adaptation to avoid destroying zero-shot capability.

Parameter	Value	Rationale
Batch Size	4	
Learning Rate	5e-6	12× SMALLER than v2
Vision LR	0.0	COMPLETELY frozen
Epochs	30	SHORT training
Warmup	25%	LONGER warmup for stability
Weight Decay	0.001	VERY LOW

Minimal Augmentation:

```
ColorJitter:
  brightness: 0.1  # HALVED from v2
  contrast: 0.1
  saturation: 0.1
  hue: 0.02        # MINIMAL hue shift
# NO rotation (preserve pretrained orientation knowledge)
```

Loss Weights:

```
loss_mask: 2   # REDUCED (less aggressive)
loss_dice: 5   # INCREASED (emphasize shape/boundary)
loss_iou: 2    # INCREASED (better overlap)
loss_class: 1
```

LR Schedule:

- 25% Linear warmup: 0 → 5e-6
- 75% Cosine decay: 5e-6 → 1e-7

Training Curves:

```
Epoch 0:  Loss=36.6, Mask=14.3, Dice=0.96
Epoch 5:  Loss=11.3, Mask=3.2,  Dice=0.87
Epoch 10: Loss=4.3,  Mask=0.1,  Dice=0.74
Epoch 20: Loss=4.0,  Mask=0.1,  Dice=0.72
Epoch 30: Loss=4.0,  Mask=0.1,  Dice=0.72
```

Result: Test Dice 0.355 (fast convergence but still worse than zero-shot)

A.4 Path-SAM2 Configurations (Dual-Encoder)

A.4.1 path_sam2_ctranspath.yaml

Purpose: Integrate CTransPath (histopathology-pretrained Swin) with SAM2

Architecture: SAM2 Hiera-L (256-dim) + CTransPath Swin (768-dim) → Fusion → Decoder

CTransPath: Swin Transformer pretrained on 15M histopathology patches (open-source)

Parameter	Value	Rationale
Batch Size	4	Reduced for dual-encoder memory
Learning Rate	5e-5	
Epochs	50	
Warmup	15%	
CTransPath Freeze	❌ No	Train both encoders
Fusion Type	attention	Learn attention-based fusion
Gradient Clip	1.0	↑ from 0.1 (higher for stability)
Distributed Backend	gloo	Required for multi-encoder
AMP	❌ Off	Stability during fusion training

CTransPath Config:

```
ctranspath:
  checkpoint: models/ctranspath/ctranspath.pth
  freeze: false
  embed_dim: 768
  fusion_type: attention
```

Loss Weights:

```
loss_mask: 5
loss_dice: 3  # Increased for boundary quality
loss_iou: 1
loss_class: 1
```

Outcome: Designed for domain-specific feature injection.

A.4.2 path_sam2_ctranspath_optimized.yaml

Purpose: Optimized version of CTransPath integration

Parameter	Value	Change
Batch Size	6	↑ from 4
Workers	12	↑ from 8
Learning Rate	6e-5	↑ from 5e-5
Epochs	40	↓ from 50
Warmup	25%	↑ from 15%
CTransPath Freeze	✅ Yes	Only train fusion
Fusion Type	concat	Changed from attention
Distributed Backend	nccl	Changed from gloo
AMP	✅ On	Enabled for speed

Outcome: Faster training with frozen CTransPath.

A.4.3 path_sam2_focal.yaml

Purpose: CTransPath + Focal Loss combination

Inherits from: `path_sam2_ctranspath_optimized`

Parameter	Value
Loss	`src.focal_loss.FocalDiceLoss`
Class Weights	Same as sam2_box_focal
Gamma	2.0
Dice Weight	2.0
Focal Weight	1.0

A.4.4 path_sam2_focal_stain.yaml ⭐ (Advanced)

Purpose: CTransPath + Focal Loss + H&E Stain Augmentation

Inherits from: `path_sam2_focal`

Key Addition: Stain normalization and augmentation for H&E variation handling

```yaml
transforms:
  - _target_: src.stain_augmentation.StainAugmentationTransform
    normalize: true   # Macenko stain normalization
    augment: true     # Stain transfer augmentation
  - RandomHorizontalFlip
  - RandomVerticalFlip
  - RandomAffine: 90°
  - ColorJitter: ...
```

Stain Augmentation Benefits:

1. Normalizes H&E color variation across labs
2. Augments with different staining protocols
3. Improves generalization to unseen staining

A.4.5 path_sam2_uni_fusion.yaml

Purpose: UNI encoder (1024-dim) fusion with SAM2

Note: Requires UNI model access (originally restricted, now open)

Parameter	Value
UNI Checkpoint	models/uni/pytorch_model.bin
UNI Freeze	✅ Yes
UNI Embed Dim	1024
Batch Size	4 (reduced for memory)
Warmup	15% (longer for dual-encoder)

