CS5830 Big Data Lab Project

Group-2

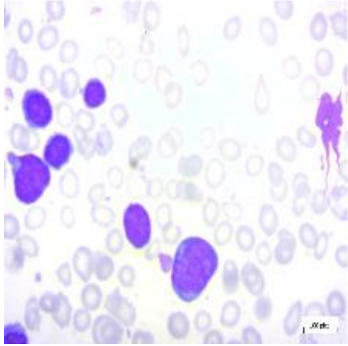
Vishal V ME20B204
Akranth Reddy ME20B100
Sai Gowtham ED19B063

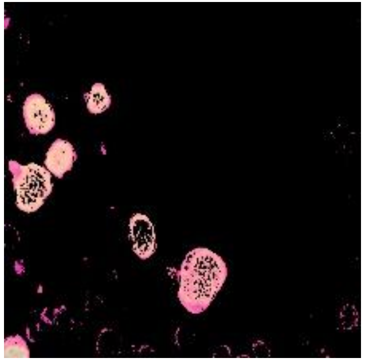
Problem Statement & Workflow

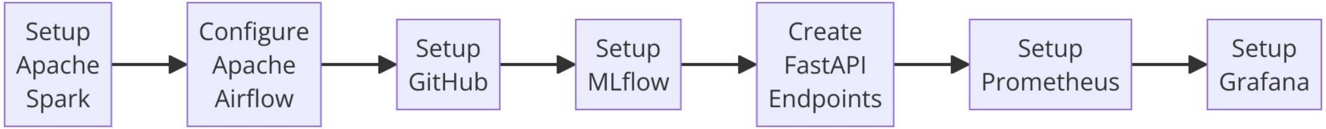
Acute Lymphoblastic Leukemia (ALL) is an aggressive form of cancer that predominantly affects children. Early detection and accurate classification of ALL are crucial for effective treatment and improving patient outcomes. However, diagnosing ALL through microscopic blood smear images is challenging due to visual similarities with other conditions and the need for specialized expertise.

This project aims to leverage an MLOps approach to build an end-to-end machine learning solution for the detection and classification of ALL from microscopic blood smear images. The solution will include a

- data preprocessing pipeline using Apache Airflow,
- machine learning model tracking via MLflow, and
- a scalable REST API for model deployment using FastAPI.
- the entire solution will be containerized for seamless deployment and monitored using Prometheus and Grafana.





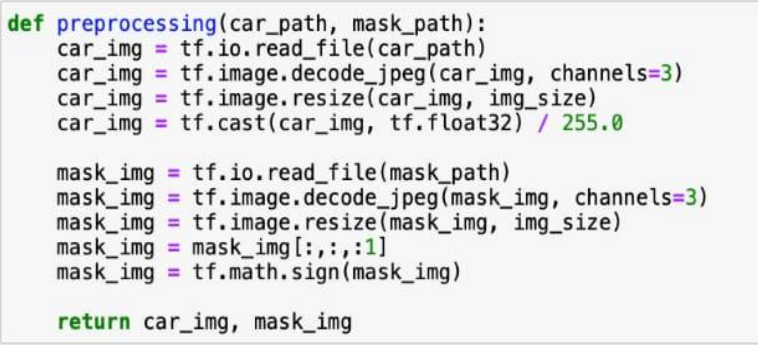


Setup Apache Spark → Configure Apache Airflow → Setup GitHub → Setup MLflow → Create FastAPI Endpoints → Setup Prometheus → Setup Grafana

