Exp No. 8 Date:	Monitor and logging in Machine Learning with Python	REG. NO: URK23CS1197

Objective:

The objective is to ensure the machine learning models are performing as expected by tracking their performance, logging relevant metrics, and diagnosing any issues that arise during training and deployment.

Tools/ Software Required:

- pandas
- scikit-learn
- joblib
- pytest
- fastapi
- uvicorn
- python-multipart

Job Role:

Machine Learning Engineer / Data Scientist

Skills Required:

- **Proficiency in Python:** Strong understanding of Python programming language.
- **Machine Learning Frameworks:** Familiarity with frameworks like TensorFlow, PyTorch, or Scikit-learn.
- **Logging Libraries:** Knowledge of Python logging libraries (e.g., logging module) and ML-specific logging tools (e.g., MLflow).
- **Monitoring Tools**: Experience with monitoring tools and platforms (e.g., Prometheus, Grafana).
- **Data Analysis:** Ability to analyze logged data to identify trends and issues.
- **Debugging:** Strong debugging skills to troubleshoot and resolve issues in ML models.

Prerequisites:

- **Basic Understanding of Machine Learning:** Knowledge of machine learning concepts and algorithms.
- **Experience with Python:** Comfortable working with Python for data manipulation and analysis.
- **Familiarity with ML Frameworks:** Prior experience with at least one machine learning framework.
- **Basic Knowledge of Logging and Monitoring:** Understanding of how to implement logging and monitoring in software development.

Description:

Logging is a method of tracking and storing data to ensure application availability and to assess the impact of state transformations on performance. The purpose of logging is to create an ongoing record of application events. Log files can be used to review any event within a system, including failures and state transformations. Consequently, log messages can provide valuable information to help pinpoint the cause of performance problems. Log data can help DevOps teams troubleshoot issues by identifying which changes resulted in error reporting, but is only as valuable as the information it contains.

Monitoring is a diagnostic tool used for alerting DevOps to system-related issues by analysing metrics. Monitoring metrics is like the security alarm that alerts you to a possible intrusion; log files act as the security camera footage that will provide clues to tell you what happened and how.

Commands:
Logging

```
import logging

# Create a custom logger
logger = logging.getLogger(__name__)

# Set the logging level
logger.setLevel(logging.INFO)

# Create handlers
c_handler = logging.StreamHandler()
f_handler = logging.FileHandler('file.log')

# Create formatters and add them to handlers
c_format = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
f_format = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')

c_handler.setFormatter(c_format)
f_handler.setFormatter(f_format)

# Add handlers to the logger
logger.addHandler(c_handler)
logger.addHandler(f_handler)

# Example logging
logger.info('This is an info message')
logger.error('This is an error message')
```

Monitoring

```
import tensorflow as tf
from tensorflow.keras.callbacks import TensorBoard


# Load your dataset
```

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Preprocess your data
x_train, x_test = x_train / 255.0, x_test / 255.0

# Create your model
model = tf.keras.models.Sequential([
tf.keras.layers.Flatten(input_shape=(28, 28)),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dropout(0.2),

tf.keras.layers.Dense(10)
])

# Compile your model
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])

# Set up TensorBoard callback
tensorboard_callback = TensorBoard(log_dir="./logs")

# Train your model with TensorBoard callback
model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test),
callbacks=[tensorboard_callback])
```

To visualize the logs, you can run TensorBoard in your terminal:

bash

```
tensorboard --logdir=./logs
```

Questions:

1.Write a Python script using the logging module to log the training loss and accuracy of a neural network model at each epoch.

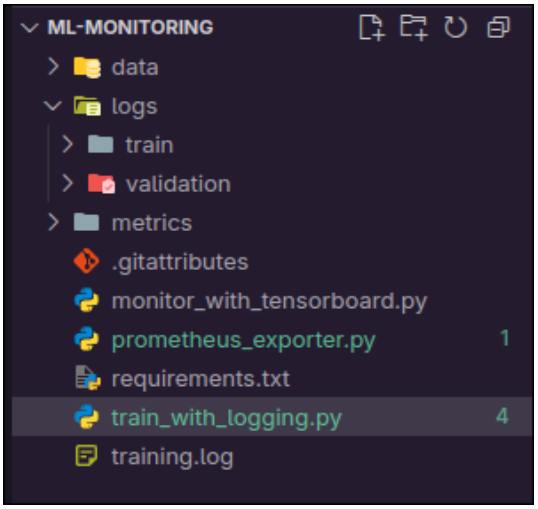
2. Design and implement a custom monitoring and logging system for a machine learning project using Python, TensorBoard, and Prometheus.

Program:

Project Structure:

Ex8_ML_Monitoring/

