# Report

## TITLE : Surface Crack Detection AI Application


## NAME : Mohit Singh
## Roll No : DA24M010


---

## 1. Abstract


The Surface Crack Detection App allows you to upload photos of surfaces (e.g., concrete walls, pavements) and automatically detects whether cracks are present. It is designed for ease of use and does not require any technical expertise.

**This tool is ideal for:**

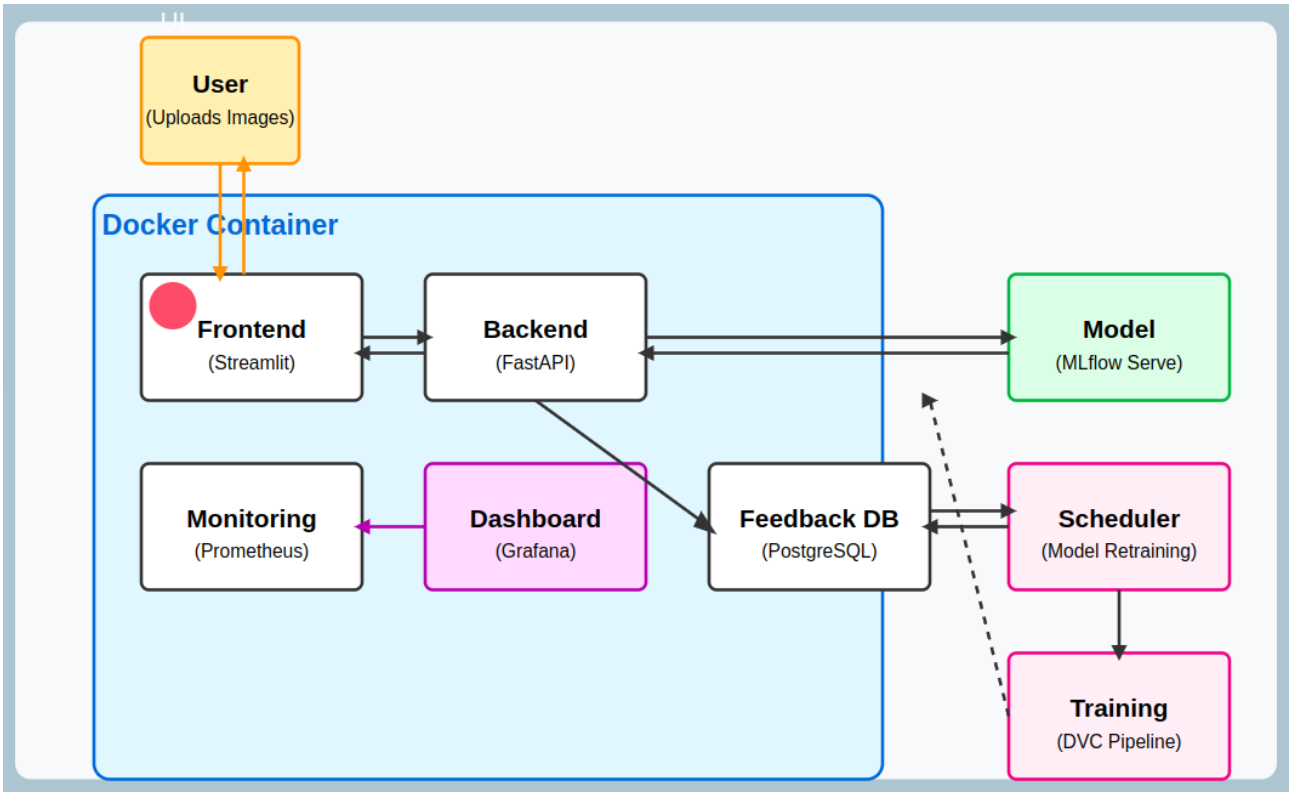
- Civil engineers and construction professionals
- Maintenance teams inspecting infrastructure
- Property owners or site inspectors with minimal technical experience

With just a few clicks, users can upload images and receive instant feedback on the presence of cracks, helping in early detection and maintenance planning.


## 2. Overview

## Tools used:

- MLflow (model tracking & serving)
- DVC (data & model versioning)
- Docker (containerization)
- Grafana + Prometheus (monitoring)
- PostgreSQL (feedback storage)




# 3. Getting Started

# Features

The Surface Crack Detection App provides the following features to help you easily upload, analyze, and view surface images for cracks:

## Image Upload

- Upload image of surfaces (e.g., walls, roads) from your computer.
- Supported formats: .jpg, .jpeg, .png.