Data preprocessing and training

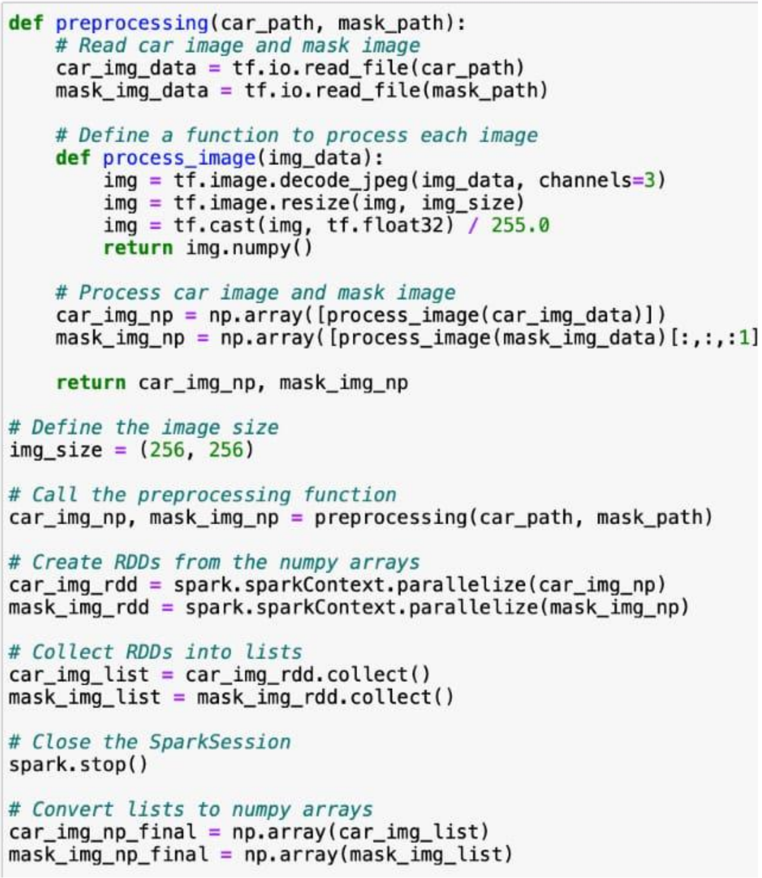
Apache Spark

Without Spark



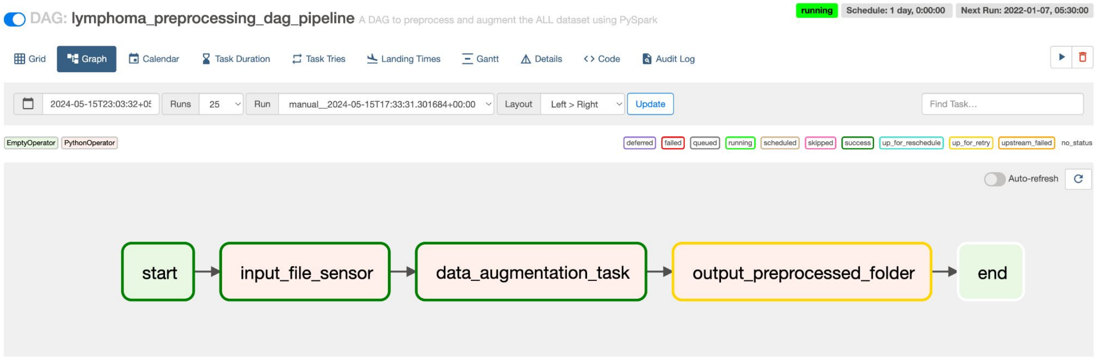
```
def preprocessing(car_path, mask_path):
    car_img = tf.io.read_file(car_path)
    car_img = tf.image.decode_jpeg(car_img, channels=3)
    car_img = tf.image.resize(car_img, img_size)
    car_img = tf.cast(car_img, tf.float32) / 255.0

    mask_img = tf.io.read_file(mask_path)
    mask_img = tf.image.decode_jpeg(mask_img, channels=3)
    mask_img = tf.image.resize(mask_img, img_size)
    mask_img = mask_img[:,:,:1]
    mask_img = tf.math.sign(mask_img)

    return car_img, mask_img
```

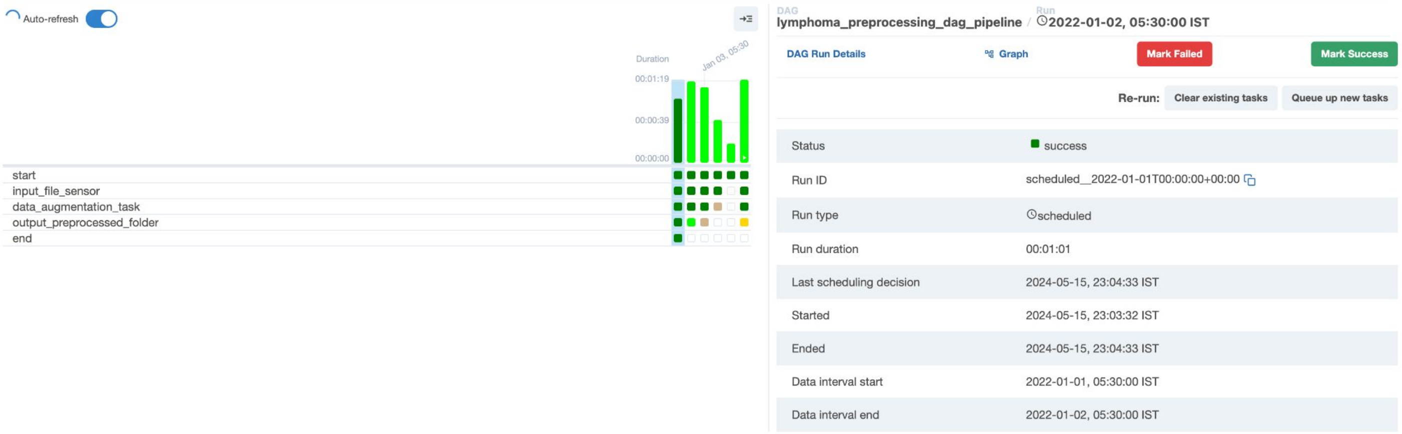
With Spark



```
def preprocessing(car_path, mask_path):
    # Read car image and mask image
    car_img_data = tf.io.read_file(car_path)
    mask_img_data = tf.io.read_file(mask_path)

    # Define a function to process each image
    def process_image(img_data):
        img = tf.image.decode_jpeg(img_data, channels=3)
        img = tf.image.resize(img, img_size)
        img = tf.cast(img, tf.float32) / 255.0
        return img.numpy()

    # Process car image and mask image
    car_img_np = np.array([process_image(car_img_data)])
    mask_img_np = np.array([process_image(mask_img_data)[:,:,:1]

    return car_img_np, mask_img_np

# Define the image size
img_size = (256, 256)

# Call the preprocessing function
car_img_np, mask_img_np = preprocessing(car_path, mask_path)

# Create RDDs from the numpy arrays
car_img_rdd = spark.sparkContext.parallelize(car_img_np)
mask_img_rdd = spark.sparkContext.parallelize(mask_img_np)

# Collect RDDs into lists
car_img_list = car_img_rdd.collect()
mask_img_list = mask_img_rdd.collect()

# Close the SparkSession
spark.stop()

# Convert lists to numpy arrays
car_img_np_final = np.array(car_img_list)
mask_img_np_final = np.array(mask_img_list)
```