Implementation Note:

```
# Requires modifications to run_finetuning.py:
# 1. Load UNI encoder
# 2. Replace SAM2 image encoder with PathSAM2Encoder
# 3. Train only fusion module + decoder
```

A.5 Ablation Experiments

A.5.1 histology_optimized.yaml

Purpose: Optimized configuration for histopathology/BCSS

Inherits from: `base_finetune`

Parameter	Value	Rationale
Learning Rate	2e-5	Conservative for medical
Vision LR	2e-6	10× smaller for backbone
Epochs	150	More for small dataset
LR End Value	1e-7	Very small final LR

Augmentation (same as base_finetune):

```
ColorJitter:
  brightness: 0.3
  contrast: 0.3
  saturation: 0.3
  hue: 0.08
```

A.5.2 strong_aug.yaml

Purpose: Test aggressive augmentation impact

Inherits from: `base_finetune`

Parameter	Value	Change
Brightness	0.4	↑ from 0.3
Contrast	0.4	↑ from 0.3
Saturation	0.4	↑ from 0.3
Hue	0.1	↑ from 0.08

Outcome: Test whether more augmentation helps generalization.

A.5.3 low_lr.yaml

Purpose: Test lower learning rate

Inherits from: `base_finetune`

Parameter	Value	Change
Base LR	1e-5	↓ from 2e-5
Vision LR	1e-6	↓ from 2e-6

A.6 Testing/Debugging Configurations

A.6.1 dry_run.yaml

Purpose: Quick local CPU testing

Inherits from: `base_finetune`

Parameter	Value	Rationale
Epochs	1	Minimal
Batch Size	1	Single sample
Workers	0	Avoid multiprocessing
Accelerator	cpu	No GPU needed
Backend	gloo	CPU-compatible
AMP	❌ Off	CPU mode

Model Reduction:

```yaml
# Uses Hiera-Tiny instead of Hiera-L
trunk:
  embed_dim: 96
  num_heads: 1
  stages: [1, 2, 7, 2]
checkpoint: sam2.1_hiera_tiny.pt
```

A.6.2 local_test.yaml

Purpose: Test path_sam2_focal_stain locally

Inherits from: `path_sam2_focal_stain`

Parameter	Value
Epochs	1
Batch Size	1
Workers	0
Accelerator	cpu
Backend	gloo
AMP	❌ Off

A.7 Configuration Comparison Table

Config	Encoder	Epochs	LR	Loss	Prompt	Augmentation
base_finetune	❌ Train	150	2e-5	Standard	Mixed	Strong
base_finetune_v2	✅ Freeze	100	6e-5	Standard	Mixed	Mild
base_finetune_v3	✅ Freeze	40	8e-5	Standard	Mixed	Strong
sam2_box_focal	✅ Freeze	50	3e-5	Focal+Dice	Box	Moderate
sam2_box_simple	✅ Freeze	100	6e-5	Standard	Box	Mild
sam2_lora	LoRA	20	1e-4	Focal	Mixed	Moderate
sam2_lora_light	✅ Freeze	30	5e-6	Standard	Mixed	Minimal
path_sam2_ctranspath	❌ Train	50	5e-5	Standard	Mixed	Mild
path_sam2_optimized	✅ Freeze	40	6e-5	Standard	Mixed	Mild
path_sam2_focal	✅ Freeze	40	6e-5	Focal+Dice	Mixed	Mild
path_sam2_focal_stain	✅ Freeze	40	6e-5	Focal+Dice	Mixed	Stain+Mild
path_sam2_uni_fusion	✅ Freeze	50	5e-5	Standard	Mixed	Mild
histology_optimized	❌ Train	150	2e-5	Standard	Mixed	Strong
strong_aug	❌ Train	150	2e-5	Standard	Mixed	Very Strong
low_lr	❌ Train	150	1e-5	Standard	Mixed	Strong
dry_run	❌ Train	1	2e-5	Standard	Mixed	Strong
local_test	✅ Freeze	1	6e-5	Focal+Dice	Mixed	Stain+Mild

A.8 Experiment Outcomes Summary

Experiment	Test Dice	vs Zero-Shot	Key Learning
Zero-Shot (Box+Neg)	**0.555**	—	Baseline
base_finetune	0.371	-33%	Overfits with full training
base_finetune_v2_stable	0.371	-33%	Warmup doesn't help
sam2_box_focal	0.372	-33%	Focal loss marginal help
sam2_lora_light	0.355	-36%	LoRA also fails
path_sam2_focal	0.372	-33%	CTransPath no benefit

Conclusion: All finetuning approaches failed. Zero-shot remains best.