```
├── metrics/
├── logs/
├── train_with_logging.py
├── monitor_with_tensorboard.py
└── prometheus_exporter.py
```



requirements.txt

```
tensorflow
numpy
prometheus-client
tensorboard
protobuf
grpcio
```

monitor_with_tensorboard.py

```
import tensorflow as tf
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10)
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
```

```
        metrics=['accuracy'])

tensorboard_callback = TensorBoard(log_dir="logs/")

model.fit(x_train, y_train, validation_data=(x_test, y_test),
          epochs=5, callbacks=[tensorboard_callback])
```

prometheus_exported.py

```
from prometheus_client import start_http_server, Gauge
import tensorflow as tf
from tensorflow.keras.datasets import mnist
import time

print("✅ Prometheus Metrics Exporter Running at http://127.0.0.1:8000/metrics")

loss_metric = Gauge("training_loss", "Training Loss")
accuracy_metric = Gauge("training_accuracy", "Training Accuracy")

start_http_server(8000)

(x_train, y_train), _ = mnist.load_data()
x_train = x_train / 255.0

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(28,28)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

for epoch in range(5):
    history = model.fit(x_train, y_train, epochs=1, verbose=0)
    loss = history.history['loss'][0]
    accuracy = history.history['accuracy'][0]
    loss_metric.set(loss)
    accuracy_metric.set(accuracy)
    print(f"Epoch {epoch+1} | Loss: {loss:.4f}, Accuracy: {accuracy:.4f}")

# ✅ Keep server running forever
while True:
    time.sleep(1)
```

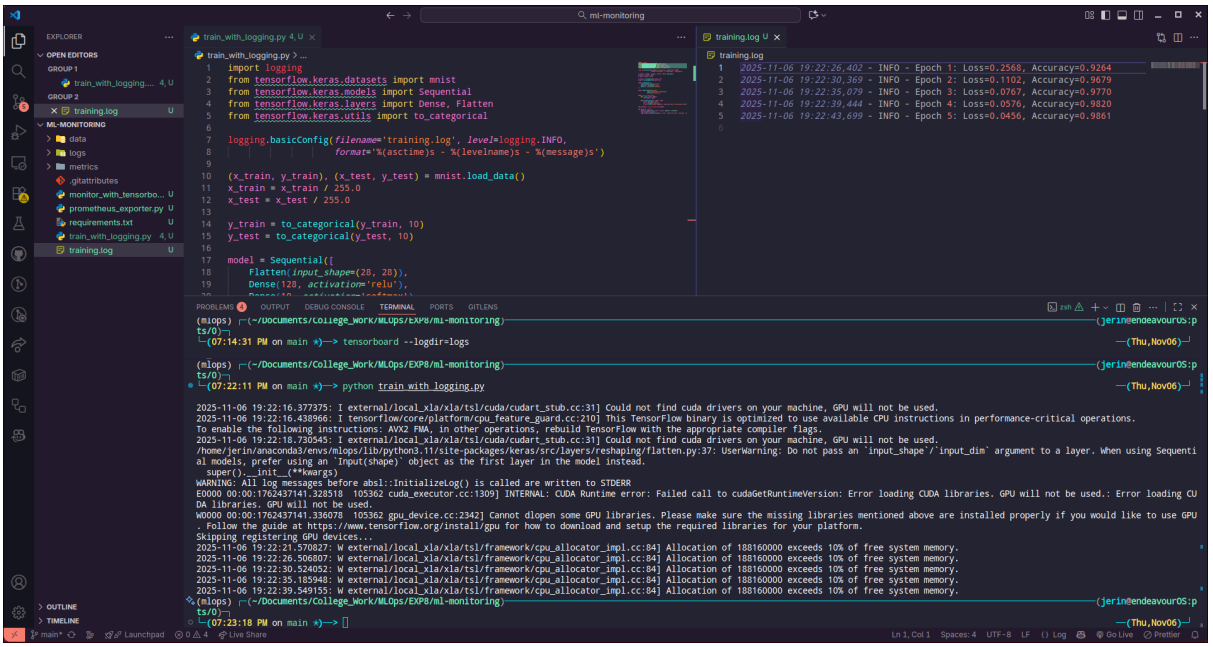
train_with_logging.py

