PSEUDO-CODE

1. SYSTEM INITIALIZATION AND SETUP

ALGORITHM: Initialize_System

```
BEGIN
  // Check GPU availability
  device ← get_gpu_device_name()
  IF device ≠ 'GPU:0' THEN
    RAISE SystemError('GPU not found')
  END IF

  // Set random seeds for reproducibility
  SET tensorflow_seed = 666
  SET numpy_seed = 666

  // Initialize hyperparameters
  BATCH_SIZE ← 64
  EPOCHS ← 50
  IMAGE_SIZE ← (224, 224)
  NUM_CLASSES ← 25
  LEARNING_RATE ← 2e-5
END
```

2. DATA AUGMENTATION PIPELINE

ALGORITHM: Data_Augmentation

```
FUNCTION flip_random_crop(image):
  image ← apply_random_horizontal_flip(image)
  image ← random_crop(image, size=(224, 224, 3))
  RETURN image

FUNCTION color_jitter(image, strength=[0.4, 0.4, 0.4, 0.1]):
  image ← adjust_brightness(image, delta=0.8*strength[0])
  image ← adjust_contrast(image, lower=1-0.8*strength[1], upper=1+0.8*strength[1])
  image ← adjust_saturation(image, lower=1-0.8*strength[2], upper=1+0.8*strength[2])
```

```
  image ← adjust_hue(image, delta=0.2*strength[3])
  image ← clip_values(image, min=0, max=255)
  RETURN image

FUNCTION color_drop(image):
  image ← convert_to_grayscale(image)
  image ← tile_to_rgb(image)
  RETURN image

FUNCTION random_apply(func, image, probability):
  IF random() < probability THEN
    RETURN func(image)
  ELSE
    RETURN image
  END IF

FUNCTION custom_augment(image, label):
  image ← flip_random_crop(image)
  image ← random_apply(color_jitter, image, p=0.8)
  image ← random_apply(color_drop, image, p=0.2)
  RETURN image, label
```

3. DATASET PREPARATION

ALGORITHM: Load_Dataset

```
BEGIN
  // Define 25 classes for gastrointestinal images
  classes ← [
    cecum, ileum, retroflex_rectum, hemorrhoids, polyps,
    ulcerative_colitis_grade_0-1, ulcerative_colitis_grade_1,
    ulcerative_colitis_grade_1-2, ulcerative_colitis_grade_2,
    ulcerative_colitis_grade_2-3, ulcerative_colitis_grade_3,
    bbps_0-1, bbps_2-3, impacted_stool,
    dyed_lifted_polyps, dyed_resection_margins,
    pylorus, retroflex_stomach, z_line,
    barretts, barretts_short_segment,
    esophagitis_a, esophagitis_b-d,
    colon_adenocarcinoma, colon_benign_tissue
  ]

  // Create dataset dictionary mapping class_id to file_paths
```

```
    dataset ← {}
    FOR i = 0 TO 24 DO
        dataset[i] ← get_file_paths(classes[i])
    END FOR
END

FUNCTION combineDataAndLabel(width, height, dataset):
    data ← empty_list()
    labels ← empty_list()

    FOR EACH class_id IN dataset DO
        FOR EACH image_path IN dataset[class_id] DO
            image ← load_image(image_path, size=(width, height))
            data.append(image)
            labels.append(class_id)
        END FOR
    END FOR

    data ← convert_to_array(data)
    labels ← convert_to_array(labels)
    RETURN data, labels
```

4. MOBILENETV3 MODEL ARCHITECTURE

ALGORITHM: Build_MobileNetV3_Model

```
FUNCTION create_model(num_classes, input_shape):
  // Load pre-trained MobileNetV3Large model
  base_model ← MobileNetV3Large(
    weights='imagenet',
    include_top=False,
    input_shape=input_shape,
    pooling='avg'
  )

  // Freeze base model layers initially
  base_model.trainable ← False

  // Build complete model
  model ← Sequential([
    Input(shape=input_shape),
    base_model,
    BatchNormalization(),
    Dropout(rate=0.2),
    Dense(units=num_classes, activation='softmax')
  ])

  RETURN model
```

