# **Covid-19 prediction ML Ops**

Transferring a Data Science project into ML Ops

# Project Report & Technical Documentation

# Team:

* Vishal Desai ([vishalkirandesai@gmail.com](mailto:vishalkirandesai@gmail.com))
* Markus Fuchs ([markus\_fuchs@gmx.net](mailto:markus_fuchs@gmx.net))
* Philipp Würfel ([phi.wuerfel@gmail.com](mailto:phi.wuerfel@gmail.com))
* Benjamin Ries ([benjaminjacobries@gmail.com](mailto:benjaminjacobries@gmail.com))

Mentor: Kilyan Poilly

Submission Date: 18.06.2025

## 1. Introduction

### Project Overview

This project aims to provide an end-to-end MLOps pipeline for Covid-19 detection using machine learning. It leverages containerized microservices, robust data versioning, observability, and modern CI/CD practices.

The data science part of the project had already been done. It consists of the following parts which have already been developed in the previous course:

* Image files, classified by whether the person is infected with Covid 19 or not,
* preprocessing scripts,
* machine learning models and the respective training scripts,
* a streamlit presentation.

### Objectives

The project has two goals:

1. To set up an MLOPS environment for the data science project and thus to implement the MLOPS lifecycle,
2. and to use open source solutions instead of proprietary solutions whenever possible to avoid vendor lock-in and dependency on hyperscalers.

## 2. Importance of MLOPS

MLOPS is necessary to turn the data science project into a viable solution for several reasons:

* Training and maintaining big image classification models on a personal computer is a fool’s errand. The hardware needs are considerable, especially when the training process is expected to be anywhere near what we consider fast.
* Image classification data, i.e. a database of classified images big enough to train decent models, is huge. It makes sense to store it wherever it is safe and storage is cheap instead of a single system.
* Covid 19 is evolving rapidly. Hence, classification models need to be updated and even split up for different variants regularly.
* For the same reason, data needs to be updated. Hence, data version control is necessary.

## 

## 

## 

## 3. Project Requirements

### Data Description

The data and its preprocessing was exactly the same as in the Data Science project. It is the [Kaggle COVID-19 Radiography Database](https://www.kaggle.com/datasets/tawsifurrahman/covid19-radiography-database) data and consists of image files from X-ray images of lungs which are in one of four states: COVID infection, viral pneumonia, lung opacity and normal. The database has a size of over 800 MB.

### Tools and Technologies

* + Python 3.11.9
  + Git & Github for version control
  + Pre-commit - saving time by ensuring certain standards are met from the beginning and enforced via Pre-commit hooks
  + MLFlow for experiment tracking and model registry
  + DVC for S3 as a standalone solution, used without Dagshub which bears a risk of vendor lock-in
  + Linux & Bash
  + FastAPI with Uvicorn for serving applications
  + Streamlit for interactive dashboards and visualization
  + Prometheus, Loki and Grafana (Alloy) for observability/logging and monitoring
  + Docker for containerization and orchestration
  + Docker Compose for multi-container orchestration
  + PyJWT for security

### Hardware and Software Needs

Python libraries:

* + Numpy, Pandas, Tensorflow for the Data Science part
  + gdown - to load models from google drive, getting away from Dagshub
  + TensorFlow: Deep learning framework
  + pytest: Testing framework
  + black, flake8, isort, pylint: Code formatting and linting
  + dvc, dvc[s3]: Data version control and S3 integration
  + pre-commit: Git hooks for code quality
  + streamlit: Web app framework for ML visualization
  + python-json-logger: Structured logging
  + prometheus-fastapi-instrumentator: FastAPI metrics for Prometheus

See requirements\_dev.txt and requirements.txt for full lists.

Hardware recommendations:

* CPU: Quad-core or higher recommended
* RAM: Minimum 8GB (16GB+ recommended for training)
* Disk: SSD with at least 20GB free space (more for large datasets/models)
* GPU: Optional, but recommended for faster model training (NVIDIA CUDA-compatible)

## 

## 4. Architecture Overview

## 

### Data & Model Flow

1. Data Ingestion: Data is versioned and managed using DVC.
2. Model Training: *ml\_train\_hub* registers trained models and logs experiments to MLflow.
3. Model Hosting: *ml\_host\_backend* has client facing APIs serving login, model information and prediction.
4. User Management: *ml\_user\_mgmt* handles authentication and user data.
5. Observability: Prometheus scrapes metrics, Loki collects logs, Grafana visualizes both.
6. Visualization: Streamlit provides a user-facing dashboard for predictions and monitoring.

### Folder Structure

* services/: All containerized microservices
  + ml\_host\_backend/: Model hosting backend
  + ml\_train\_hub/: Model training and registration hub
  + ml\_user\_mgmt/: User management service
* prototyping/: Experimental scripts and notebooks
* streamlit/: Streamlit dashboard and API client
* observability/: Monitoring stack (Prometheus, Grafana, Loki)
* data/: Data storage (managed by DVC)
* .envs/: Environment variable files

### Setup Instructions

1. Clone the repository
2. Install Python requirements:  
   pip install -r requirements\_dev.txt
3. Install DVC:  
   pip install dvc dvc[s3]
4. Install Docker & Docker Compose
5. Set up environment variables in .envs
6. Start services:  
   docker-compose up -d
7. Run Streamlit app:  
   cd streamlit && streamlit run covid19mlops\_app.py

## 5. Methodology

We agreed on an agile project methodology and thus against a waterfall model. Therefore, not every single tool and technology was decided and listed from the beginning. This was especially necessary since at the beginning of the project we had not explored every single technology but were already required to create deliverables, as is agile practice. We agreed on what to do during the first sprints and to decide everything else while sailing.

To deal with the funnel of insecurity, we used a user story map which was not fully specified at sprint 1 either but provided us with a sufficient backlog for the first two sprints. We updated the user story map as we progressed. After sprint 3 the story map was stable.

The first increment was to be able to serve a model and be able to register a model with MLflow. This entailed a locally running implementation of Docker, ML models, MLflow and a FastAPI hub to connect everything. We decided to accept temporary solutions to enable the incremental approach associated with agility. Thus, we could immediately get to work without spending too much time on planning too far ahead. After sprint 2 and due to the limited overall project scope, we decided to no longer set specific sprint goals as the remaining tasks had been identified and the goal was just to get done as much as possible.

## 

## 6. References

* Citations:
  + <https://mlflow.org/docs/latest/index.html>
  + <https://dvc.org/doc>
  + <https://docs.streamlit.io/>
  + <https://prometheus.io/>
  + <https://grafana.com/>
  + <https://grafana.com/oss/loki/>