**📘 MLflow and DagsHub**

**MLflow** is an open-source platform for managing the ML lifecycle, including experimentation, reproducibility, and deployment.

### **Key Components**

| **Component** | **Description** |
| --- | --- |
| **Tracking** | Logs parameters, metrics, artifacts, and models during experiments. |
| **Projects** | Packaging format for reproducible runs using MLproject file. |
| **Models** | Standard format for packaging ML models for diverse deployment tools. |
| **Registry** | Central hub for managing model lifecycle (staging, production, archiving). |

### **MLflow Tracking Basics**

import mlflow

with mlflow.start\_run():

mlflow.log\_param("learning\_rate", 0.01)

mlflow.log\_metric("rmse", 0.78)

mlflow.log\_artifact("model.pkl")

**Artifacts** can be model files, plots, or other outputs.

### **🔹MLflow UI**

* Launch via: mlflow ui
* Default at: http://localhost:5000

## **⚙️ MLflow Project Structure**

# MLproject

name: MyProject

conda\_env: conda.yaml

entry\_points:

main:

parameters:

alpha: {type: float, default: 0.5}

command: "python train.py --alpha {alpha}"

**🏷️ MLflow Model Registry Lifecycle**

1. **Register** a model version
2. Assign **Stages**: Staging, Production, Archived
3. Track model metadata and transitions

**🛰️ DagsHub**

**DagsHub** is a web-based platform built on top of Git, DVC, and MLflow for collaborative ML development and reproducible research.

### **🔹 Key Features**

| **Feature** | **Description** |
| --- | --- |
| **Git & DVC** | Version control for code and data via Git and DVC. |
| **MLflow UI** | Integrated MLflow tracking dashboard. |
| **Data Lineage** | Track changes and connections across code, models, and datasets. |
| **Experiment Comparison** | Visual diff of runs, metrics, parameters. |

**🔧 Basic Setup with DagsHub**

### **1. Initialize Git + DVC**

git init

dvc init

dvc remote add origin https://dagshub.com/<user>/<repo>.dvc

### **2. Add and Push Data**

dvc add data/train.csv

git add data/train.csv.dvc .gitignore

git commit -m "Add training data"

dvc push

### **3. Integrate MLflow**

* Set tracking URI:

mlflow.set\_tracking\_uri("https://dagshub.com/<user>/<repo>.mlflow")

* Set DagsHub credentials:

export MLFLOW\_TRACKING\_USERNAME=<your\_username>

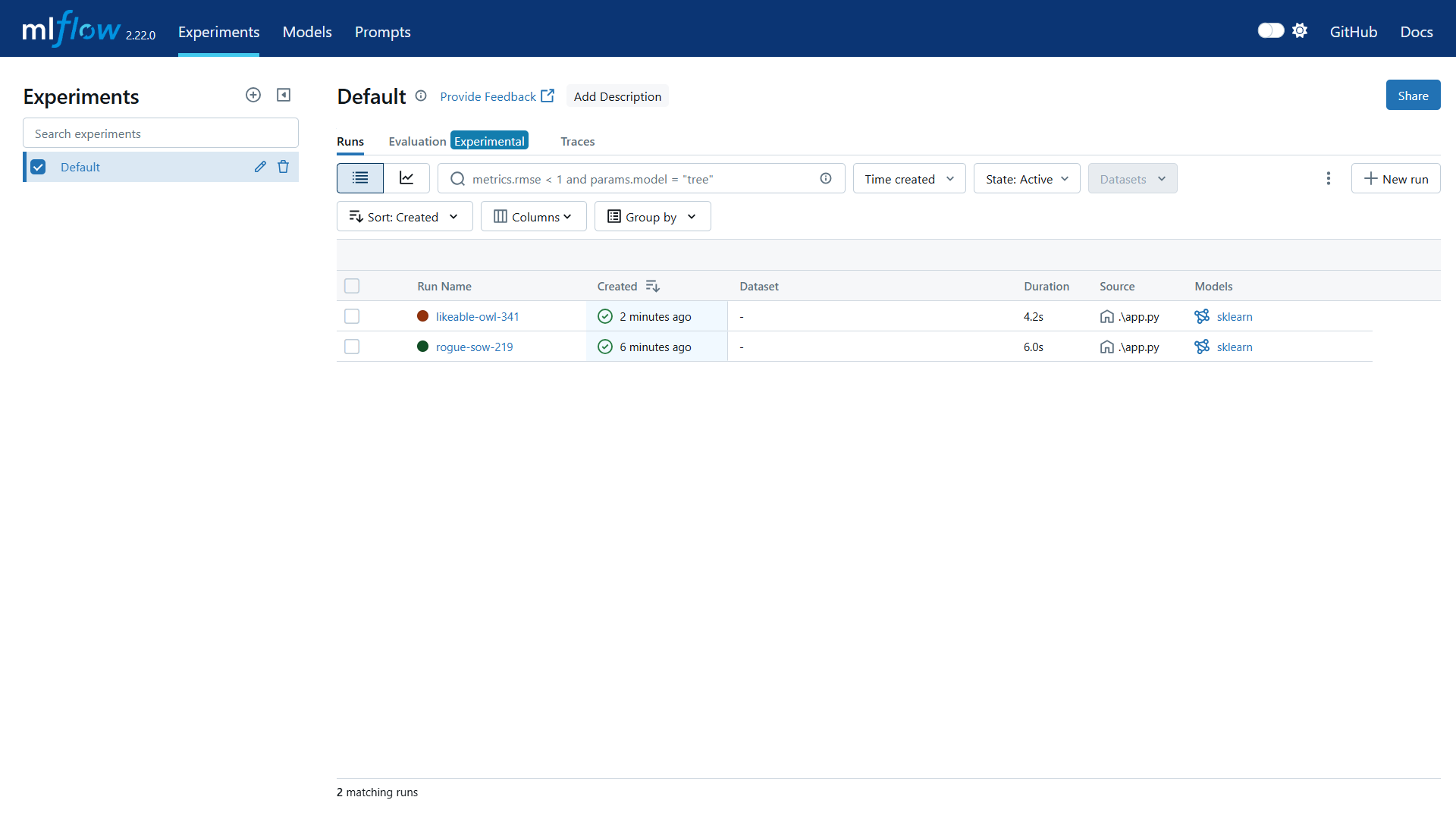
export MLFLOW\_TRACKING\_PASSWORD=<your\_token>

