📊 MLflow Usage Tutorial (Internal Guide)

This guide provides a complete walkthrough of how to use **MLflow** for experiment tracking in our internal ML infrastructure. It covers:

- How to connect to our **internal MLflow server**
- How to track experiments **during** and **after** training
- How to track classic ML, Hugging Face transformers, and inference-only use cases
- How to **log custom preprocessing code and comments** as artifacts
- Understanding the MLflow UI
- Markdown and Jupyter Notebook examples provided

🔐 1. Connecting to Internal MLflow Server

All tracking is done on our **central MLflow instance** (ask your lead for access credentials). You should be connected to VPN.

```
# —— MLflow connection ——————————————————————————————————————————
import os, mlflow
MLFLOW_TRACKING_URI = "http://mlflow.daniam.am"
mlflow.set_tracking_uri(MLFLOW_TRACKING_URI)

os.environ["MLFLOW_TRACKING_USERNAME"] = "your_username"
os.environ["MLFLOW_TRACKING_PASSWORD"] = "your_password"

mlflow.set_experiment("Your-Experiment-Name")
```

🟨 2. Tracking After Training

If you forgot to use MLflow during training, log all relevant outputs after the experiment has completed:

```
# train normally
model = train_model(...)
metrics = evaluate(model)

# later: open or create run

run = mlflow.start_run(run_name="descriptive-run-name")
mlflow.log_metric("f1", metrics["f1"])
mlflow.sklearn.log_model(model, "model")

mlflow.end_run()
```

OR

```python
# Assume results already saved locally
with mlflow.start_run(run_name="posthoc-run"):
    mlflow.log_param("data_split", "2024-week-18")
    mlflow.log_metric("f1_score", 0.821)
    mlflow.log_artifact("configs/final_config.yaml")
    mlflow.log_artifact("plots/confusion_matrix.png")
    mlflow.set_tag("note", "Logged after experiment completion")
```

If you forgot to add something to your run find the run-id in Mlflow website and restart the run

```python
mlflow.start_run(run_id="abc123")   # resumes that run
# add everything you forgot
mlflow.end_run()
```

🟩3. Tracking During Run

```
# Start an MLflow run

with mlflow.start_run():

    # Log parameters

    learning_rate = 0.01

    epochs = 10

    random_state = 42


    mlflow.log_param("learning_rate", learning_rate)

    mlflow.log_param("epochs", epochs)

    mlflow.log_param("random_state", random_state)

    # Load data

    iris = load_iris()

    X, y = iris.data, iris.target

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=random_state)


    # Train a classic ML model (Logistic Regression)

    model = LogisticRegression(solver='liblinear', random_state=random_state, max_iter=epochs)


    # Simulate training with intermediate logging

    for epoch in range(epochs):

        model.fit(X_train, y_train)

        y_pred = model.predict(X_test)

        accuracy = accuracy_score(y_test, y_pred)

        precision = precision_score(y_test, y_pred, average='weighted', zero_division=0)

        recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)

        f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0)


        # Log metrics for each epoch

        mlflow.log_metric("accuracy", accuracy, step=epoch)

        mlflow.log_metric("precision", precision, step=epoch)

        mlflow.log_metric("recall", recall, step=epoch)
```

```
mlflow.log_metric("f1_score",f1,step=epoch)print(f"Epoch {epoch+1}/{epochs}: Accuracy={accuracy:.4f}")# Log the final modelmlflow.sklearn.log_model(model,"logistic_regression_model")# Log a tag for this runmlflow.set_tag("model_type","LogisticRegression")mlflow.set_tag("data_source","Iris Dataset")print(f"MLflow Run ID: {mlflow.active_run().info.run_id}")
```

⚙️ 4. Logging Classic ML Models

```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score

X, y = load_iris(return_X_y=True)
model = LogisticRegression().fit(X, y)
preds = model.predict(X)

with mlflow.start_run():
    mlflow.log_metric("accuracy", accuracy_score(y, preds))
    mlflow.sklearn.log_model(model, "model")
```

🤗 5. Logging Hugging Face Transformer Models

When Training Your Own

```
from transformers import Trainer

trainer = Trainer(...)
trainer.train()

mlflow.pytorch.log_model(trainer.model, "hf_model")
mlflow.log_param("transformer_name", "bert-base-multilingual-cased")
```

When Using Pretrained Only

Run inference code

```
from transformers import pipeline

# Define model and task
model_name = "distilbert-base-uncased-finetuned-sst-2-english"

example_input = "MLflow is an amazing tool for experiment tracking!"
example_output = nlp_pipeline(example_input)
```

Log resulsts

```
# Start an MLflow run

with mlflow.start_run(run_name="Sentiment Analysis with Pre-trained Model after logging"):

    # Log model name as a parameter

    mlflow.log_param("hf_model_name", model_name)

    mlflow.log_param("hf_task", "sentiment-analysis")

    mlflow.log_param("example_input", example_input)

    mlflow.log_param("example_output", example_output[0]["label"])

    mlflow.log_param("example_output_score", example_output[0]["score"])

    # Log a tag

    mlflow.set_tag("usage_type", "pre-trained_inference")

    mlflow.set_tag("library", "HuggingFace Transformers")
```

🧱 6. Logging Preprocessing Code + Notes

a) Save preprocessing snippet from notebook

```
with open("cleaning_v3.py", "w") as f:
    f.write(cleaning_function_code)
mlflow.log_artifact("cleaning_v3.py")
```

b) Log preprocessing method comment

```
mlflow.set_tag("translation_type", "word-by-word")
mlflow.set_tag("note", "Used aggressive stopword filtering")
```

🧭 6. Understanding MLflow UI

Visit http://mlflow.daniam.am

Section	What You See
Experiments	List of experiments tracked
Runs	All executions (filter by name, tag, etc.)
Metrics	Plot training progress

Artifacts	Code, configs, logs, visualizations
Tags	Notes and context (e.g., model type)

🧪 Best Practices

- Always use a `with mlflow.start_run():` block
- Use descriptive run names and tags
- Log code versions and configs as artifacts
- Don't forget to `push` your code and docs to GitHub afterward