A.9 Complete Training Curves

SAM2 Focal Loss Training (50 epochs)

Epoch	Total Loss	Focal Loss	Dice Loss	Train Steps
0	45.42	42.61	0.940	21
5	62.11	59.66	0.818	126
10	38.53	36.22	0.769	231
15	31.65	29.45	0.732	336
20	34.07	31.86	0.734	441
25	33.96	31.80	0.719	546
30	30.44	28.26	0.728	651
35	30.59	28.43	0.720	756
40	32.70	30.66	0.681	861
45	40.30	38.17	0.708	966
49	28.90	26.82	0.696	1050

Observations:

- Loss decreases from 45.4 to 28.9 (36% reduction)
- Dice loss decreases from 0.94 to 0.70 (26% reduction)
- Training loss volatile (high variance between epochs)
- Despite low training loss, test Dice (0.372) is worse than zero-shot (0.555)

SAM2 LoRA Light Training (30 epochs)

Epoch	Total Loss	Mask Loss	Dice Loss	IoU Loss	Class Loss
0	36.65	14.29	0.961	0.230	2.807
5	11.33	3.18	0.866	0.109	0.415
7	6.11	0.95	0.764	0.119	0.159
10	4.27	0.13	0.737	0.140	0.034
15	4.11	0.10	0.730	0.129	0.000
20	4.16	0.15	0.726	0.112	0.003
25	4.06	0.09	0.729	0.115	0.002
29	4.04	0.09	0.723	0.119	0.002

Observations:

- Very fast convergence: epoch 7 already near minimum
- Mask loss drops from 14.29 to 0.09 (99.4% reduction)
- Class loss drops from 2.81 to 0.002 (99.9% reduction)
- Total loss plateaus at ~4.0 from epoch 10 onward
- Despite near-perfect training metrics, test Dice (0.355) is worst

Key Insight: Fast convergence + low training loss + poor test performance = classic overfitting to small dataset.

Appendix B: Complete Prompt Files & Generation Scripts

This appendix documents ALL 8 prompt files and ALL 6 generation scripts used in our experiments.

B.1 Prompt Files Overview

File	Type	Generator	Classes	Prompts/Class	CLIP Accuracy
`hard_coded_prompts.json`	Manual v1	Human expert	6	5	23.3%
`hard_coded_prompts_v2.json`	Manual v2	Human expert	5	5	**42.2%**
`llm_text_prompts_v1_gemini_pro_latest.json`	LLM Text	Gemini Pro	6	7	12.2%
`llm_text_prompts_v2_clip_friendly.json`	LLM Text	Gemini Pro	5	7	35.6%
`llm_text_prompts_v3_fewshot.json`	LLM Text	Gemini Pro	5	7	**44.4%**
`llm_multimodal_prompts_v1_gemini_2.5_flash.json`	LLM VLM	Gemini Flash	6	7	8.3%
`llm_multimodal_prompts_v2_clip_friendly.json`	LLM VLM	Gemini Flash	5	7	15.0%
`llm_multimodal_prompts_v3_fewshot.json`	LLM VLM	Gemini Flash	5	7	29.4%

B.2 Manual (Hardcoded) Prompts

B.2.1 hard_coded_prompts.json (v1 - Medical Jargon)

Approach: Domain expert wrote prompts using pathology terminology.
Result: 23.3% accuracy - CLIP doesn’t understand medical jargon.

```
{
    "tumor": [
        "a histopathology image of a tumor",
        "cancerous tissue with malignant cells",
        "a dense cluster of large, irregularly shaped cells with dark nuclei",
        "invasive ductal carcinoma cells",
        "a region of neoplastic cells"
    ],
    "stroma": [
        "a histopathology image of stroma",
        "connective tissue supporting the tumor",
        "spindle-shaped cells with elongated nuclei",
        "fibrous tissue surrounding cancer cells",
        "a region of desmoplastic stroma"
    ],
    "lymphocyte": [
        "a histopathology image of lymphocytes",
        "a cluster of immune cells",
        "small, round cells with dark, circular nuclei and minimal cytoplasm",
        "an infiltration of lymphocytes",
        "a region of immune response in tissue"
    ],
    "necrosis": [
        "a histopathology image of necrosis",
        "dead tissue, often with fragmented cells and debris",
        "an area of cell death with loss of cell structure",
        "necrotic core of a tumor",
        "a region of eosinophilic, anucleated cells"
    ],
    "blood_vessel": [
        "a histopathology image of a blood vessel",
        "a channel for blood flow lined by endothelial cells",
        "a cross-section of a capillary or arteriole",
        "a vessel containing red blood cells",
        "vascular structure in tissue"
    ],
    "background": [
        "a histopathology image of background tissue",
        "adipose tissue or empty space",
        "fat cells and slide background",
        "a region with no distinct cellular structures",
        "normal, non-cancerous tissue"
    ]
}
```

Why It Failed:

- Terms like “desmoplastic”, “eosinophilic”, “neoplastic” not in CLIP’s training
- Generic prefixes “a histopathology image of” add noise
- No visual color/texture descriptions

B.2.2 hard_coded_prompts_v2.json (v2 - Visual Descriptions)

Approach: Rewrote prompts focusing on H&E stain colors and visual patterns.
Result: 42.2% accuracy (+81% improvement over v1)

```
{
    "tumor": [
        "densely crowded dark purple cells packed together",
        "large irregular purple nuclei in chaotic arrangement",
        "thick masses of deep purple tissue with high cell density",
        "disorganized clusters of dark purple cells",
        "purple nuclei dominating the image with minimal space between cells"
    ],
    "stroma": [
        "bright pink wavy fibers forming streaming patterns",
        "light pink collagen with scattered thin elongated nuclei",
        "parallel bundles of pink fibrous tissue",
        "smooth bright pink background with sparse dark spindle nuclei",
        "thread-like pink fibers running in wavy patterns"
    ],
    "lymphocyte": [
        "many tiny dark blue dots scattered throughout",
        "small uniform round purple circles densely clustered",
        "dense groups of tiny dark blue spheres",
        "scattered small dark purple dots of equal size",
        "clusters of tiny uniform blue-purple cells"
    ],
    "necrosis": [
        "pale ghostly pink tissue with faded blurry appearance",
        "washed out pale areas with fragmented cell debris",
        "smudgy faint pink tissue losing structure",
        "ghostly pale region with indistinct borders",
        "faded pink areas with scattered dark fragments"
    ],
    "blood_vessel": [
        "circular empty white space surrounded by thin pink wall",
        "ring-shaped hollow area with pink rim",
        "round opening with thin pink tissue lining the edge",
        "tube-like hollow structure with smooth pink walls",
        "circular white lumen with delicate pink border"
    ]
}
```

Why It Succeeded:

- **Color focus**: "dark purple", "bright pink", "pale pink", "dark blue"
- **Size descriptors**: "tiny", "large", "thin"
- **Texture words**: "crowded", "wavy", "scattered", "dense"
- **Removed background class**: Reduced confusion

B.3 LLM Text-Only Prompts (Gemini Pro)

B.3.1 Generation Script: generate_text_prompts.py (v1)

Meta-prompt sent to Gemini:

```
prompt_text = (
    f"You are an expert computational pathologist. I need to generate text prompts for a\n"
    f"CLIP vision-language model to classify breast cancer histology images.\n\n"
    f"For the tissue class '{class_name}', please generate 7 short, descriptive, \n"
    f"and visually specific phrases.\n\n"
    f"Rules:\n"
    f"1.  Focus *only* on the visual characteristics (cell shape, nucleus, color, texture, arrangement).\n"
    f"2.  Each of the 7 phrases must be on its own line.\n"
    f"3.  Do not use bullets, numbering, or JSON formatting. Just plain text, one line per phrase.\n"
    f"4.  Do not use generic phrases like 'a photo of' or 'an image of'."
)
```

Output: `llm_text_prompts_v1_gemini_pro_latest.json` (12.2% accuracy)

Problem: Despite asking for "visual characteristics", Gemini defaulted to medical jargon.

B.3.2 Generation Script: generate_text_prompts_v2.py (CLIP-Friendly)

Enhanced Meta-prompt with explicit examples:

```
prompt_text = (
    f"You are helping create text descriptions for a CLIP vision-language model to classify "
    f"H&E-stained breast cancer histology images. CLIP was trained on natural images and everyday language, "
    f"NOT medical textbooks.\n\n"
    f"For the tissue class '{class_name}', generate 7 SHORT, visually descriptive phrases.\n\n"
    f"CRITICAL RULES:\n"
    f"1. Use SIMPLE, EVERYDAY language (avoid medical jargon like 'pleomorphic', 'hyperchromatic', 'desmoplastic').\n"
    f"2. Emphasize COLORS from H&E staining: dark purple/blue (nuclei), bright pink (collagen/stroma), pale pink (necrosis).\n"
    f"3. Emphasize SIZE and SHAPE: 'tiny dots', 'large irregular', 'wavy fibers', 'circular hollow'.\n"
    f"4. Emphasize TEXTURE and ARRANGEMENT: 'densely packed', 'scattered', 'crowded', 'sparse', 'wavy patterns'.\n\n"
    f"EXAMPLES OF GOOD PROMPTS:\n"
    f"- 'densely crowded dark purple cells packed together'\n"
    f"- 'bright pink wavy fibers forming streaming patterns'\n"
    f"- 'many tiny dark blue dots scattered throughout'\n\n"
    f"EXAMPLES OF BAD PROMPTS:\n"
    f"- 'infiltrating nests of pleomorphic epithelial cells' ❌\n"
    f"- 'desmoplastic stroma with hyalinized collagen' ❌"
)
```