Airflow



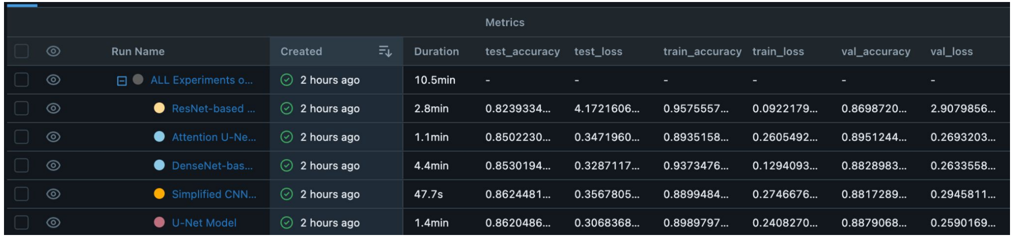


Status	🟩 success
Run ID	scheduled__2022-01-01T00:00:00+00:00
Run type	scheduled
Run duration	00:01:01
Last scheduling decision	2024-05-15, 23:04:33 IST
Started	2024-05-15, 23:03:32 IST
Ended	2024-05-15, 23:04:33 IST
Data interval start	2022-01-01, 05:30:00 IST
Data interval end	2022-01-02, 05:30:00 IST

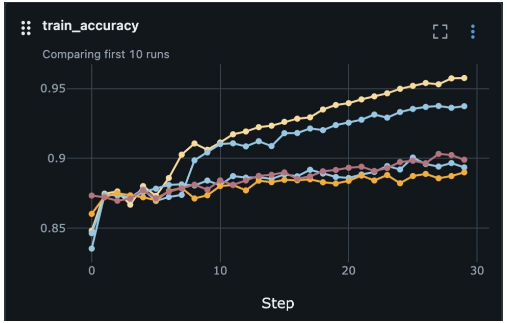
Use

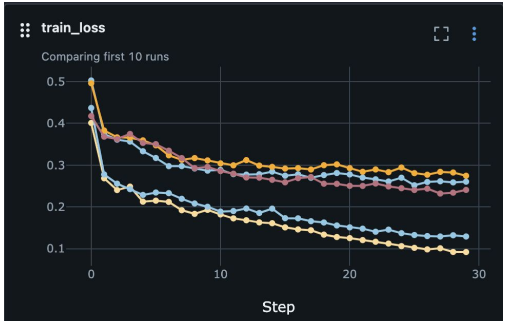
- Automation: Reduces manual effort and ensures consistent preprocessing and augmentation of images.
- Reproducibility: Provides a repeatable process that can be easily triggered and monitored.
- Flexibility: Can be adapted for different datasets and preprocessing requirements.

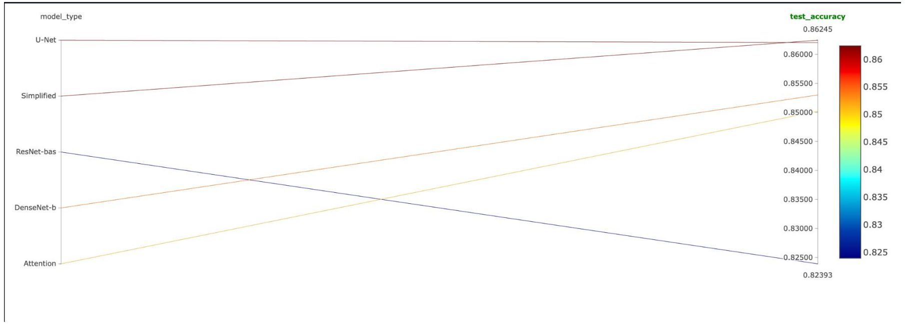
MLflow



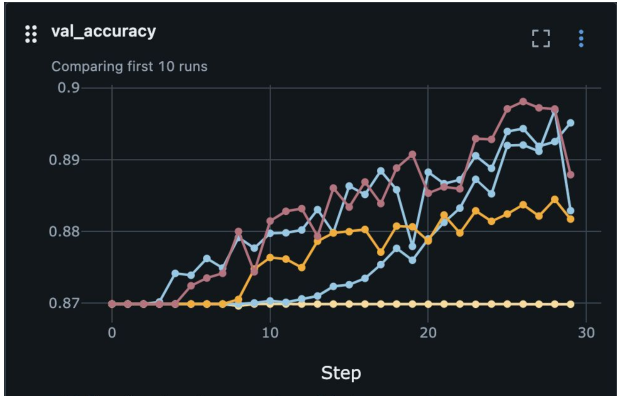
					Metrics					
□		Run Name	Created	Duration	test_accuracy	test_loss	train_accuracy	train_loss	val_accuracy	val_loss
□		⊟ ● ALL Experiments o...	✓ 2 hours ago	10.5min	-	-	-	-	-	-
□		● ResNet-based ...	✓ 2 hours ago	2.8min	0.8239334...	4.1721606...	0.9575557...	0.0922179...	0.8698720...	2.9079856...
□		● Attention U-Ne...	✓ 2 hours ago	1.1min	0.8502230...	0.3471960...	0.8935158...	0.2605492...	0.8951244...	0.2693203...
□		● DenseNet-bas...	✓ 2 hours ago	4.4min	0.8530194...	0.3287117...	0.9373476...	0.1294093...	0.8828983...	0.2633558...
□		● Simplified CNN...	✓ 2 hours ago	47.7s	0.8624481...	0.3567805...	0.8899484...	0.2746676...	0.8817289...	0.2945811...
□		● U-Net Model	✓ 2 hours ago	1.4min	0.8620486...	0.3068368...	0.8989797...	0.2408270...	0.8879068...	0.2590169...