```
import logging
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical

logging.basicConfig(filename='training.log', level=logging.INFO,
                    format='%(asctime)s - %(levelname)s - %(message)s')

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

class LogTrainingCallback:
    def __init__(self, model):
        self.model = model

    def on_epoch_end(self, epoch, logs):
        loss = logs.get('loss')
        acc = logs.get('accuracy')
        logging.info(f"Epoch {epoch+1}: Loss={loss:.4f}, Accuracy={acc:.4f}")

log_callback = LogTrainingCallback(model)

for epoch in range(5):
    history = model.fit(x_train, y_train, epochs=1, verbose=0)
    logs = history.history
    log_callback.on_epoch_end(epoch, {'loss': logs['loss'][0], 'accuracy': logs['accuracy'][0]})
```

Expected Output :

Q1.

python train_with_logging.py

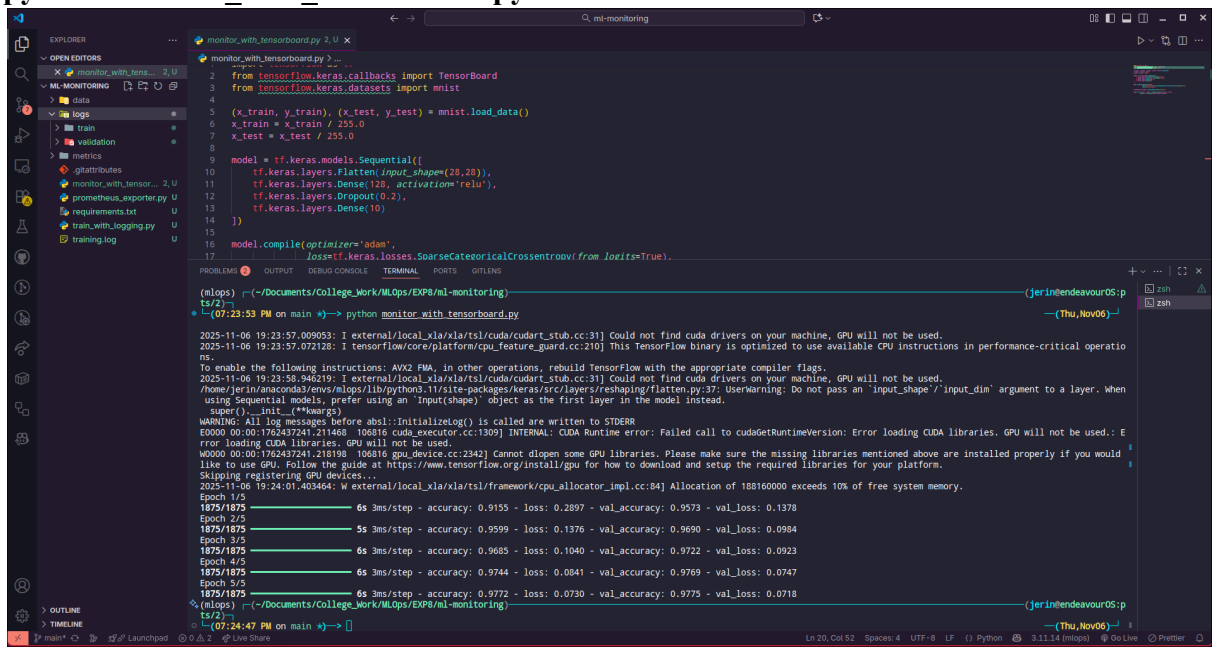


training.log

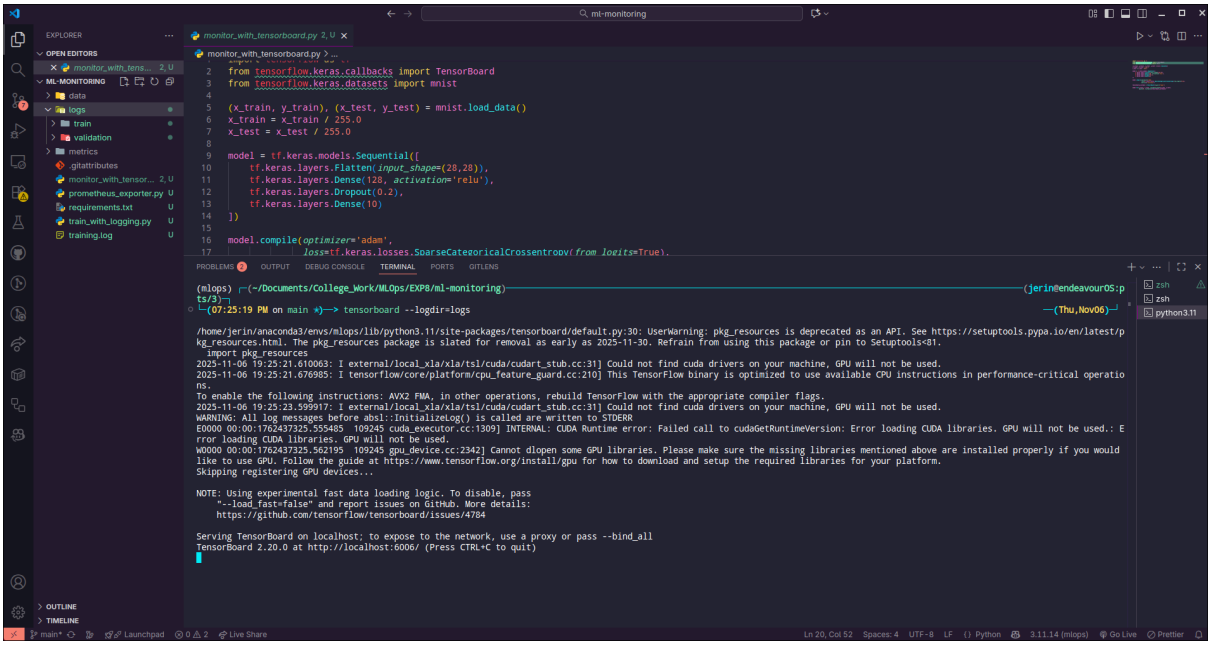
2025-11-06 19:22:26,402 - INFO - Epoch 1: Loss=0.2568, Accuracy=0.9264
2025-11-06 19:22:30,369 - INFO - Epoch 2: Loss=0.1102, Accuracy=0.9679
2025-11-06 19:22:35,079 - INFO - Epoch 3: Loss=0.0767, Accuracy=0.9770
2025-11-06 19:22:39,444 - INFO - Epoch 4: Loss=0.0576, Accuracy=0.9820
