To monitor a deployed ML model with MLflow, you'll need to set up MLflow Tracking to log metrics, predictions, and other relevant data generated during inference. Here's a step-by-step breakdown of how to use MLflow for monitoring an inference model that is already deployed.

1. Set Up MLflow Tracking Server

- If MLflow Tracking Server is not yet configured, set it up on your on-premise infrastructure.
- You can run MLflow Tracking as a standalone server with a backend database (like PostgreSQL or MySQL) and an artifact store (e.g., a file system).
- Start the tracking server by running:

bash

Copy code

```
mlflow server --backend-store-uri <database-uri> --default-artifact-root <artifact-storage-uri> --host 0.0.0.0 --port 5000
```

2. Configure the Model for Logging with MLflow

- Update your inference code to log predictions, input features, and any additional performance metrics to MLflow.
- Set the MLflow tracking URI to the address where the MLflow Tracking Server is running:

python

Copy code

```
import mlflow

mlflow.set_tracking_uri("http://localhost:5000")
```

3. Log Model Predictions and Metrics in Inference Code

- Modify the deployed model's inference pipeline to log predictions and input data at runtime. This typically involves:
    - **Input Features**: Log input features for monitoring data drift.
    - **Predictions**: Log predictions to track the model's output over time.

- **Custom Metrics**: Log metrics such as accuracy, latency, and errors to monitor model performance.

- Here's an example of logging these values using MLflow:

python

Copy code

```
import mlflow


# Wrap inference code to log data and predictions
def predict_and_log(model, input_data):
  # Start a new MLflow run
  with mlflow.start_run():
    # Record the input data (be cautious with large inputs for storage reasons)
    mlflow.log_params({"input_data": input_data})


    # Get the model's prediction
    prediction = model.predict(input_data)


    # Log the prediction
    mlflow.log_metric("prediction", prediction)


    # Additional metrics
    inference_time = calculate_inference_time()
    mlflow.log_metric("inference_time", inference_time)


  return prediction
```

4. Monitor Drift by Logging Input Features

- Regularly log the input data features or summaries (such as mean, standard deviation) to track data drift over time.
- By examining feature distribution changes in MLflow, you can detect shifts in the data used during inference, which could impact model accuracy.
- Example:

python

Copy code

```
import numpy as np


def log_feature_statistics(input_data):
  mlflow.log_metric("input_mean", np.mean(input_data))
  mlflow.log_metric("input_stddev", np.std(input_data))
```

5. Track and Visualize Performance Over Time in MLflow UI

- Use the MLflow UI to observe logged metrics, input distributions, and predictions over time. This allows you to identify patterns, trends, or anomalies in the model's behavior.
- Set alerts based on specific metrics or thresholds within MLflow or using a custom script to regularly check metrics for unusual values.

6. Comparing Model Versions for Performance Degradation

- If you update your model, MLflow allows you to log different model versions, making it easier to compare performance metrics between versions.
- You can use these comparisons to check for issues like accuracy degradation or data drift, indicating when retraining may be necessary.

7. Automate Logging and Alerting (Optional)

- If you want to automate this monitoring, you can write a script or cron job that checks MLflow metrics and sets up alerts when certain thresholds are exceeded.
- For example, if the accuracy falls below a threshold or latency goes above a limit, an alert can be triggered.

Example Workflow for Monitoring

Here's a possible flow of steps to monitor inference with MLflow in production:

1. **Inference**: As predictions are made, log inputs, outputs, and other metrics in MLflow.
2. **Drift Detection**: Track summary statistics of input features to detect drift over time.
3. **Performance Monitoring**: Observe response time, errors, or other latency metrics that indicate the model's health.
4. **Comparative Analysis**: Use version tracking in MLflow to identify if performance degradation happens in newer models.

Example of an Inference Monitoring Code Block

Here's a consolidated example code for monitoring inference using MLflow:

python

Copy code

```
import mlflow

import numpy as np


def monitor_inference(model, input_data):

  mlflow.set_tracking_uri("http://localhost:5000")


  with mlflow.start_run():

    # Log input features statistics

    mlflow.log_metric("input_mean", np.mean(input_data))

    mlflow.log_metric("input_stddev", np.std(input_data))


    # Perform prediction and log it

    prediction = model.predict(input_data)

    mlflow.log_metric("prediction", prediction)
```

```python
    # Log custom metrics (e.g., inference latency)

    inference_time = calculate_inference_time()

    mlflow.log_metric("inference_time", inference_time)


  return prediction
```

By following these steps and leveraging MLflow’s UI, you can gain insights into model performance, detect data drift, and troubleshoot issues for an on-premise deployed model.