## Automatic Crack Detection

- The app uses an AI model to automatically analyze uploaded images.
- Each image is processed to detect cracks and classify whether they are present or not.

### Visual Feedback

- After detection, results are shown with the original image and predicted label.
- You can view whether the image is:
    - **Crack Detected**
    - **No Crack Detected**

### Flag Wrong Predictions

- If the prediction is incorrect, you can **flag it** with a single click.

- Flagged images are saved and can be used later to **retrain the model**, improving accuracy over time.

### Clear and Re-upload

- A **Clear** button is available to reset the image upload section.
- This allows you to **remove previous images** and **upload new ones** seamlessly

# 4. Running the app

After installing docker-desktop you just need to run simple commands to get the project repository and run the full application in your system.

- **Open a terminal window** (Command Prompt, PowerShell, or Terminal depending on your OS).
- **Clone and project and Navigate to the project folder**

  Use the cd command to go to the folder where you have stored the application files
  *git clone https://github.com/DA24M010/da5402_aiapp.git*

  *cd da5402_aiapp*
- **Start the application using Docker Compose**
  Run the following command to build and start all services (frontend, backend, database, etc.)*:*

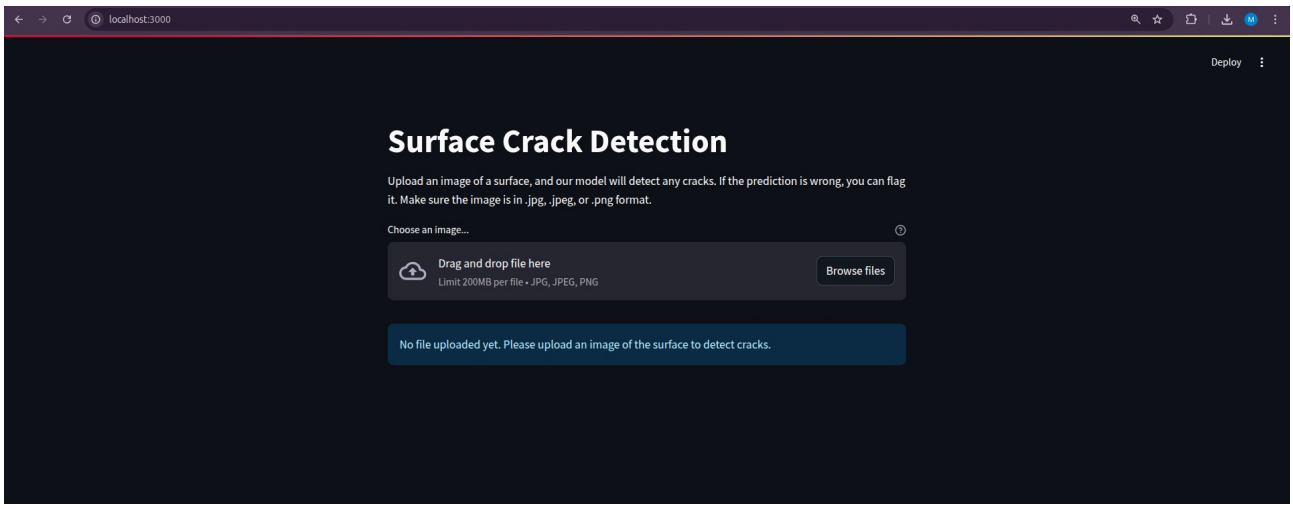
  *docker-compose up –build*
- **Access the web application**
  Once all services are up, open your browser and go to:

  `http://localhost:3000`
- **Stopping the application**
  To stop the application, press `Ctrl + C` in the terminal where Docker is running, and then run:

  *docker-compose down*

# 5. Application Interface

On lauching *http://localhost:3000* on your browser window the below application window will appear.

