The Hoang

Model Development in MLOps

Understand and Implement Production-Grade Machine Learning Operations

Where We Are?

Define Problem & Plan
1. Data Engineering
2. Model Engineering
3. Model Evaluation
4. Model Deployment
5. Operations
6. Monitoring & Maintenance
MLOps Lifecycle

Recap 1/

Stage 1: Business Understanding & Problem Definition

- Identify the business problem.
- Define success metrics (e.g., accuracy, precision, recall).
- Understand constraints (data availability, computational resources).

DONE

Recap 2/

Stage 2: Data Engineering & Feature Engineering

- **Data Collection**: Gather relevant data from various sources (databases, APIs, logs).
- **Data Preprocessing**: Handle missing values, outliers, and inconsistencies.
- **Feature Engineering**: Transform raw data into meaningful features.
- **Data Versioning**: Use tools like **DVC**, **LakeFS**, or **MLflow** for data versioning.

We Are Here

Recap 3/

Stage 3: Model Development & Experimentation

- **Select Model Architecture**: Choose between traditional ML models (e.g., Random Forest, XGBoost) or deep learning models (CNNs, RNNs).
- **Hyperparameter Tuning**: Optimize parameters using techniques like grid search, random search, or Bayesian optimization.
- **Logging Experiments**: Use **MLflow**, **Weights & Biases**, or **TensorBoard** to track experiments.
- **Code Versioning**: Store code in **GitHub**, **GitLab**, or **Bitbucket**.

Coming Soon

Recap 4/

Stage 4: Model Training & Evaluation

- **Train Model**: Use GPUs or TPUs for faster training.
- **Evaluate Performance**: Compute metrics like accuracy, RMSE, F1-score.
- **A/B Testing**: Compare different models on a validation set.
- **Bias & Fairness Testing**: Ensure the model does not introduce bias.

Coming Soon

Recap 5/

Stage 5: Model Packaging & Versioning

- **Convert Model**: Save models in formats like **ONNX**, **TF SavedModel**, or **MLflow Model Format**.
- **Version Control**: Use **MLflow Model Registry** or **DVC** for model versioning.
- **Containerization**: Package models using **Docker** for portability.

Coming Soon

Recap 6/

Stage 6: Model Deployment

- **Deploy as REST API**: Use **FastAPI**, **Flask**, or **TensorFlow Serving**.
- **Deploy to Cloud**: Use **AWS SageMaker**, **Azure ML**, or **Google Vertex AI**.
- **Deploy with Kubernetes**: Use **Kubeflow Serving** for scalable deployment.

Coming Soon

Recap 7/

Stage 7: Model Monitoring & Retraining

- **Monitor Performance**: Use **Prometheus**, **Grafana**, or **EvidentlyAI**.
- **Detect Data Drift**: Identify changes in data distribution.
- **Automate Retraining**: Set up pipelines for periodic model retraining.

Learning Objectives

- MLOps Model Development
- Reproducibility of Experiments
- Hands-On Practice with MLflow for Experiment Tracking
- Use Git for Code Versioning
- Ensure Data Versioning Using Mlflow / DVC
- Best Practices

Difficulties of machine learning – The “Why”

Complexity of the ML Lifecycle

The machine learning process involves multiple stages, from data preprocessing to model deployment and monitoring. Each stage requires careful management to ensure efficiency and reproducibility

Experiment Management

Data scientists often conduct numerous experiments with different parameters, datasets, and algorithms. Without a centralized system, tracking these experiments and their outcomes becomes cumbersome.

Reproducibility

Ensuring that experiments yield consistent results across different environments and runs is crucial. This involves managing code versions, parameters, and library dependencies.

Difficulties of machine learning – The “Why”

Deployment Consistency

With many ML libraries available, deploying models consistently can be challenging. MLflow standardizes this process, ensuring models are packaged and deployed reliably.

Model Management

As teams produce multiple models, managing their lifecycle—versioning, testing, and deployment—becomes complex without a centralized platform

Why Manage Experiments?

Managing experiments is crucial for several reasons:

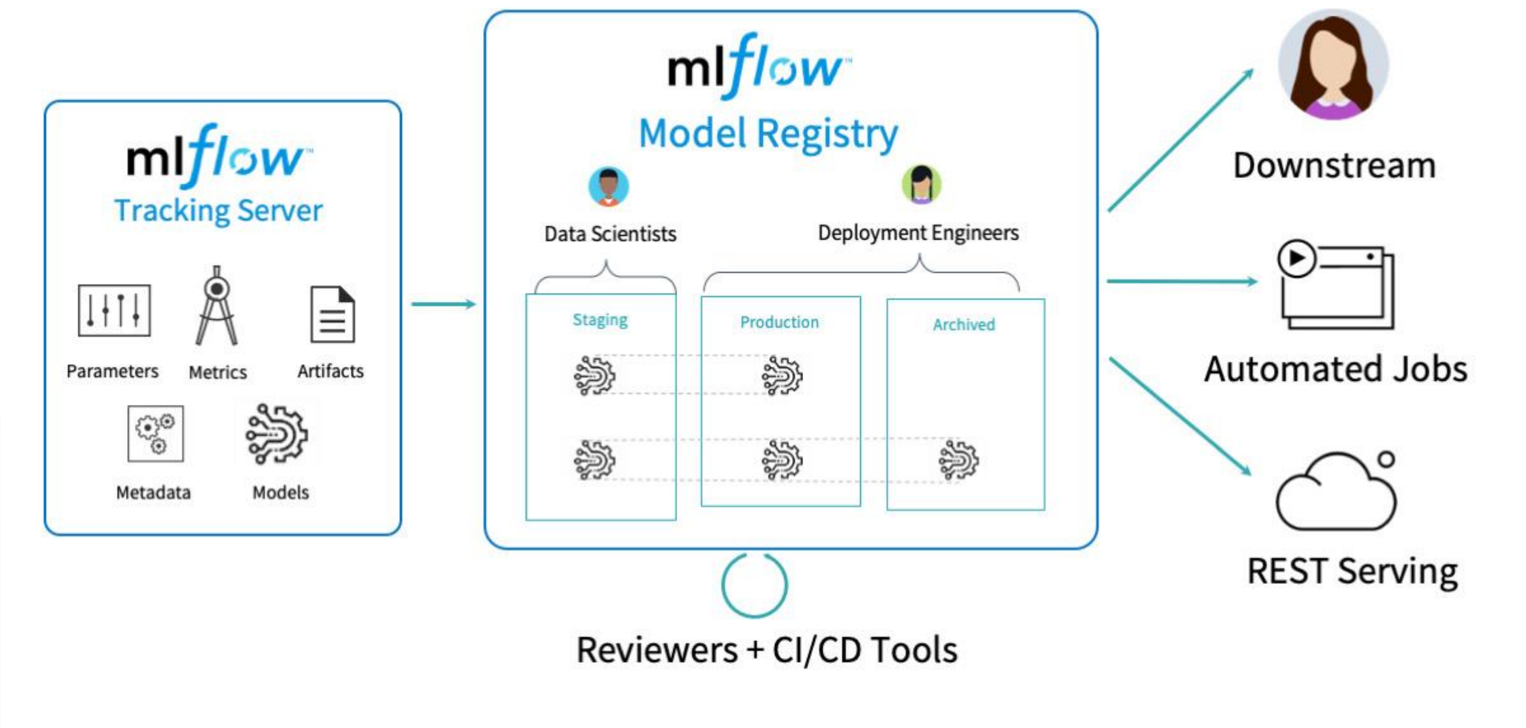
1. **Transparency and Comparison**: By logging experiments, data scientists can compare different models' performance and identify the best approach.
2. **Reproducibility**: Experiment management ensures that results are consistent across runs, which is vital for validating findings and scaling models.
3. **Collaboration**: Centralized experiment tracking facilitates collaboration among team members by providing a clear overview of all experiments and their outcomes.
4. **Efficiency**: Managing experiments reduces the time spent on trial-and-error approaches and helps in optimizing resources by focusing on promising models.

Problems That Cause Data Scientists to Need MLflow

- **Lack of Standardization**: Without a standardized tool, teams often work with different model versions stored in various locations, making it difficult to compare performance and manage models effectively.
- **Inefficient Collaboration**: Collaboration is hindered when team members cannot easily access or reproduce each other's work due to inconsistent environments or missing documentation.
- **Difficulty in Scaling**: As projects grow, managing and deploying models becomes increasingly complex without a scalable platform like MLflow.
- **Risk of Errors**: Manual tracking and deployment processes are prone to errors, which can lead to incorrect conclusions or model failures in production.

