Malaria Object Detection System - Project Documentation

1. Project Overview

This project implements a professional-grade Object Detection System for detecting malaria parasites in microscopy images. Unlike traditional binary classification (Infected vs Uninfected), this system identifies the exact location of parasites within a cell using the YOLOv11 architecture.

Key Engineering Highlights:

- Synthetic Data Pipeline: Generates object detection datasets from single-cell images.

- State-of-the-Art Model: Uses YOLOv11 Nano for real-time inference.

- MLOps: Deployed via Dockerized FastAPI backend and Streamlit frontend.

2. System Architecture

The system follows a microservice architecture:

1. Data Layer: Fetches NIH data, synthesizes 'slides' with bounding box labels.

2. Training Layer: Fine-tunes YOLOv11n on the synthetic dataset.

3. Inference Layer (FastAPI): Exposes a REST API endpoint (/predict) that accepts images and returns JSON detections.

4. Presentation Layer (Streamlit): A user-friendly web UI for uploading images and visualizing results.

3. Code Structure & Explanation

A. Data Pipeline (data_pipeline.py)

The 'CellCompositor' class is the core engine. It downloads the NIH dataset and randomly places single cells onto a blank canvas to create synthetic microscopy slides. Crucially, it calculates the bounding box coordinates (x, y, w, h) for every placed cell, allowing us to train an Object Detection model without manual

annotation.

B. Model Training (train_yolo.py)

We utilize the 'ultralytics' library to train YOLOv11. The script points to the generated 'dataset.yaml' and runs for a specified number of epochs. We use the 'yolo11n.pt' (Nano) weights for speed and efficiency.

C. Backend API (main.py)

Implemented using FastAPI. It has a single endpoint '/predict'. When an image is received, it is converted to a generic PIL format, passed through the loaded YOLO model, and the resulting bounding boxes are returned as a JSON response.

D. Frontend UI (app.py)

Built with Streamlit. It handles file uploads, sends the image to the FastAPI backend, and uses OpenCV to draw the returned bounding boxes onto the image for visualization. Red boxes indicate Parasitized cells, Green boxes indicate Uninfected.

4. How to Run

1. Generate Data: python data_pipeline.py

2. Train Model: python train_yolo.py

3. Start Backend: uvicorn main:app --port 8000

4. Start Frontend: streamlit run app.py