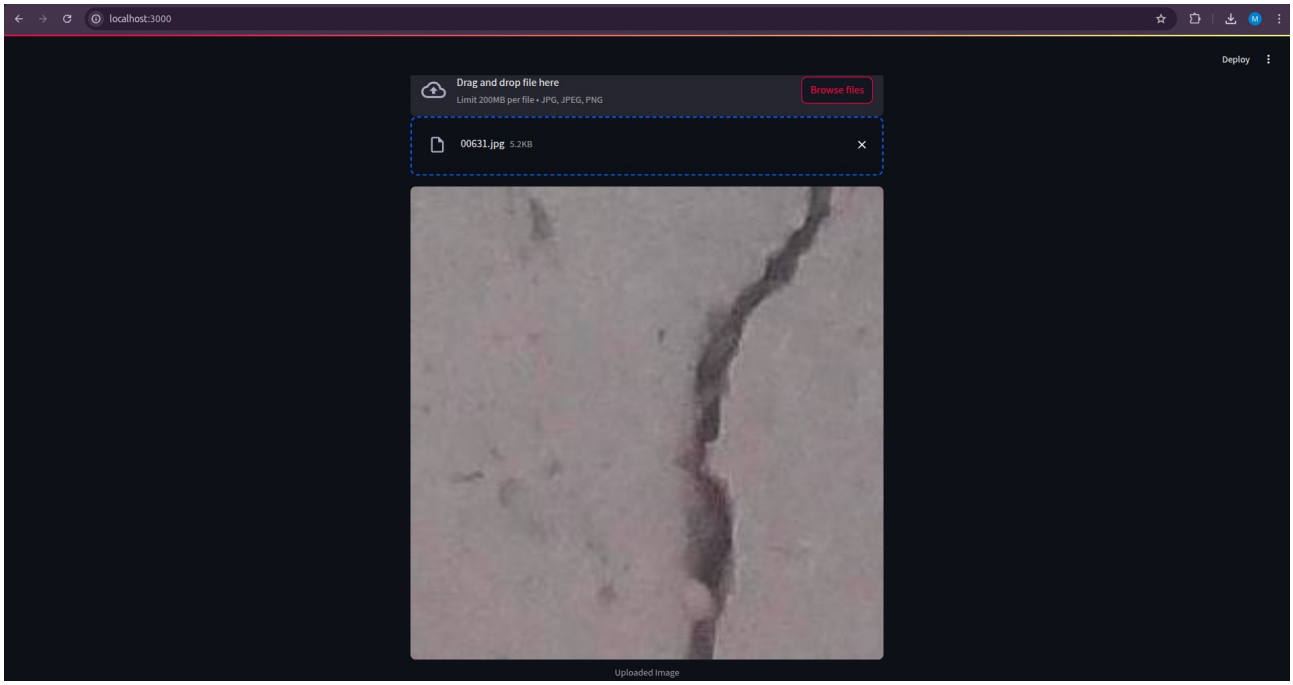



## Upload an Image

- Click on the "**Browse files**" button.
- Select an image of a concrete surface from your computer (e.g., `.jpg`, `.png`).
- The uploaded image will be displayed on the screen.