Output: `llm_text_prompts_v2_clip_friendly.json` (35.6% accuracy)

B.3.3 Generation Script: generate_text_prompts_v3_fewshot.py (Few-Shot)

Innovation: Provide successful v2 manual prompts as in-context examples:

```
# Proven successful prompts from manual v2 (42.2% accuracy)
SUCCESSFUL_EXAMPLES = {
    "tumor": [
        "densely crowded dark purple cells packed together",
        "large irregular purple nuclei in chaotic arrangement",
        "thick masses of deep purple tissue with high cell density"
    ],
    # ... (all 5 classes)
}

# Few-shot meta-prompt includes contrastive examples
prompt_text = f"""You are creating CLIP-friendly prompts for H&E breast cancer histology.

TASK: Generate 5 NEW prompts for the class '{class_name}' that follow the EXACT STYLE of these PROVEN SUCCESSFUL examples:

SUCCESSFUL EXAMPLES for '{class_name}':
1. "{examples[0]}"
2. "{examples[1]}"
3. "{examples[2]}"

CONTRAST WITH OTHER CLASSES (what NOT to match):
{other_classes_examples}

WHAT MAKES THESE EXAMPLES WORK:
- They use CONCRETE visual descriptors (colors, sizes, shapes, textures)
- They combine multiple features that TOGETHER uniquely identify the class
"""
```

Output: `llm_text_prompts_v3_fewshot.json` (**44.4% accuracy** - BEST)

Key Innovation:

1. Provide 3 proven examples per class as few-shot demonstrations
2. Include contrastive examples from OTHER classes
3. Keep original successful examples + add 4 new LLM-generated ones
4. Result: 7 prompts per class, 44.4% accuracy

B.4 LLM Multimodal Prompts (Gemini Flash VLM)

B.4.1 Generation Script: generate_multimodal_prompts.py (v1)

Approach: Send actual tissue images to Gemini Flash, ask for descriptions.

```
# Get 5 example images per class from training set
example_images = get_example_images(class_name, class_id, dataset, n_examples=5)

# Meta-prompt for image analysis
meta_prompt_text = f"""
You are an expert computational pathologist. Look at the {len(example_images)} example images of '{class_name}' tissue.

Based *only* on these images, generate 7 short, distinct, and visually specific phrases.

Rules:
1.  Focus *only* on the visual characteristics (cell shape, nucleus, color, texture, arrangement).
2.  Each of the 7 phrases must be on its own line.
3.  Do not use bullets or numbering.
"""

# Send images + text to Gemini
api_prompt_content = example_images + [meta_prompt_text]
response = model.generate_content(api_prompt_content)
```

Output: `llm_multimodal_prompts_v1_gemini_2.5_flash.json` (8.3% accuracy - WORST)

Generated Prompts (problematic):

```
{
    "tumor": [
        "Complex branching papillary structures lined by atypical epithelium.",
        "Foci of squamous differentiation forming concentric keratin pearls.",
        "Highly pleomorphic nuclei with coarse, vesicular chromatin."
    ]
}
```

Why It Failed:

- Gemini defaulted to medical textbook language
- Described architectural patterns CLIP can't understand
- "Papillary", "pleomorphic", "vesicular" outside CLIP vocabulary

B.4.2 Generation Script: generate_multimodal_prompts_v2.py (CLIP-Friendly)

Enhanced meta-prompt with explicit anti-jargon rules:

```
meta_prompt_text = f"""
You are helping create visual descriptions for a CLIP vision-language model. CLIP was trained on everyday images and simple language, NOT medical l

Look at these {len(example_images)} example images of '{class_name}' tissue (H&E staining). Generate 7 SHORT phrases describing what you SEE.

CRITICAL RULES:
1. Use SIMPLE, EVERYDAY words a non-expert would understand.
2. Describe COLORS you see: dark purple, bright pink, pale pink, blue dots, white spaces.
3. Describe SIZES and SHAPES: tiny, large, round, irregular, wavy, circular.
4. NO medical jargon (avoid: pleomorphic, hyperchromatic, desmoplastic, karyorrhectic).

EXAMPLES OF GOOD DESCRIPTIONS:
- "densely packed dark purple cells with minimal space"
- "bright pink wavy fibers running in patterns"

EXAMPLES OF BAD DESCRIPTIONS:
- "infiltrating nests of atypical cells"  ❌
"""
```

Output: `llm_multimodal_prompts_v2_clip_friendly.json` (15.0% accuracy)

Still problematic - Gemini couldn't fully suppress architectural language:

```
{
    "tumor": [
        "Densely packed clusters of dark purple cells",
        "Jagged and irregular branching patterns of tissue",  // ❌ Architectural
        "Large dark purple round cells with tiny white circles inside"
    ]
}
```