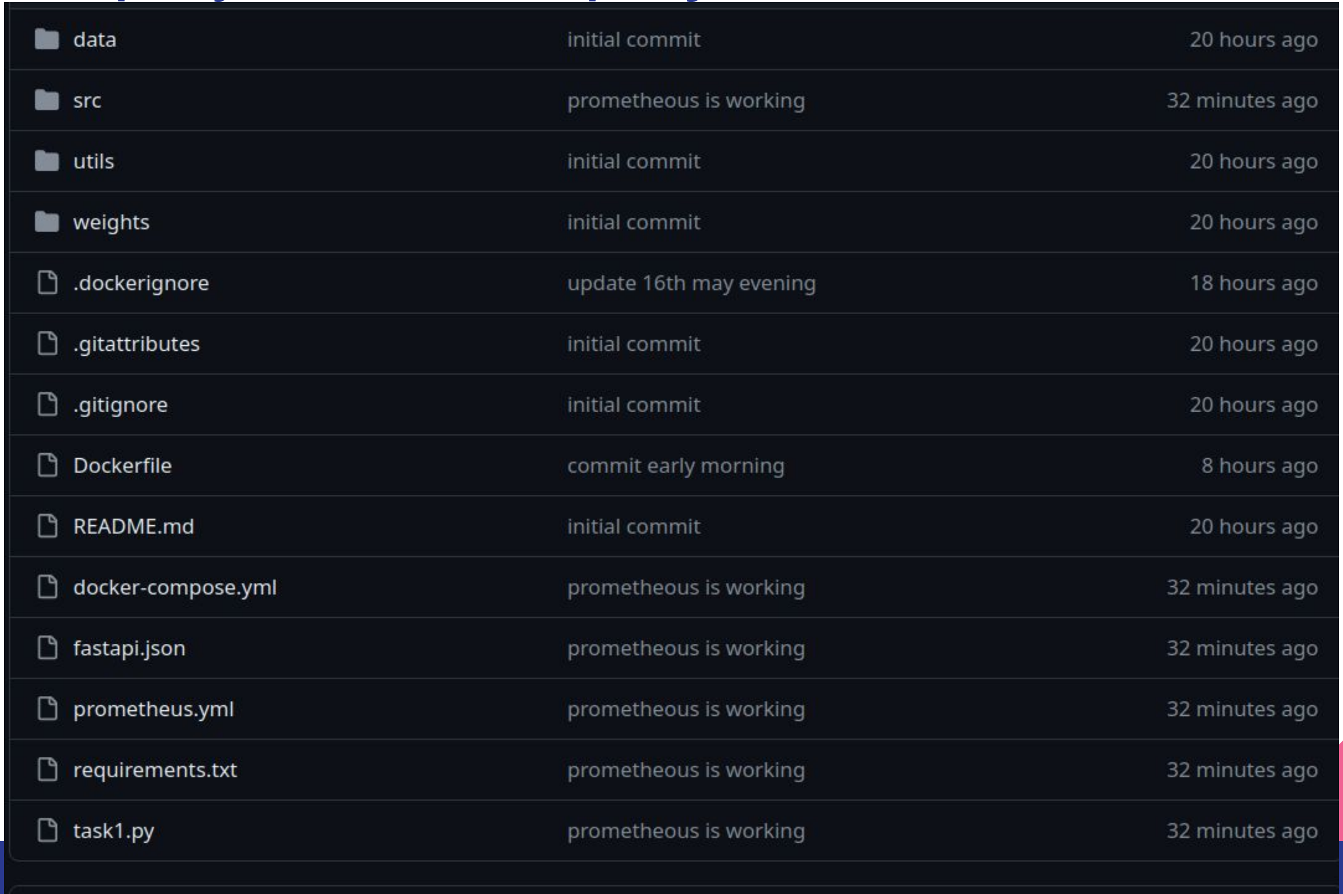




- The experiment tracking with MLflow provides valuable insights into the performance of different models
- DenseNet-based Model: Offers a balanced performance with a high training accuracy (93.73%) and a strong validation accuracy (88.29%). Its consistent metrics make it a reliable choice for further use.