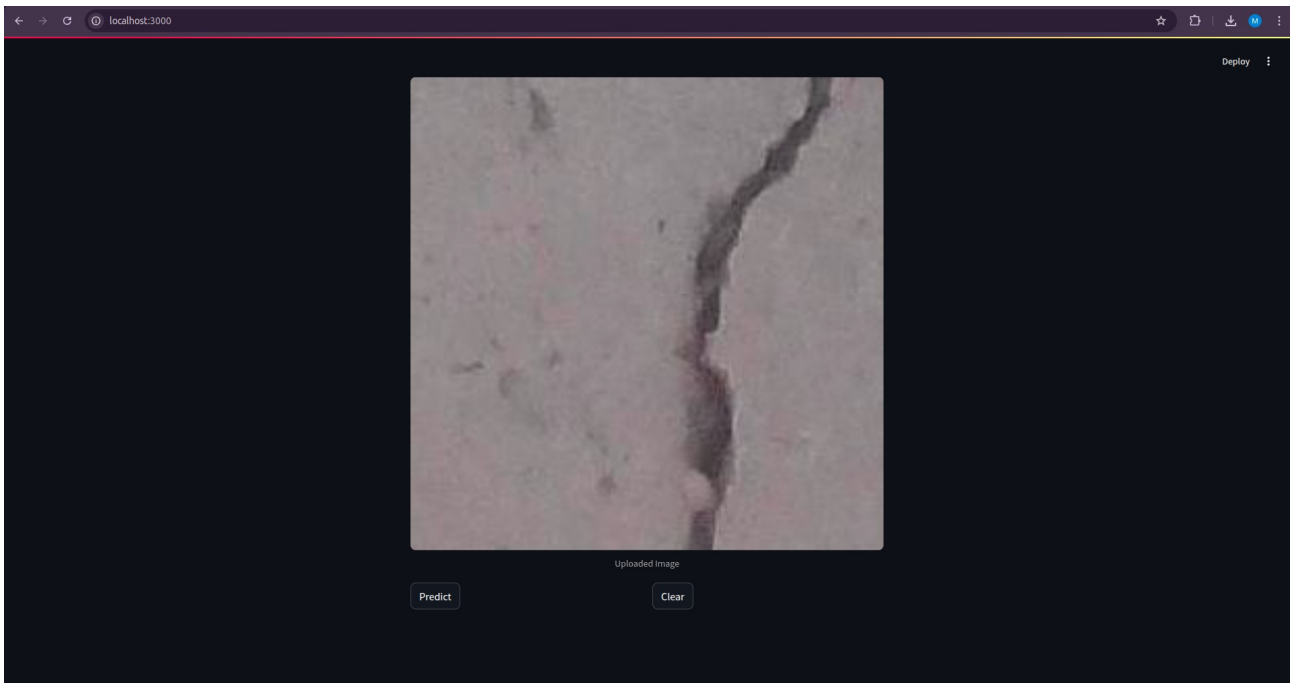
OR

- The image can also be dragged from the system on top of the specified place to load the image.



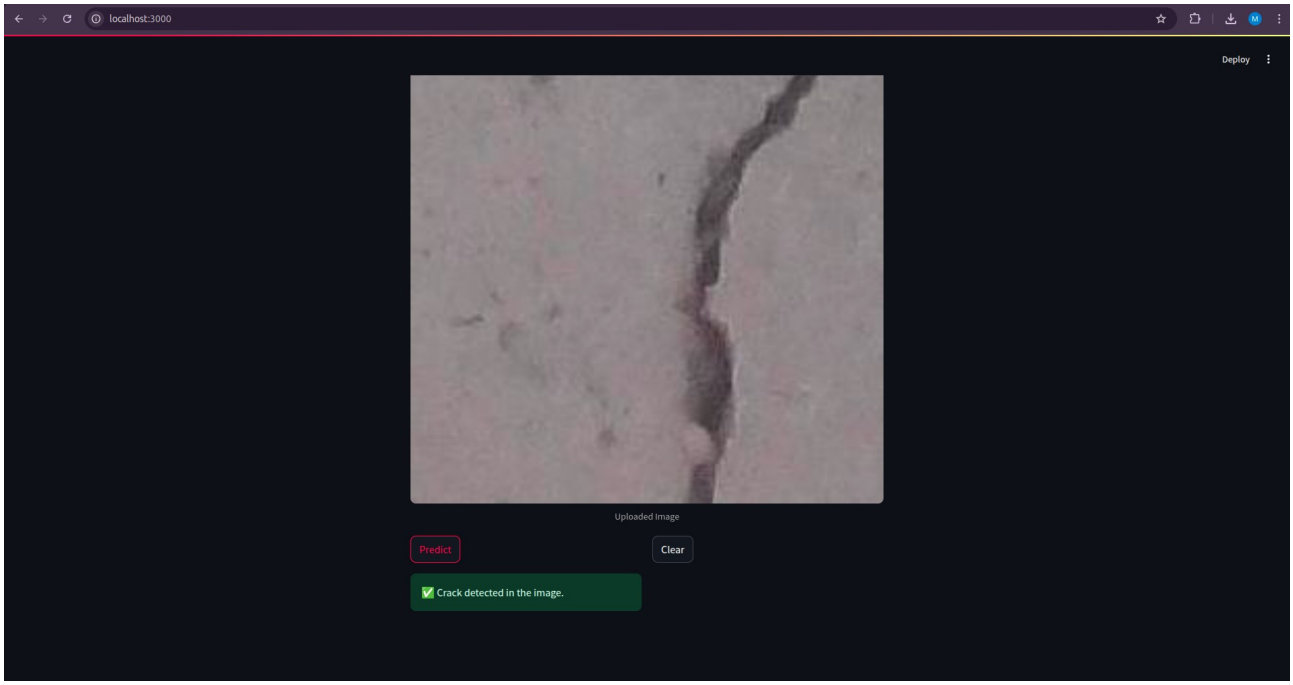

The above screen shows the loaded image on the screen.

When you scroll down slightly, the UI shifts to the next section, where two new buttons become visible: "Predict" and "Clear."




## Predict Button

- The app will automatically process the image and display whether the surface contains a **crack** or **no crack**.