2025-11-06 19:22:43,699 - INFO - Epoch 5: Loss=0.0456, Accuracy=0.9861

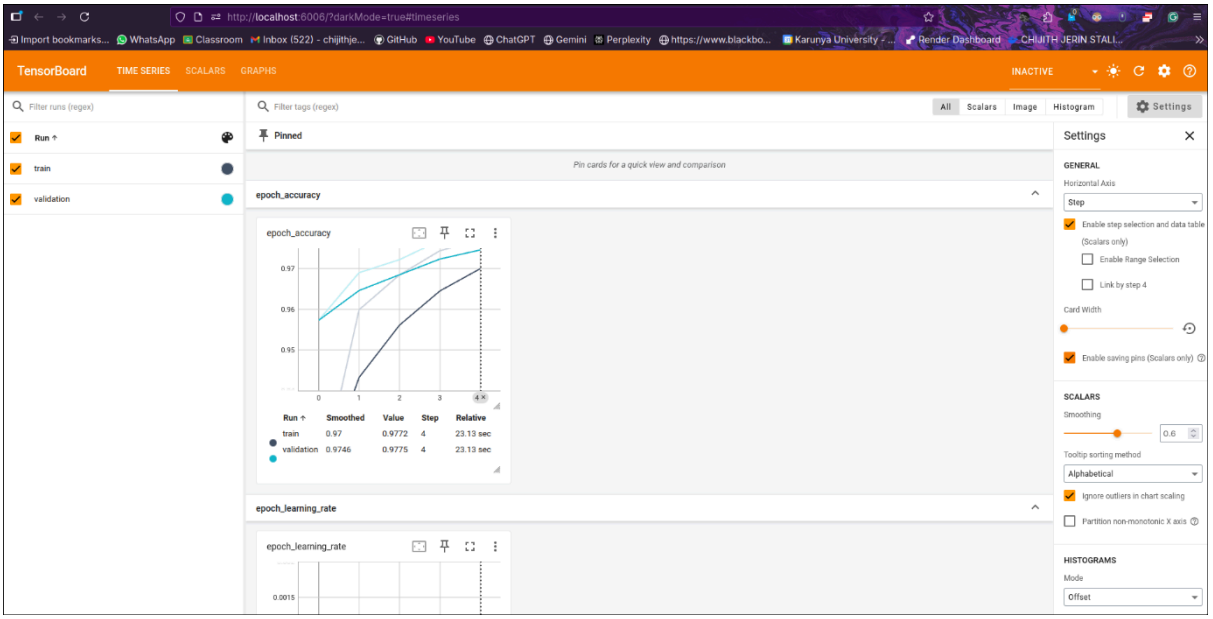
Q2.

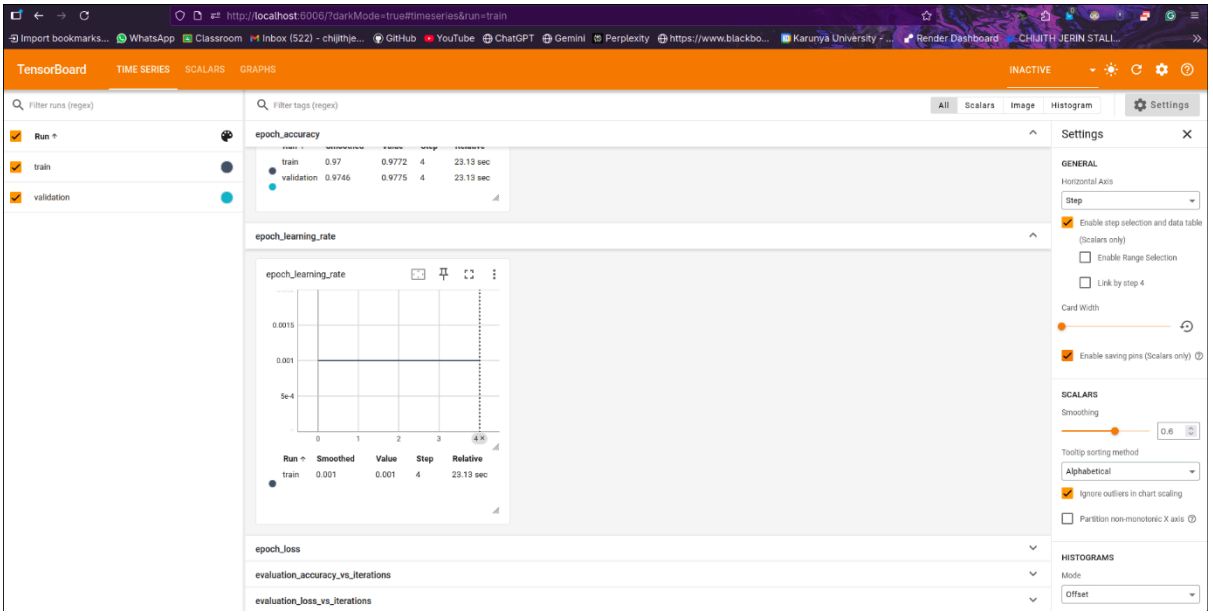
python monitor_with_tensorboard.py

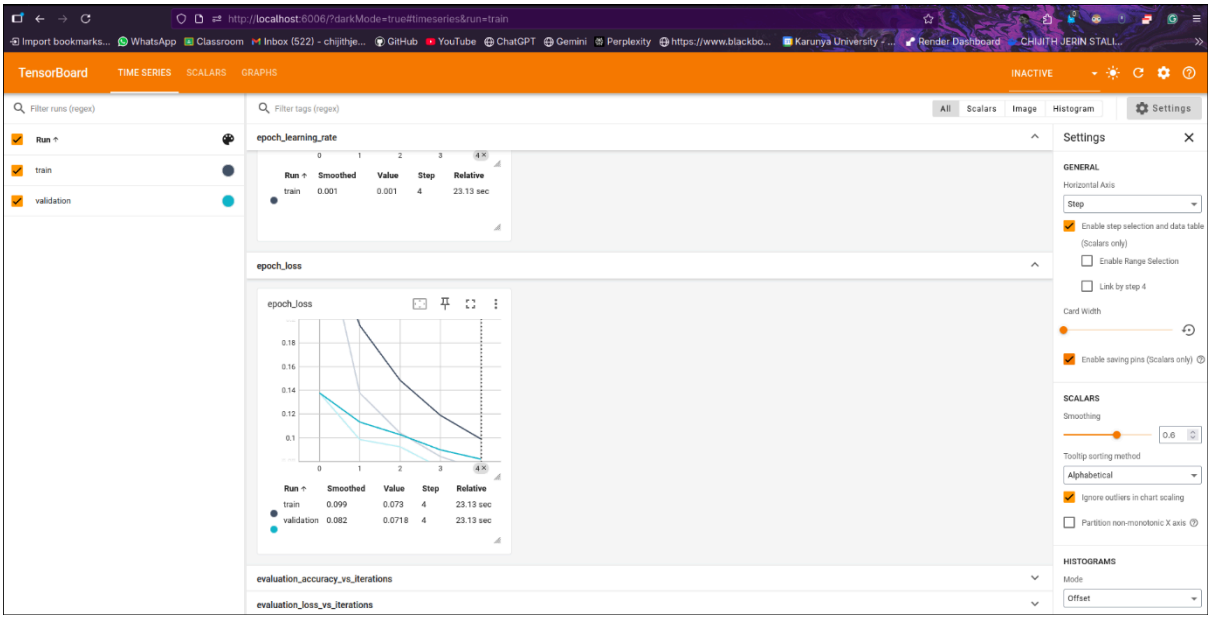


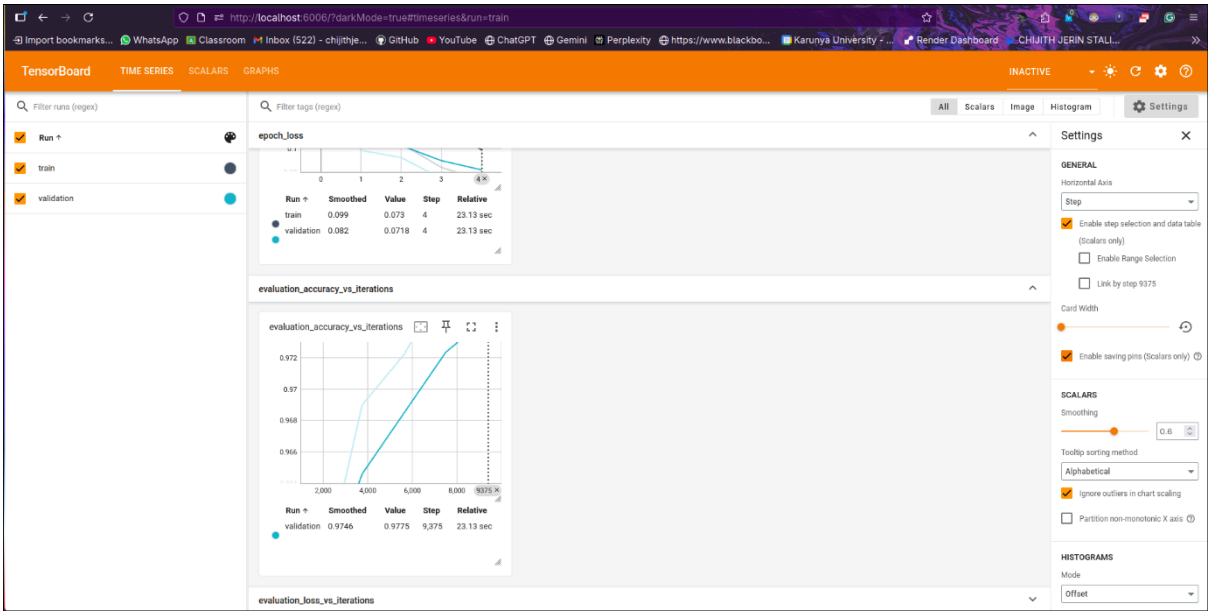
Tensorboard
tensorboard --logdir=logs

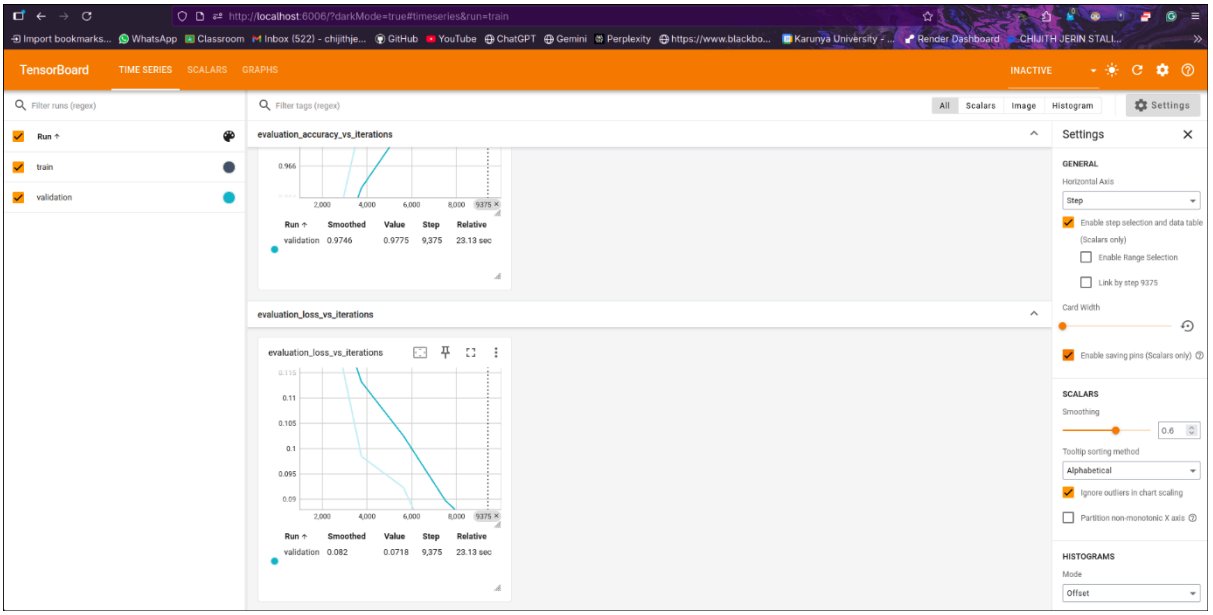



TensorBoard
TIME SERIES
SCALARS
GRAPHS
INACTIVE
Filter runs (regex)
Filter tags (regex)
All
Scalars
Image
Histogram
Settings
Run
train
validation
Pinned
Pin cards for a quick view and comparison
epoch_accuracy
epoch_learning_rate
