MLOps Object(Sparrow) Detection Project — User Manual

This project is an end-to-end MLOps system for training, serving, and monitoring an object detection model using: - **DVC** for data version control - **Docker** for orchestration - **Prometheus** for monitoring - **Flask** for serving the model via API - **Python scripts** for model training and utility functions

Project Structure

```
mlops-object-detection/
├── data/                    # Datasets managed via DVC
├── models/                  # Trained models tracked with DVC
├── train.py                 # Model training script
├── api.py                   # API server exposing inference endpoints
├── docker-compose.yml       # Orchestrates services
├── dvc.yaml                 # DVC pipeline config
├── params.yaml              # Training hyperparameters
├── prometheus.yml           # Prometheus config file
├── requirements.txt         # Training dependencies
├── api_requirements.txt     # API dependencies
├── app_requirements.txt     # UI dependencies (if present)
├── utils.py                 # Shared utility functions
└── README.md                # Project summary
```

Setup Instructions

1 Install Dependencies

Training dependencies:

```
pip install -r requirements.txt
```

API dependencies:

```
pip install -r api_requirements.txt
```

UI dependencies (if available):

```
pip install -r app_requirements.txt
```

Data Version Control (DVC)

Pull existing data:

```
dvc pull
```

Add new data:

```
dvc add data/your_dataset
git add data/.gitignore data/your_dataset.dvc
git commit -m "Added new dataset"
dvc push
```

Model Training

Modify hyperparameters: Edit `params.yaml`

Train the model:

```
python train.py
```

Training logs:
Check `train.log` for progress and issues.

Serving the Model (API)

Run API locally:

```
python api.py
```

Or via Docker Compose:

```
docker-compose up --build
```

API runs on: `http://localhost:5000/`

Monitoring (Prometheus)

Prometheus config is in `prometheus.yml`.

Prometheus Dashboard:
http://localhost:9090

View metrics and system health.

API Endpoints

Endpoint	Method	Description
`/predict`	POST	Upload an image and get detections
`/metrics`	GET	Expose Prometheus metrics

Adding a New Model

1. Place dataset under `data/`
2. Track with DVC: `bash dvc add data/your_dataset git add data/.gitignore data/your_dataset.dvc git commit -m "new dataset" dvc push`
3. Update `params.yaml`
4. Train with: `bash python train.py`
5. Replace old model in `models/` if needed
6. Update `api.py` to load the new model
7. Restart API: `bash docker-compose restart api`

Docker Compose

Run everything together:

```
docker-compose up --build
```

Stop services:

```
docker-compose down
```

Notes

- Ensure DVC remote is configured for data sharing
- Prometheus targets API metrics for system health
- Check `logs/train.log` for model training details
- `utils.py` has helper utilities used in both training and inference