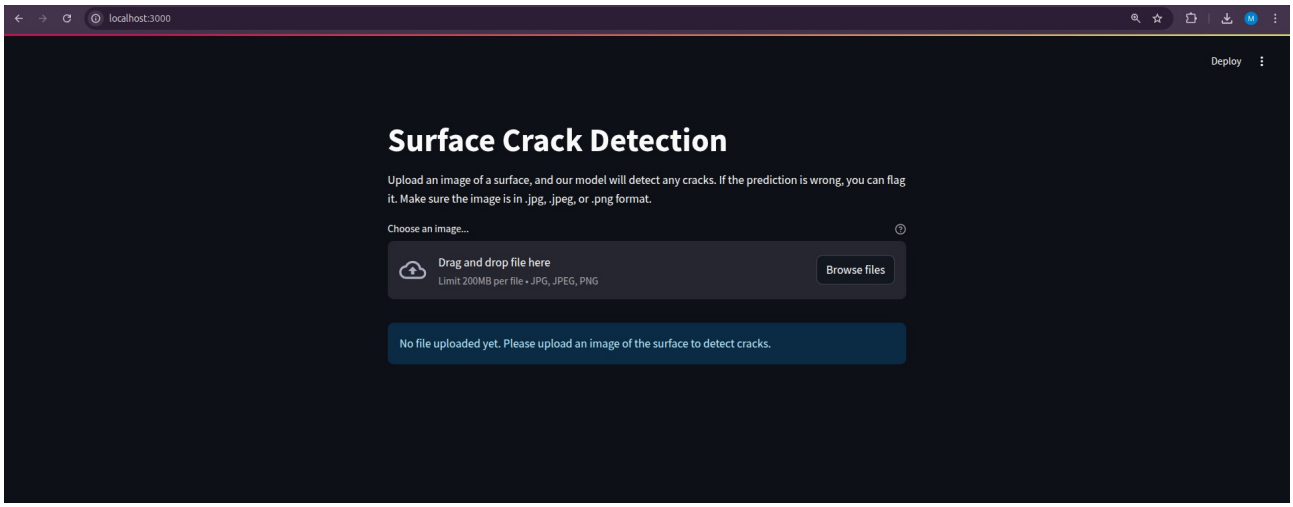



The image above displays the prediction for the input image, which indicates a crack has been detected, resulting in a positive outcome.

## Clear Button

- Click the **"Clear"** button to remove the uploaded image and prediction result.

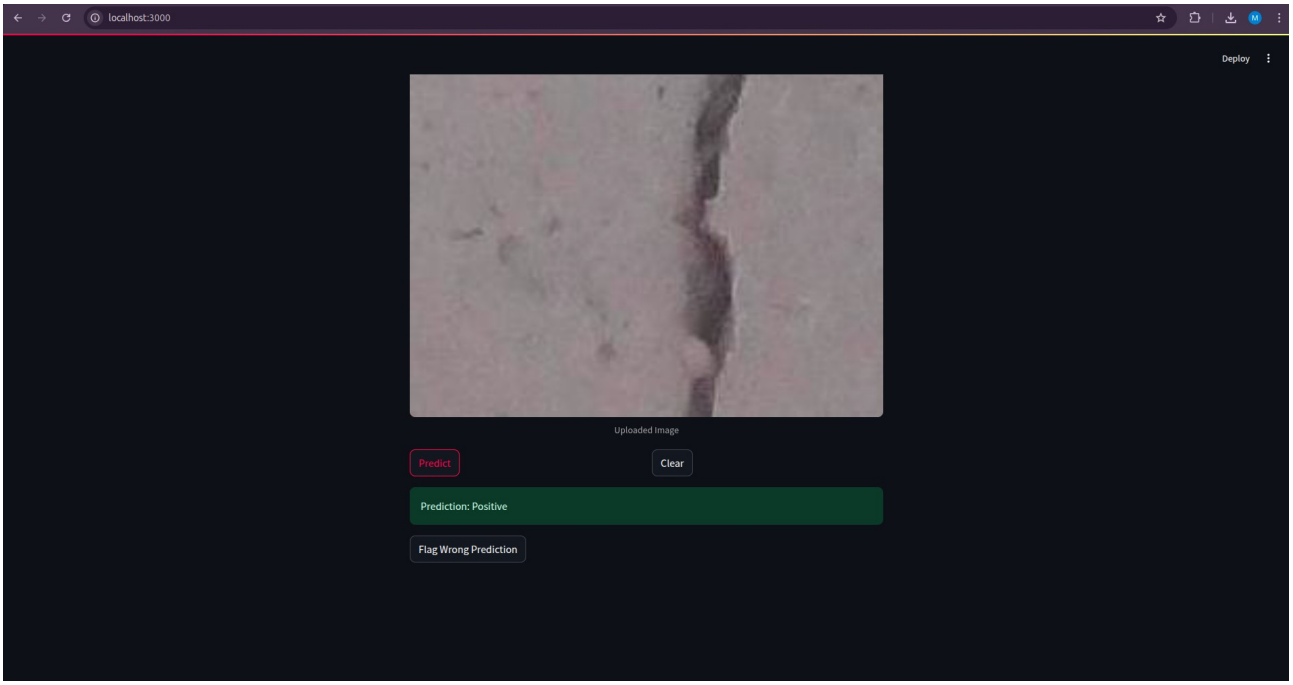
- You can now upload a new image.