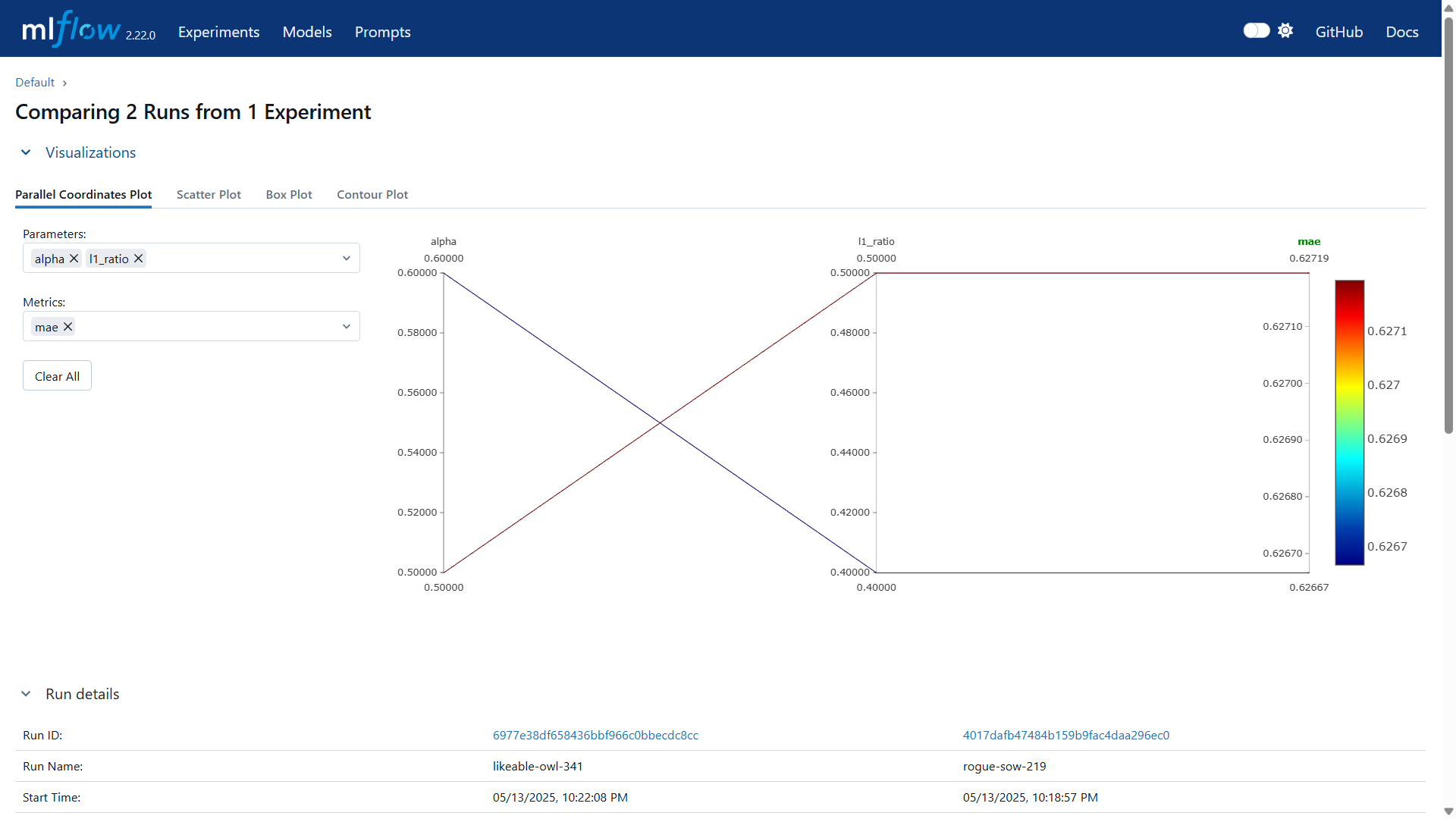
## **🔍 When to Use MLflow + DagsHub Together**