B.4.3 Generation Script: generate_multimodal_prompts_v3_fewshot.py (Few-Shot)

Innovation: Combine images + proven successful text prompts as examples:

```
# Send fewer images (3) but with successful prompts as examples
NUM_EXAMPLE_IMAGES = 3

# Include proven prompts in meta-prompt
meta_prompt_text = f"""Look at these {len(example_images)} example images of '{class_name}' tissue.

PROVEN SUCCESSFUL PROMPTS for '{class_name}' (40.4% accuracy):
1. "{examples[0]}"
2. "{examples[1]}"
3. "{examples[2]}"

TASK: Generate 5 NEW prompts that:
1. Follow the EXACT STYLE of the successful examples above
2. Describe what you SEE in the images using SIMPLE color/shape/texture words

AVOID (lessons from failed v2):
❌ Architectural language ("branching patterns", "lace-like")
❌ Generic descriptors that apply to multiple classes
"""
```

Output: `llm_multimodal_prompts_v3_fewshot.json` (29.4% accuracy)

Improved but still lagging text-only:

```
{
    "tumor": [
        "densely crowded dark purple cells packed together",  // Kept from examples
        "large irregular purple nuclei in chaotic arrangement",
        "thick masses of deep purple tissue with high cell density",
        "Large dark purple nuclei showing significant variation in shape.",  // New
        "Dense sheets of purple cells with minimal pale pink cytoplasm visible."
    ]
}
```

B.5 Prompt Generation Pipeline Summary

```
┌─────────────────────────────────────────────────────────────────────┐
│                     PROMPT GENERATION STRATEGIES                     │
├─────────────────────────────────────────────────────────────────────┤
│                                                                      │
│  MANUAL (Human Expert)                                               │
│  ├── v1 Medical Jargon → 23.3% (Failed)                             │
│  └── v2 Visual Colors  → 42.2% ✓                                    │
│                                                                      │
│  LLM TEXT-ONLY (Gemini Pro)                                         │
│  ├── v1 Basic prompt        → 12.2% (Jargon problem)                │
│  ├── v2 Anti-jargon rules   → 35.6% (Improved)                      │
│  └── v3 Few-shot examples   → 44.4% ✓ BEST                          │
│                                                                      │
│  LLM MULTIMODAL (Gemini Flash + Images)                             │
│  ├── v1 Basic + images      → 8.3%  (Worst - full jargon)           │
│  ├── v2 Anti-jargon + images→ 15.0% (Still architectural)           │
│  └── v3 Few-shot + images   → 29.4% (Better but lags text)          │
│                                                                      │
└─────────────────────────────────────────────────────────────────────┘
```

B.6 Key Findings: Why Text-Only LLM Beat Multimodal

Factor	Text-Only LLM	Multimodal LLM
Focus	Abstract class concept	Specific image artifacts
Language	Can follow “no jargon” rule	Reverts to textbook language
Generalization	Describes typical appearance	Describes training images
Noise	No visual distractions	Picks up irrelevant patterns
Control	Easy to guide with examples	Images override text guidance

Critical Insight: When Gemini sees actual histology images, it activates medical training and produces jargon. Text-only prompts allow better control over vocabulary.

B.7 Complete Prompt Files Location

```
configs/prompts/
├── hard_coded_prompts.json              # Manual v1 (medical jargon) - 23.3%
├── hard_coded_prompts_v2.json           # Manual v2 (visual colors) - 42.2%
├── llm_text_prompts_v1_gemini_pro_latest.json    # LLM text v1 - 12.2%
├── llm_text_prompts_v2_clip_friendly.json        # LLM text v2 - 35.6%
├── llm_text_prompts_v3_fewshot.json              # LLM text v3 - 44.4% ✓ BEST
├── llm_multimodal_prompts_v1_gemini_2.5_flash.json  # VLM v1 - 8.3%
├── llm_multimodal_prompts_v2_clip_friendly.json     # VLM v2 - 15.0%
└── llm_multimodal_prompts_v3_fewshot.json           # VLM v3 - 29.4%

src/prompt_generators/
├── generate_text_prompts.py           # Text v1 generator
├── generate_text_prompts_v2.py        # Text v2 generator (anti-jargon)
├── generate_text_prompts_v3_fewshot.py # Text v3 generator (few-shot)
├── generate_multimodal_prompts.py      # VLM v1 generator
├── generate_multimodal_prompts_v2.py   # VLM v2 generator (anti-jargon)
└── generate_multimodal_prompts_v3_fewshot.py  # VLM v3 generator (few-shot)
```

B.8 How to Generate New Prompts

Text-Only (Recommended)

```
export GEMINI_API_KEY="your-key-here"
cd vfm_project
python src/prompt_generators/generate_text_prompts_v3_fewshot.py
```