The UI resets back to the original UI, removing any image that you inserted or any prediction

## Flag Wrong Prediction

- If you believe the prediction is incorrect, click the **"Flag this prediction"** button.
- This image will be saved for future model retraining, helping improve accuracy over time



## 6. Dataset

The dataset used is a surface crack detection open source data available at kagglehub.
Dataset

The datasets contains images of various concrete surfaces with and without crack. The image data are divided into two as negative (without crack) and positive (with crack) in separate folder for image classification. Each class has 20000images with a total of 40000 images with 227 x 227 pixels with RGB channels.

The dataset is generated from 458 high-resolution images (4032x3024 pixel) with the method proposed by Zhang et al (2016). High resolution images found out to have high variance in terms of surface finish and illumination condition. No data augmentation in terms of random rotation or flipping or tilting is applied.

The dataset was sampled for 5000 images each from positive and negative samples for training the base model and the other part was used as test samples and feedback dataset.

## 7. Machine Learning Model

The model used in this project is a Convolutional Neural Network (CNN) designed for binary classification—determining whether a given surface image contains a crack or no crack. The architecture is implemented using PyTorch and is simple yet effective for image-based defect detection tasks.

Input size: 3-channel RGB images of size 120x120

Output: Single sigmoid-activated value representing the probability of a crack

## 8. Workflow

### Version Control

- Git for code.

- DVC for dataset and model weights.