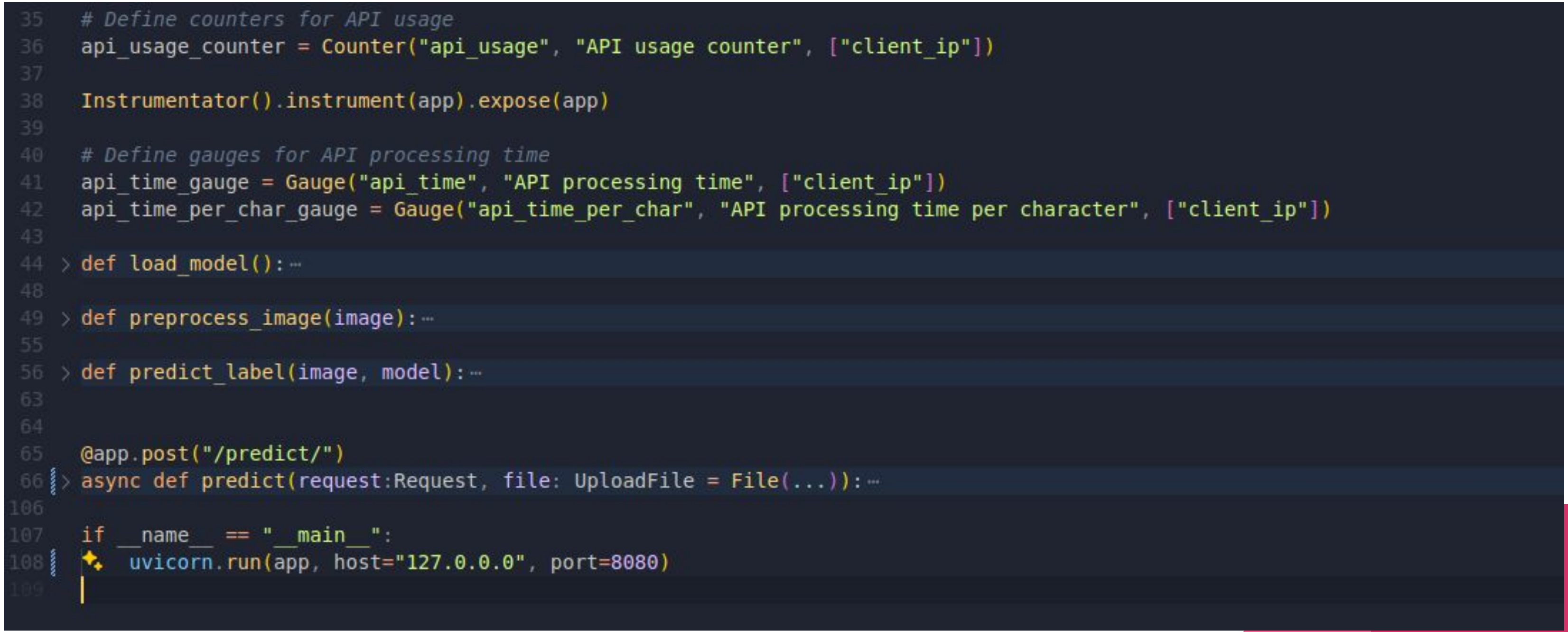


Deployment - the project



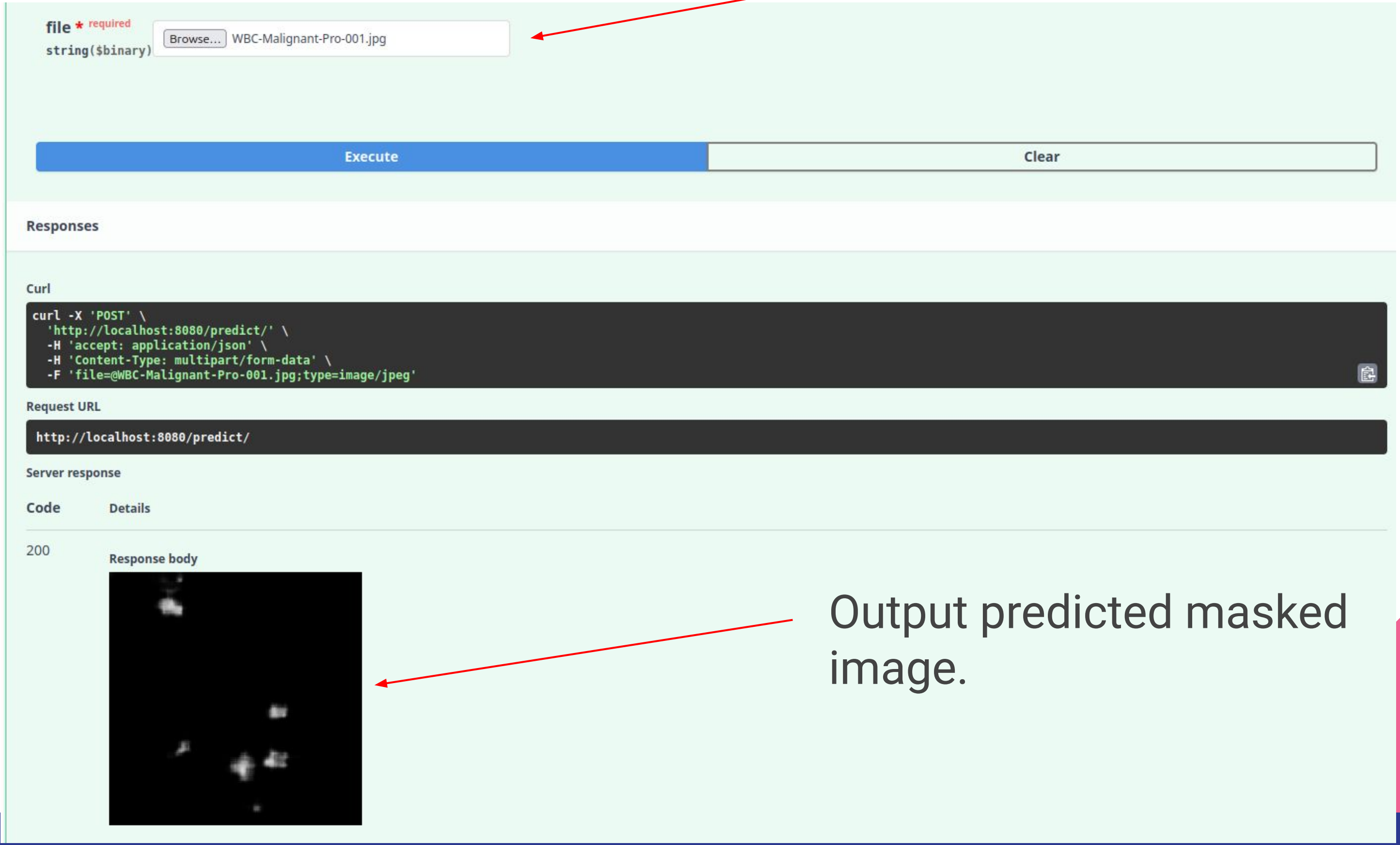
data	initial commit	20 hours ago
src	prometheous is working	32 minutes ago
utils	initial commit	20 hours ago
weights	initial commit	20 hours ago
.dockerignore	update 16th may evening	18 hours ago
.gitattributes	initial commit	20 hours ago
.gitignore	initial commit	20 hours ago
Dockerfile	commit early morning	8 hours ago
README.md	initial commit	20 hours ago
docker-compose.yml	prometheous is working	32 minutes ago
fastapi.json	prometheous is working	32 minutes ago
prometheus.yml	prometheous is working	32 minutes ago
requirements.txt	prometheous is working	32 minutes ago
task1.py	prometheous is working	32 minutes ago