Multimodal (Experimental)

```
export GEMINI_API_KEY="your-key-here"
cd vfm_project
python src/prompt_generators/generate_multimodal_prompts_v3_fewshot.py
```

Evaluate New Prompts

```
python src/evaluation.py \
    --clip_prompts configs/prompts/YOUR_NEW_PROMPTS.json \
    --output_dir results/your_experiment
```

Appendix C: Complete Metrics

C.1 SAM2 Segmentation (All Configurations)

Config	Dice	Std	IoU	Std	Samples
Centroid	0.338	0.263	0.236	0.212	180
Multi-Point	0.418	0.209	0.287	0.178	170
Box	0.553	0.195	0.407	0.188	181
Box+Neg	0.555	0.193	0.408	0.185	181

C.2 MedSAM Segmentation

Config	Dice	Std	IoU	Samples
Box	0.522	0.189	0.375	181
Box+TTA	0.536	0.191	0.389	181

C.3 CLIP Classification (All Prompt Strategies)

Prompt Source	Strategy	Accuracy	Macro F1	Weighted F1
LLM (GPT-4)	**Text + Few-shot**	**44.4%**	**0.338**	**0.462**
Manual	Hardcoded v2	42.2%	0.311	0.440
LLM (GPT-4)	Text + CLIP-optimized	35.6%	0.270	0.380
LLM (Gemini)	Multimodal + Few-shot	29.4%	0.220	0.320
Manual	Hardcoded v1	23.3%	0.138	0.210
LLM (Gemini)	Multimodal + CLIP-opt	15.0%	0.100	0.150
LLM (GPT-4)	Text + Jargon	12.2%	0.097	0.140
LLM (Gemini)	Multimodal v1	8.3%	0.091	0.100

C.4 Finetuned Models

Model	Epochs	Dice	Vs Zero-Shot
Base	100	0.371	-33%
Focal	50	0.372	-33%
LoRA	30	0.355	-36%

C.5 Per-Class Segmentation Results (Zero-Shot SAM2)

Class	Dice	Std	IoU	Precision	Recall	Support
Tumor	0.560	0.148	0.403	0.592	0.551	45
Stroma	0.537	0.167	0.385	0.571	0.528	45
Lymphocyte	0.549	0.209	0.391	0.589	0.513	37
Necrosis	**0.699**	0.187	**0.559**	**0.723**	**0.678**	23
Blood Vessel	0.504	0.221	0.357	0.541	0.489	31

C.6 Complete Prompt Ablation Results (SAM2 Zero-Shot)

Per-Class Dice Scores by Prompt Type

Prompt Type	Tumor	Stroma	Lymphocyte	Necrosis	Blood Vessel	Overall
Centroid	0.270	0.331	0.307	0.514	0.339	0.335
Multi-Point (5)	0.494	0.380	0.364	0.473	0.370	0.417
Box	0.553	0.538	0.532	0.691	0.497	0.553
Box + Neg	**0.560**	**0.537**	**0.549**	**0.699**	**0.504**	**0.560**

Key Observations:

- Necrosis achieves highest Dice across all prompt types
- Box prompts provide +64% improvement over centroid
- Negative points provide minimal but consistent improvement

C.7 Finetuned Model Per-Class Results

SAM2 Box Focal (50 epochs, Epoch 15 Checkpoint)

Class	Dice	Std	Count
Tumor	0.386	0.237	45
Stroma	0.386	0.231	45
Lymphocyte	0.301	0.251	37
Necrosis	0.478	0.298	23
Blood Vessel	0.335	0.240	30
Overall	**0.372**		180

SAM2 LoRA Light (30 epochs)

Class	Dice	Std	Count
Tumor	0.320	0.217	45
Stroma	0.394	0.229	45
Lymphocyte	0.265	0.229	37
Necrosis	0.498	0.311	23
Blood Vessel	0.350	0.233	30
Overall	**0.355**		180

PathSAM2 CTransPath (40 epochs)

Class	Dice	Std	Count
Tumor	0.024	0.051	45
Stroma	0.008	0.015	45
Lymphocyte	0.022	0.048	37
Necrosis	0.017	0.036	23
Blood Vessel	0.007	0.013	30
Overall	**0.016**		180

⚠️ **Critical Failure**: PathSAM2 with CTransPath completely failed with 0.016 Dice. The dual-encoder fusion approach likely requires more careful architecture design and training.