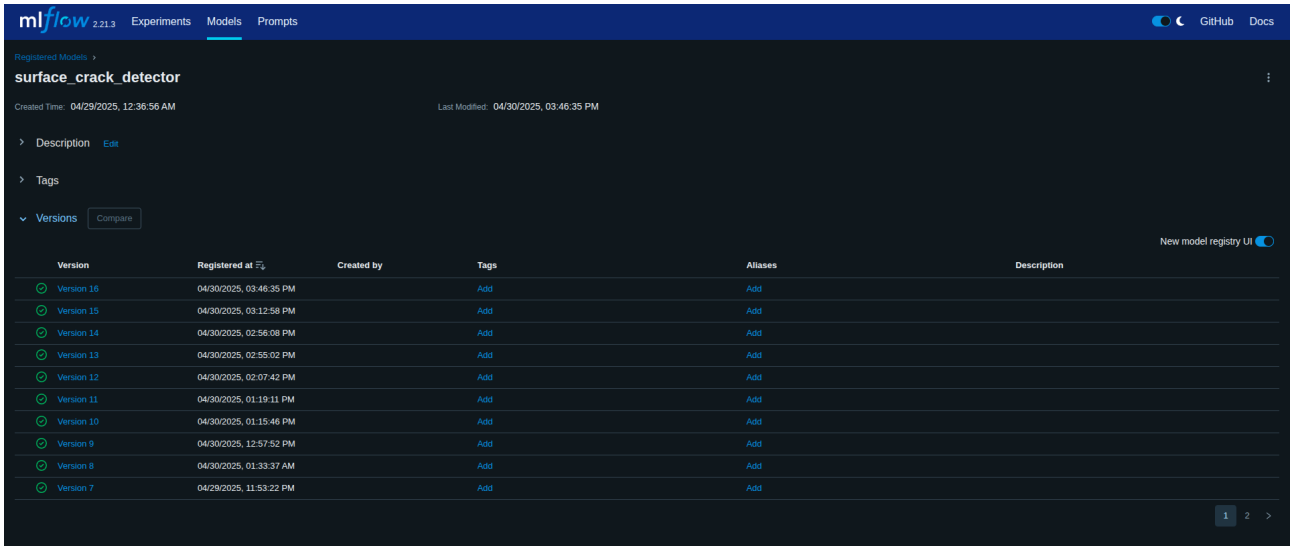
## Model Training Pipeline

DVC DAG


Ingestion → Preprocessing → Splitting → Training → Evaluation

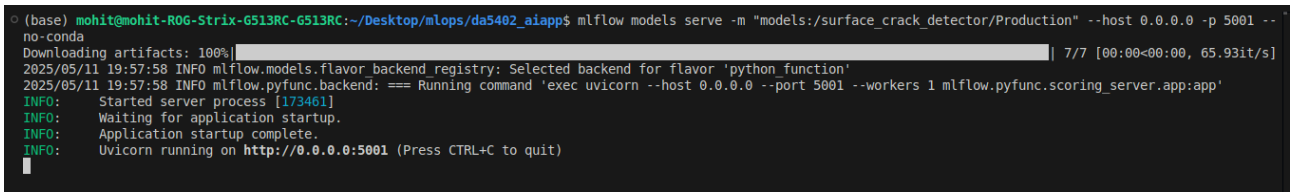
*Scripts used from ./scripts/ folder.*

## Model Deployment



- Serving with mlflow models serve.

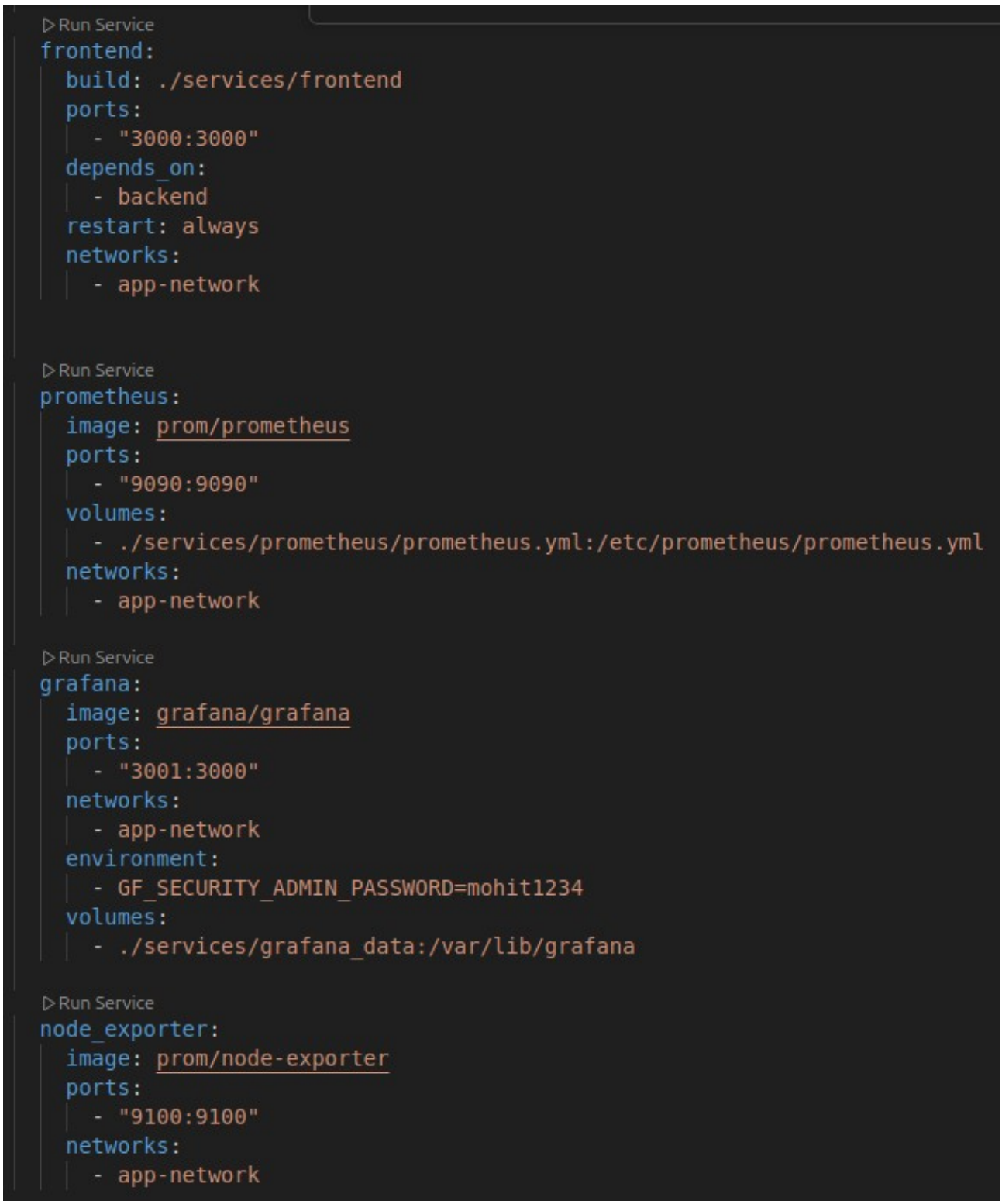
  Endpoint: http://host.docker.internal:5001/invocations.