Run Smoothed Value Step Relative
train 0.97 0.9772 4 23.13 sec
validation 0.9746 0.9775 4 23.13 sec
GENERAL
Horizontal Axis
Step
Enable step selection and data table
(Scalars only)
Enable Range Selection
Link by step 4
Card Width
Enable saving pins (Scalars only)
SCALARS
Smoothing
0.6
Tooltip sorting method
Alphabetical
Ignore outliers in chart scaling
Partition non-monotonic X axis
HISTOGRAMS
Mode
Offset

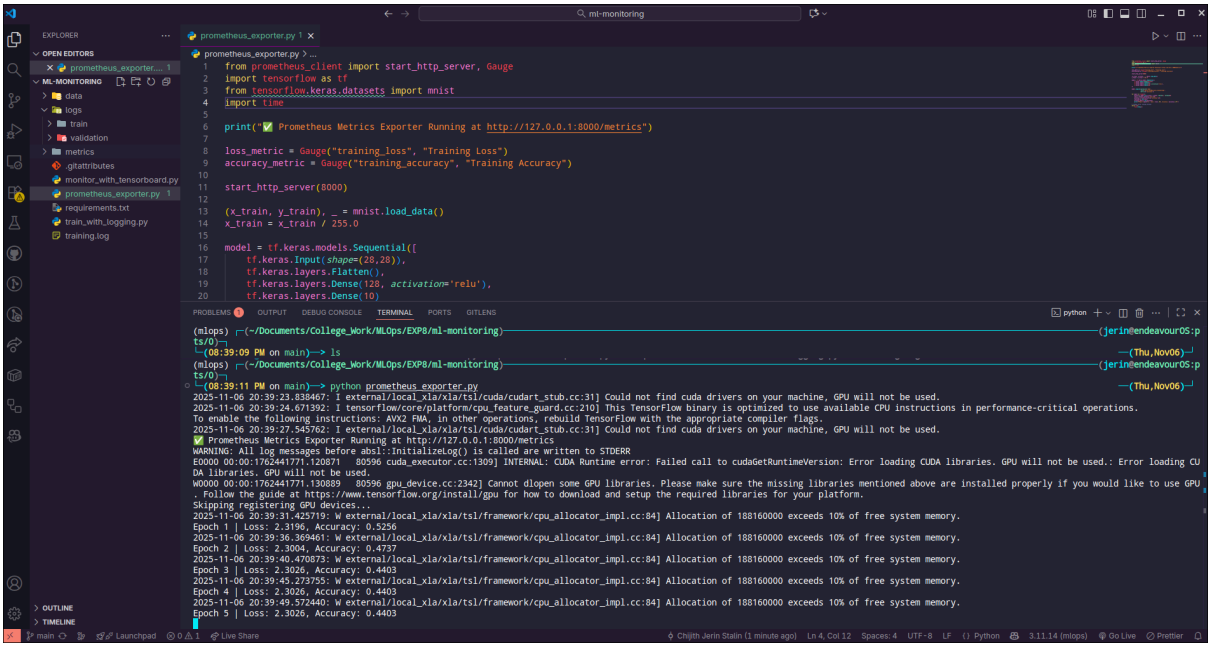

TensorBoard
TIME SERIES
SCALARS
GRAPHS
INACTIVE
Filter runs (regex)
Filter tags (regex)
All
Scalars
Image
Histogram
Settings
Run
train
validation