Fast-api



```
# Define counters for API usage
api_usage_counter = Counter("api_usage", "API usage counter", ["client_ip"])

Instrumentator().instrument(app).expose(app)

# Define gauges for API processing time
api_time_gauge = Gauge("api_time", "API processing time", ["client_ip"])
api_time_per_char_gauge = Gauge("api_time_per_char", "API processing time per character", ["client_ip"])

def load_model():⋯

def preprocess_image(image):⋯

def predict_label(image, model):⋯


@app.post("/predict/")
async def predict(request:Request, file: UploadFile = File(...)):⋯

if __name__ == "__main__":
    uvicorn.run(app, host="127.0.0.0", port=8080)

```

Swagger-ui, client side



file * required
string($binary)

WBC-Malignant-Pro-001.jpg

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/predict/' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@WBC-Malignant-Pro-001.jpg;type=image/jpeg'
```

Request URL

```
http://localhost:8080/predict/
```

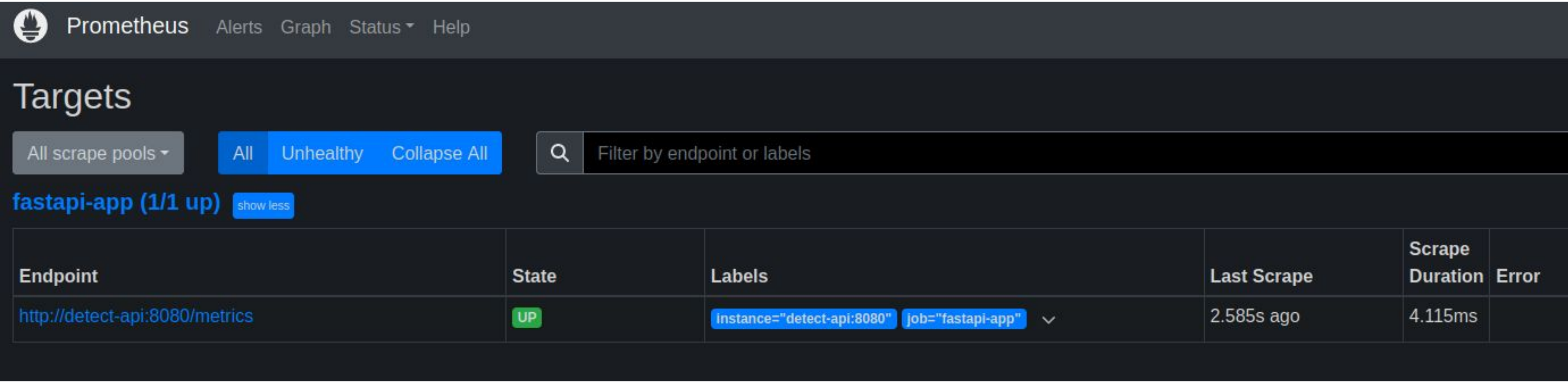
Server response

Code	Details
200	Response body

Input image

Output predicted masked image.

Prometheus

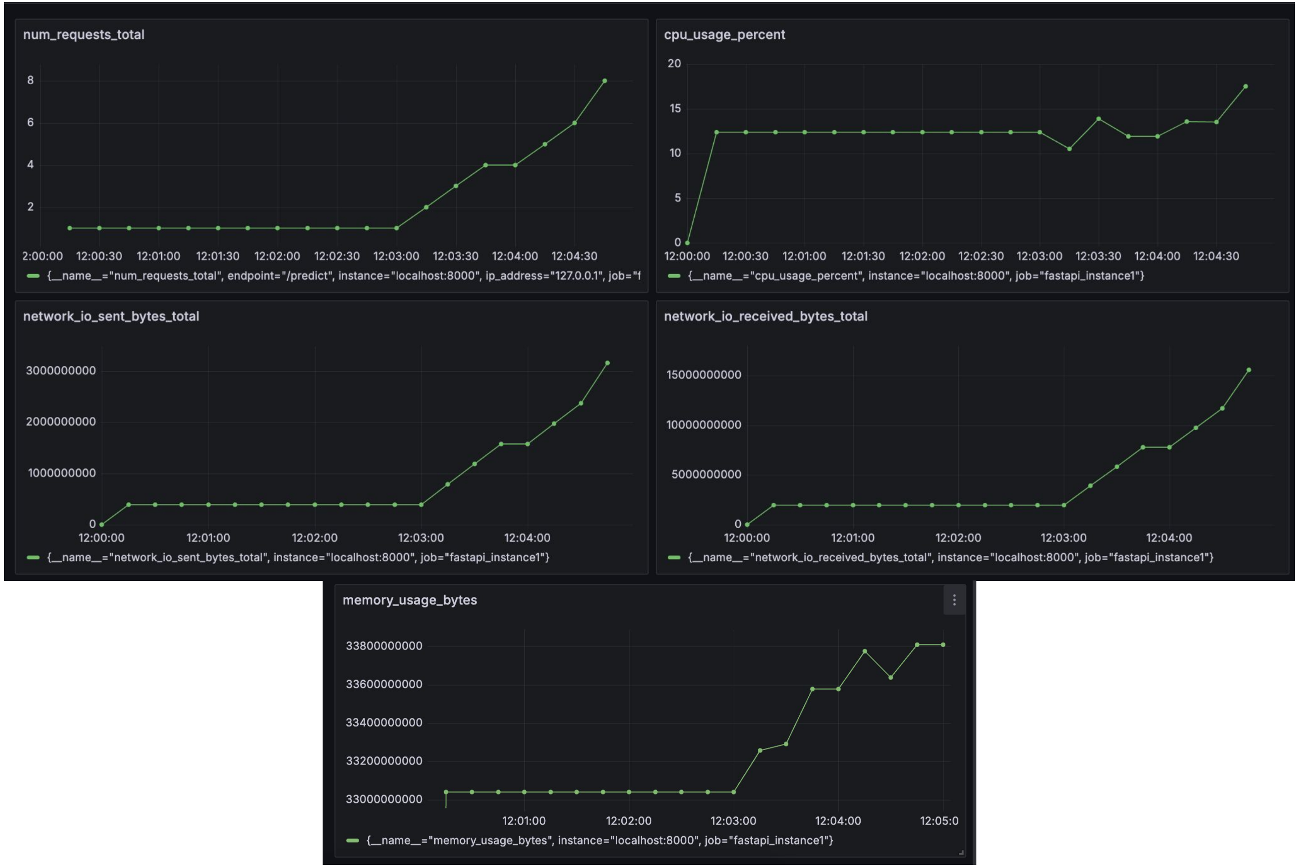


Targets

fastapi-app (1/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://detect-api:8080/metrics	UP	instance="detect-api:8080" job="fastapi-app"	2.585s ago	4.115ms	

Graphana


num_requests_total
8
6
4
2
2:00:00
12:00:30
12:01:00
12:01:30
12:02:00
12:02:30
12:03:00
12:03:30
12:04:00