## Application Deployment

docker-compose file

```
Run All Services
services:
  Run Service
  postgres:
    image: postgres:latest
    environment:
      POSTGRES_USER: mohit
      POSTGRES_PASSWORD: mohit
      POSTGRES_DB: feedback
    volumes:
      - ./services/postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    networks:
      - app-network


  Run Service
  backend:
    build: ./services/backend
    ports:
      - "8000:8000"
    restart: always
    environment:
      - POSTGRES_HOST=postgres
      - POSTGRES_PORT=5432
      - POSTGRES_DB=feedback
      - POSTGRES_USER=mohit
      - POSTGRES_PASSWORD=mohit
      - MLFLOW_MODEL_SERVER_URL=http://host.docker.internal:5001/invocations
      - DATABASE_URL=postgresql://mohit:mohit@postgres:5432/feedback

    networks:
      - app-network
```

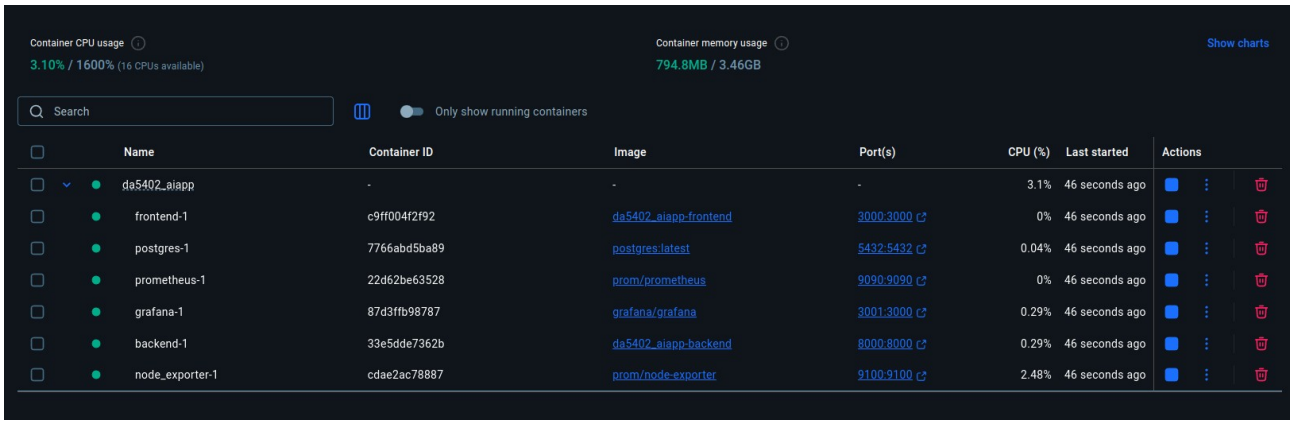


```
▷ Run Service
frontend:
  build: ./services/frontend
  ports:
    - "3000:3000"
  depends_on:
    - backend
  restart: always
  networks:
    - app-network


▷ Run Service
prometheus:
  image: prom/prometheus
  ports:
    - "9090:9090"
  volumes:
    - ./services/prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
  networks:
    - app-network

▷ Run Service
grafana:
  image: grafana/grafana
  ports:
    - "3001:3000"
  networks:
    - app-network
  environment:
    - GF_SECURITY_ADMIN_PASSWORD=mohit1234
  volumes:
    - ./services/grafana_data:/var/lib/grafana

▷ Run Service
node_exporter:
  image: prom/node-exporter
  ports:
    - "9100:9100"
  networks:
    - app-network
```

docker-compose up --build launches:

- Backend (FastAPI)
- Frontend (React)
- Postgres(DB)

- Prometheus
- Grafana




## Monitoring

- Prometheus metrics exposed from backend.
- Grafana dashboard for live tracking.

## Feedback Loop/ Model Retraining

- Flagging wrong predictions in UI.
- Flagged predictions Stored in Postgres.
- Used for retraining via feedback handler.
- Cronjob setup for checking the PG db in intervals and if a threshold of new data is met, model is retrained using feedback data and served using mlflow.