| **Scenario** | **Tool(s)** |
| --- | --- |
| Experiment tracking | **MLflow** |
| Model versioning & registry | **MLflow Registry** |
| Data and model version control | **DVC (via DagsHub)** |
| Team collaboration & review | **DagsHub UI** |

## **📌 Tips**

* Use **MLflow Autologging** for quick integration with libraries like sklearn, keras, etc.
* Use **DVC pipelines** to version and automate ML workflows.
* DagsHub is ideal for open-source and team projects needing version control + tracking.





**BentoML** is an open-source framework for building, packaging, and deploying machine learning models as scalable APIs or services.

**🚀 Why BentoML?**

| **Feature** | **Benefit** |
| --- | --- |
| Easy API serving | Turn ML models into REST/gRPC APIs in minutes |
| Model packaging | Version-controlled and reproducible model storage |
| Multiple framework support | Works with sklearn, xgboost, transformers, PyTorch, etc. |
| Deployment ready | Supports Docker, Kubernetes, AWS Lambda, and more |

## **📂 BentoML Workflow**

### **1. Model Training (anywhere)**

Train your model using your preferred framework.

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier().fit(X\_train, y\_train)

### **2. Save Model using BentoML**

import bentoml

bentoml.sklearn.save\_model("rf\_classifier", model)

✅ This saves the model along with metadata and dependencies.

### **3. Define a Service**

Create a file like service.py:

# service.py

import bentoml

from bentoml.io import NumpyNdarray

model\_ref = bentoml.sklearn.get("rf\_classifier:latest")

model\_runner = model\_ref.to\_runner()

svc = bentoml.Service("rf\_service", runners=[model\_runner])

@svc.api(input=NumpyNdarray(), output=NumpyNdarray())

async def predict(input\_data):

return await model\_runner.predict.async\_run(input\_data)

### **4. Run the Service Locally**

bentoml serve service:svc

Runs the service at http://localhost:3000

### **5. Build a Bento (package)**

bentoml build

Creates a .bento package in the bentos/ directory.

### **6. Containerize and Deploy**

* **Docker**

bentoml containerize rf\_service:latest

### **📂 DVC**

**DVC** is an open-source tool for **data versioning**, **experiment tracking**, and **reproducible ML pipelines**. It works alongside Git.

## **🔧 Why Use DVC?**

| **Feature** | **Purpose** |
| --- | --- |
| Version control for data | Like Git, but for large data files |
| Reproducible pipelines | Track data + code + metrics together |
| Remote storage | Store datasets/models in cloud/local remotes |
| Team collaboration | Share data without bloating the Git repo |

## **🚀 Core Workflow**

### **1. Initialize DVC in a Git repo**

git init

dvc init

### **2. Track a dataset or model**

dvc add data/train.csv

git add data/train.csv.dvc .gitignore

git commit -m "Add training data with DVC"

### **3. Push data to remote storage**

dvc remote add -d myremote <remote-url>

dvc push

Supports S3, GCS, Azure, SSH, etc.

### **4. Pull data**

dvc pull

Downloads data from remote storage (e.g., in CI or on a teammate’s machine).