epoch_accuracy
epoch_learning_rate
Run Smoothed Value Step Relative
train 0.001 0.001 4 23.13 sec
epoch_loss
evaluation_accuracy_vs_iterations
evaluation_loss_vs_iterations
GENERAL
Horizontal Axis
Step
Enable step selection and data table
(Scalars only)
Enable Range Selection
Link by step 4
Card Width
Enable saving pins (Scalars only)
SCALARS
Smoothing
0.6
Tooltip sorting method
Alphabetical
Ignore outliers in chart scaling
Partition non-monotonic X axis
HISTOGRAMS
Mode
Offset

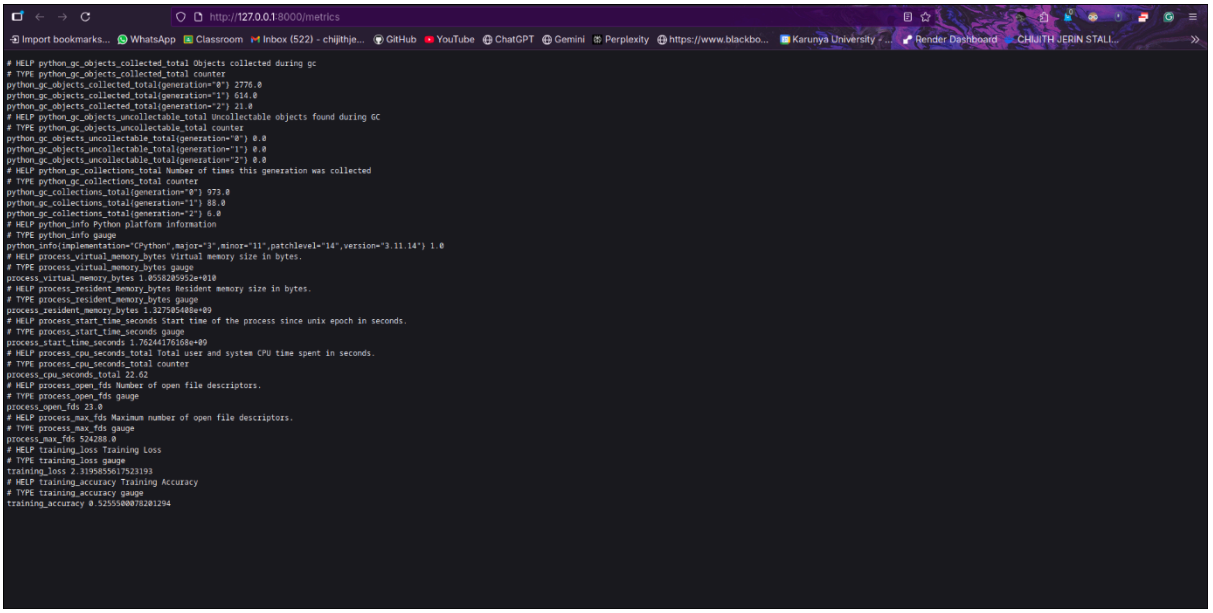

TensorBoard
TIME SERIES
SCALARS
GRAPHS
INACTIVE
Run
train
validation
epoch_learning_rate
epoch_loss
evaluation_accuracy_vs_iterations
evaluation_loss_vs_iterations
Settings
GENERAL
SCALARS
HISTOGRAMS


TensorBoard
TIME SERIES
SCALARS
GRAPHS
INACTIVE
Run
train
validation
epoch_loss
evaluation_accuracy_vs_iterations
evaluation_loss_vs_iterations
Settings
GENERAL
SCALARS
HISTOGRAMS



python prometheus_exporter.py





Result :

The machine learning model's training loss and accuracy were successfully logged and monitored using Python logging, TensorBoard, and Prometheus. This enabled real-time observation of model performance and ensured correct behavior during the training process.