5. MODEL TRAINING PROCESS

ALGORITHM: Train_Model

```
BEGIN
  // Phase 1: Transfer Learning (Frozen Base)
  model ← create_model(num_classes, input_shape)

  // Configure optimizer
  optimizer ← Adam(learning_rate=2e-5)

  // Compile model
  model.compile(
    optimizer=optimizer,
    loss='categorical_crossentropy',
    metrics=['accuracy']
```

```
  )

  // Setup callbacks
  callbacks ← [
    ReduceLROnPlateau(factor=0.2, patience=5),
    EarlyStopping(patience=10, restore_best_weights=True),
    ModelCheckpoint('best_model.h5', save_best_only=True)
  ]

  // Train initial model
  FOR epoch = 1 TO 20 DO
    train_batch ← get_next_batch(train_data, batch_size)
    augmented_batch ← custom_augment(train_batch)
    loss ← model.train_on_batch(augmented_batch)
    validate_model(model, validation_data)
  END FOR

  // Phase 2: Fine-tuning (Unfreeze Top Layers)
  unfreeze_layers ← last_100_layers(base_model)
  FOR layer IN unfreeze_layers DO
    layer.trainable ← True
  END FOR

  // Continue training with fine-tuning
  optimizer ← Adam(learning_rate=1e-5)  // Lower learning rate
  model.compile(optimizer, loss, metrics)

  FOR epoch = 21 TO 50 DO
    train_model_epoch(model, train_data, validation_data)
  END FOR
END
```

6. MODEL EVALUATION AND METRICS

ALGORITHM: Evaluate_Model

```
FUNCTION evaluate_performance(model, test_data, test_labels):
  // Make predictions
  predictions ← model.predict(test_data)
  predicted_classes ← argmax(predictions, axis=1)

  // Calculate metrics
  accuracy ← calculate_accuracy(predicted_classes, test_labels)
  precision ← calculate_precision(predicted_classes, test_labels, average='weighted')
  recall ← calculate_recall(predicted_classes, test_labels, average='weighted')
  f1_score ← calculate_f1_score(predicted_classes, test_labels, average='weighted')

  // Generate confusion matrix
  confusion_matrix ← compute_confusion_matrix(test_labels, predicted_classes)

  // Generate per-class classification report
  classification_report ← generate_classification_report(
    test_labels, predicted_classes, class_names
  )

  RETURN {
    'accuracy': accuracy,
    'precision': precision,
    'recall': recall,
    'f1_score': f1_score,
    'confusion_matrix': confusion_matrix,
    'classification_report': classification_report
  }
```

7. MODEL EXPLAINABILITY (GRAD-CAM)

ALGORITHM: Generate_Explainability_Visualizations

```
FUNCTION generate_gradcam_heatmap(model, image, class_id):
  // Prepare model for Grad-CAM
  grad_model ← create_gradient_model(model, last_conv_layer)

  // Forward pass and compute gradients
  WITH gradient_tape AS tape:
    conv_outputs, predictions ← grad_model(image)
```

```
    class_channel ← predictions[:, class_id]

  // Compute gradients
  grads ← tape.gradient(class_channel, conv_outputs)

  // Pool gradients and compute weighted activation
  pooled_grads ← global_average_pooling(grads)
  heatmap ← sum(conv_outputs * pooled_grads, axis=-1)
  heatmap ← relu(heatmap)  // Apply ReLU
  heatmap ← normalize(heatmap, min=0, max=1)

  RETURN heatmap

FUNCTION visualize_all_cams(model, test_image, true_label):
  // Generate multiple CAM visualizations
  gradcam ← generate_gradcam(model, test_image, true_label)
  gradcam_plus ← generate_gradcam_plusplus(model, test_image, true_label)
  scorecam ← generate_scorecam(model, test_image, true_label)
  layercam ← generate_layercam(model, test_image, true_label)

  // Create visualization grid
  display_grid([
    original_image,
    overlay(original_image, gradcam),
    overlay(original_image, gradcam_plus),
    overlay(original_image, scorecam),
    overlay(original_image, layercam)
  ])
```