12:04:30
{__name__="num_requests_total", endpoint="/predict", instance="localhost:8000", ip_address="127.0.0.1", job="f
cpu_usage_percent
20
15
10
5
0
12:00:00
12:00:30
12:01:00
12:01:30
12:02:00
12:02:30
12:03:00
12:03:30
12:04:00
12:04:30
{__name__="cpu_usage_percent", instance="localhost:8000", job="fastapi_instance1"}
network_io_sent_bytes_total
3000000000
2000000000
1000000000
0
12:00:00
12:01:00
12:02:00
12:03:00
12:04:00
{__name__="network_io_sent_bytes_total", instance="localhost:8000", job="fastapi_instance1"}
network_io_received_bytes_total
15000000000
10000000000
5000000000
0
12:00:00
12:01:00
12:02:00
12:03:00
12:04:00
{__name__="network_io_received_bytes_total", instance="localhost:8000", job="fastapi_instance1"}
memory_usage_bytes
33800000000
33600000000
33400000000
33200000000
33000000000
12:01:00
12:02:00
12:03:00
12:04:00
12:05:0
{__name__="memory_usage_bytes", instance="localhost:8000", job="fastapi_instance1"}

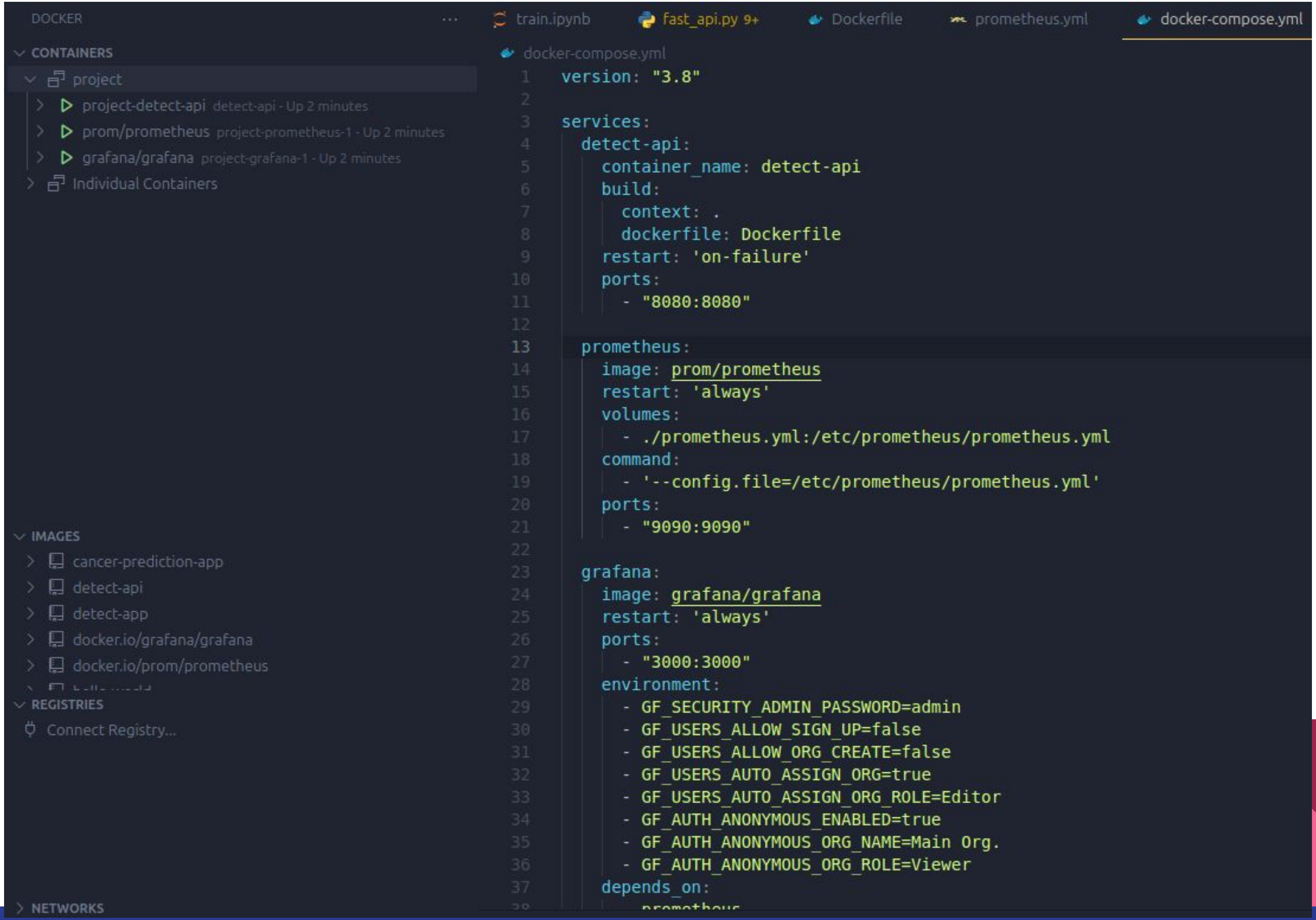
Metric 1: Number of Requests: Tracks the total number of requests received by the FastAPI server.ions.

Metric 2: CPU Usage Percentage: Monitors the percentage of CPU utilization by the FastAPI application.

Metric 3: Network I/O (Sent and Received Bytes): Measures the total bytes sent by the server, reflecting the volume of data served to the clients.

Metric 4: Memory Usage: Tracks the memory usage in bytes of the FastAPI application.

Docker