C.8 Per-Class Classification Results (All CLIP Configurations)

Best Configuration: LLM Text Few-Shot (44.4% Accuracy)

Class	Precision	Recall	F1	Support	Predicted
Tumor	97.4%	82.2%	89.2%	45	38
Stroma	61.9%	28.9%	39.4%	45	21
Lymphocyte	0.0%	0.0%	0.0%	37	0
Necrosis	0.0%	0.0%	0.0%	23	0
Blood Vessel	25.4%	100%	40.5%	30	119

Complete CLIP Classification Comparison

Prompt Strategy	Accuracy	Macro F1	Tumor F1	Stroma F1	Lymph F1	Necro F1	Vessel F1
LLM Text Few-shot	**44.4%**	**0.338**	**0.892**	**0.394**	0.000	0.000	0.405
Hardcoded v2	42.2%	0.311	0.943	0.157	0.050	0.000	0.408
LLM Text CLIP-opt	35.6%	0.270	0.831	0.085	0.000	0.000	0.432
LLM Multimodal Few-shot	29.4%	0.220	0.647	0.000	0.049	0.000	0.403
Hardcoded v1	23.3%	0.138	0.163	0.216	0.000	0.000	**0.448**
LLM Multimodal CLIP-opt	15.0%	0.100	0.000	0.125	0.023	0.354	0.000
LLM Text Jargon	12.2%	0.097	0.043	0.506	0.033	0.000	0.000
LLM Multimodal v1	8.3%	0.091	0.364	0.182	0.000	0.000	0.000

Key Insights:

1. **Tumor** is the easiest class - achieves 80-94% recall in most configs
2. **Lymphocyte & Necrosis** are impossible for CLIP - 0% recall in all configs
3. **Blood Vessel** is over-predicted in most configs (100% recall, low precision)
4. **Stroma** performance varies widely (0-51% F1)
5. **Text-only prompts** consistently outperform multimodal prompts

Confusion Matrix Analysis (LLM Few-Shot):

- CLIP predicts only 3 classes: Tumor, Stroma, Blood Vessel
- Never predicts Lymphocyte or Necrosis
- Blood vessel is the "catch-all" for uncertain predictions

Appendix D: References

1. Kirillov, A., et al. "Segment Anything." ICCV 2023.
2. Ravi, N., et al. "SAM 2: Segment Anything in Images and Videos." arXiv 2024.
3. Radford, A., et al. "Learning Transferable Visual Models From Natural Language Supervision." ICML 2021.
4. Ma, J., et al. "Segment Anything in Medical Images." Nature Communications 2024.
5. Hu, E., et al. "LoRA: Low-Rank Adaptation of Large Language Models." ICLR 2022.
6. Amgad, M., et al. "Structured crowdsourcing enables convolutional segmentation of histology images." Bioinformatics 2019.

Appendix E: Generated Figures

E.1 Qualitative Results (Real Test Images)

Location: `results/figures/qualitative/`

Figure	Description	Use Case
qualitative_method_comparison.png	4 test images × 6 methods (Original, GT, SAM2 Centroid/Box/Box+Neg, MedSAM)	Main presentation slide - shows real predictions
qualitative_per_class.png	SAM2 Box+Neg results for each tissue class (Tumor, Stroma, Lymphocyte, Necrosis, Blood Vessel)	Per-class performance visualization
qualitative_prompt_comparison.png	Same image with different prompts showing impact on segmentation	Prompt ablation visualization
qualitative_success_failure.png	Top row: High Dice (>0.7), Bottom row: Low Dice (<0.4)	Model limitations discussion
qualitative_full_segmentation.png	Full multi-class segmentation with colored overlays	Complete pipeline demonstration

Generation Script

```
# Run on HPRC with GPU
sbatch scripts/slurm/run_qualitative_figures.slurm

# Or locally (requires SAM2 + MedSAM checkpoints)
python scripts/analysis/generate_qualitative_results.py
```

E.2 Academic Charts (Synthetic/Metrics-Based)

Location: `results/figures/academic/`

Figure	Description
fig1_segmentation_comprehensive.png	4-panel: Prompt ablation, model comparison, zero-shot vs finetuned, per-class heatmap
fig2_clip_analysis.png	3-panel: Strategy comparison, per-class accuracy, prompt evolution
fig3_training_analysis.png	3-panel: Training loss, validation Dice, final test comparison
fig4_method_overview.png	Pipeline architecture schematic
fig5_summary_results.png	Complete results tables

Generation Script

```
python scripts/analysis/generate_academic_figures.py
```

E.3 Recommended Presentation Figures

For a 6-minute presentation, use these figures in order:

1. **Slide 3 (Methods)**: `academic/fig4_method_overview.png`
2. **Slide 5-6 (Main Results)**: `qualitative/qualitative_method_comparison.png` ⭐
3. **Slide 7 (Per-Class)**: `qualitative/qualitative_per_class.png`
4. **Slide 8 (CLIP)**: `academic/fig2_clip_analysis.png`
5. **Slide 9 (Finetuning)**: `academic/fig3_training_analysis.png`
6. **Slide 10 (Summary)**: `academic/fig5_summary_results.png`

Document Version: 1.1
Last Updated: December 3, 2025
Authors: VFM Project Team, CSCE 689