8. MAIN EXECUTION PIPELINE

ALGORITHM: Main_Pipeline

```
BEGIN
  // Step 1: Initialize system
  Initialize_System()

  // Step 2: Load and prepare dataset
  dataset ← Load_Dataset()
  data, labels ← combineDataAndLabel(224, 224, dataset)

  // Step 3: Split data
  x_train, x_temp, y_train, y_temp ← train_test_split(data, labels, test_size=0.2)
  x_val, x_test, y_val, y_test ← train_test_split(x_temp, y_temp, test_size=0.5)

  // Step 4: One-hot encode labels
  y_train ← to_categorical(y_train, num_classes=25)
  y_val ← to_categorical(y_val, num_classes=25)
  y_test_categorical ← to_categorical(y_test, num_classes=25)

  // Step 5: Create data pipelines
  train_dataset ← create_tf_dataset(x_train, y_train)
  train_dataset ← train_dataset.map(custom_augment).batch(64).prefetch()
  val_dataset ← create_tf_dataset(x_val, y_val).batch(64)

  // Step 6: Build and train model
  model ← create_model(25, (224, 224, 3))
  Train_Model(model, train_dataset, val_dataset)

  // Step 7: Evaluate model
  metrics ← evaluate_performance(model, x_test, y_test)
  PRINT "Accuracy:", metrics.accuracy
  PRINT "Precision:", metrics.precision
  PRINT "Recall:", metrics.recall
  PRINT "F1 Score:", metrics.f1_score

  // Step 8: Generate visualizations
  plot_confusion_matrix(metrics.confusion_matrix, class_names)
  visualize_predictions(model, x_test, y_test, sample_count=16)

  // Step 9: Generate explainability visualizations
  FOR sample IN test_samples DO
    visualize_all_cams(model, sample.image, sample.label)
```

```
    END FOR

    // Step 10: Save final model
    save_model(model, 'mobilenetv3_gastrointestinal_classifier.h5')
END
```

APPENDIX: KEY DATA STRUCTURES

STRUCTURE: Class_Mapping

Class_ID → Class_Name:

0 → cecum
1 → ileum
2 → retroflex_rectum
3 → hemorrhoids
4 → polyps
5 → ulcerative_colitis_grade_0-1
6 → ulcerative_colitis_grade_1
7 → ulcerative_colitis_grade_1-2
8 → ulcerative_colitis_grade_2
9 → ulcerative_colitis_grade_2-3
10 → ulcerative_colitis_grade_3
11 → bbps_0-1
12 → bbps_2-3
13 → impacted_stool
14 → dyed_lifted_polyps
15 → dyed_resection_margins
16 → pylorus
17 → retroflex_stomach
18 → z_line
19 → barretts
20 → barretts_short_segment
21 → esophagitis_a
22 → esophagitis_b-d
23 → colon_adenocarcinoma
24 → colon_benign_tissue

STRUCTURE: Model_Architecture

Input_Layer: (224, 224, 3)
Base_Model: MobileNetV3Large (pre-trained on ImageNet)
Global_Average_Pooling: Reduce spatial dimensions
Batch_Normalization: Normalize activations
Dropout: Rate=0.2 for regularization
Dense_Output: 25 units with softmax activation

STRUCTURE: Training_Configuration

Batch_Size: 64
Initial_Epochs: 20 (frozen base)
Fine_Tuning_Epochs: 30 (unfrozen top layers)
Initial_Learning_Rate: 2e-5

Fine_Tuning_Learning_Rate: 1e-5
Optimizer: Adam
Loss_Function: Categorical_Crossentropy
Metrics: ['accuracy']
Callbacks: [ReduceLROnPlateau, EarlyStopping, ModelCheckpoint]