```yaml
version: "3.8"

services:
  detect-api:
    container_name: detect-api
    build:
      context: .
      dockerfile: Dockerfile
    restart: 'on-failure'
    ports:
      - "8080:8080"

  prometheus:
    image: prom/prometheus
    restart: 'always'
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
    ports:
      - "9090:9090"

  grafana:
    image: grafana/grafana
    restart: 'always'
    ports:
      - "3000:3000"
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin
      - GF_USERS_ALLOW_SIGN_UP=false
      - GF_USERS_ALLOW_ORG_CREATE=false
      - GF_USERS_AUTO_ASSIGN_ORG=true
      - GF_USERS_AUTO_ASSIGN_ORG_ROLE=Editor
      - GF_AUTH_ANONYMOUS_ENABLED=true
      - GF_AUTH_ANONYMOUS_ORG_NAME=Main Org.
      - GF_AUTH_ANONYMOUS_ORG_ROLE=Viewer
    depends_on:
      - prometheus
```

Git

- Our project had lots of training data, ~190 MB.
- Used Git - LFS for tracking changes in data
- And the whole project is tracked using Git version control.
- Github link:

Contributions

Version control: **Git** and **Git-lfs**

Vishal V (ME20B204):

Apache Airflow Preprocessing Pipeline, Experiments Tracking using **MLFlow**, Monitoring Dashboard using **Grafana**

Akranth (ME20B100):

Deployed APIs with **FastAPI**, tracking the API usage with **prometheus** and visualization with **grafana**, containerized the whole thing using **docker**. Tracking the project using git.

Sai Gowtham Tamminaina (ED19B063)

Apache Spark and Airflow for the Preprocessing Pipeline