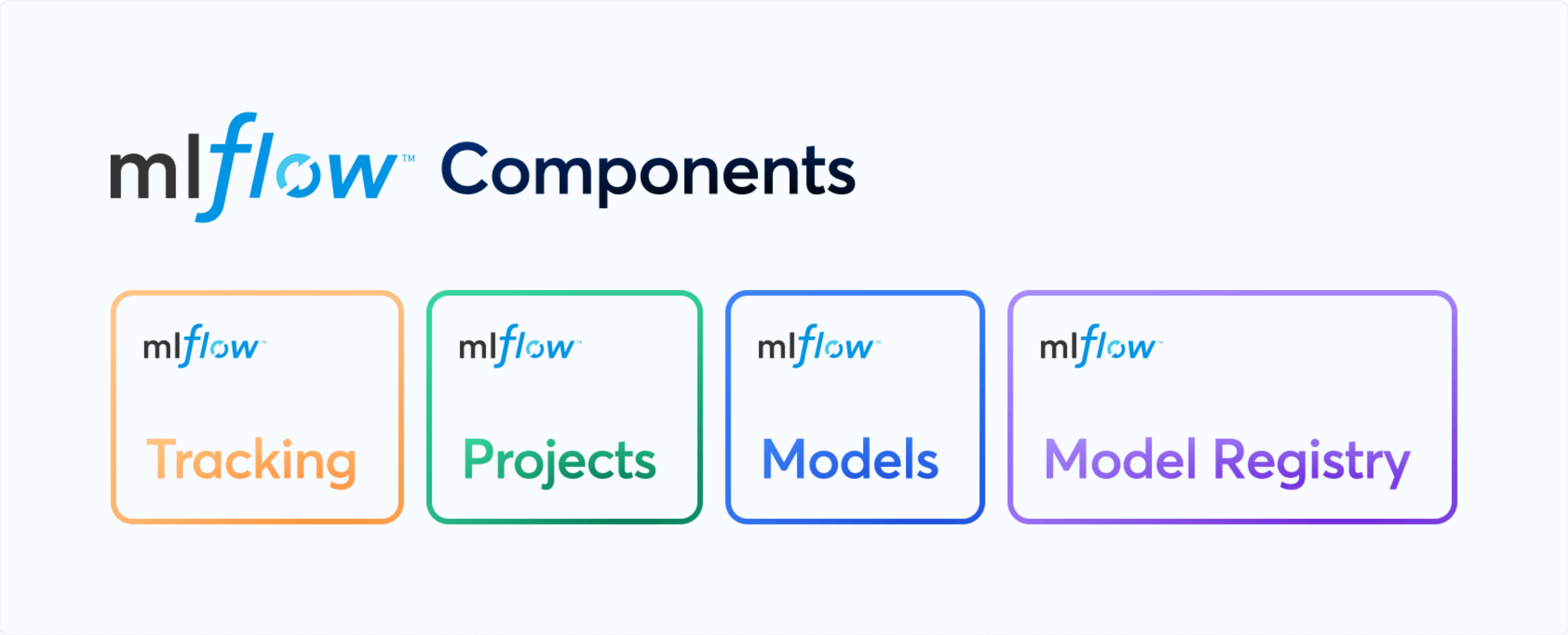
Introduction to MLflow

MLflow is an open-source platform created by Databricks to streamline the machine learning lifecycle. It provides tools for managing experiments, packaging code into reproducible runs, and deploying models in a consistent, scalable, and easy-to-monitor manner.



Core Components of MLflow

MLflow consists of several core components that work together to manage the ML workflow:



Core Components of Mlflow - Tracking

1. MLflow Tracking

- **Purposes**:
  - Logs and tracks machine learning experiments, including parameters, metrics, and artifacts.
  - Query data from experiment.
  - Store models, artifacts and code.
- **Features**: Allows data scientists to visualize and compare different runs to identify the best-performing model. It can be used locally or remotely via the MLflow server.
- **APIs**: Supports Java, Python, R, and REST APIs for recording and querying runs.

Core Components of Mlflow - Projects

2. MLflow Projects

- **Purpose**:
  - Packages machine learning code into a reusable form that can be easily reproduced and shared.
- **Features**: Each project is a directory with code or a Git repository, using a descriptor file to specify dependencies and execution methods. Projects can be chained into multi-step workflows.
- **Example**: A project might include a conda.yaml file for specifying a Python environment.

Core Components of Mlflow - Models

3. MLflow Models

- **Purpose**:
  - Standardize models for deployment.
  - Build customized models.
- **Features**: Models are saved as directories with a descriptor file listing supported flavors (e.g., TensorFlow, Python function). Supports deployment to platforms like AWS SageMaker and Azure ML.

Core Components of Mlflow - Model Registry

4. MLflow Model Registry

- **Purpose**:
  - Provides a centralized model store for managing the lifecycle of MLflow models (Store and version ML models)
  - Load and deploy ML models.
- **Features**: Offers model lineage, versioning, stage transitions (e.g., staging to production), and annotations. Facilitates collaboration and governance.

Mlflow - Additional Components & Features

5. MLflow Deployments for LLMs

- **Purpose**: Streamlines access to SaaS and OSS Large Language Models (LLMs) through standardized APIs.
- **Features**: Enhances security with authenticated access and provides a common API interface for prominent LLMs.

6. Evaluate

- **Purpose**: Facilitates in-depth model analysis and comparison.
- **Features**: Supports objective comparison of traditional ML algorithms and cutting-edge LLMs.

7. Prompt Engineering UI

- **Purpose**: Offers a dedicated environment for prompt experimentation, refinement, evaluation, testing, and deployment.
- **Features**: Ideal for working with LLMs and other models requiring precise input crafting.

Mlflow - Additional Components & Features

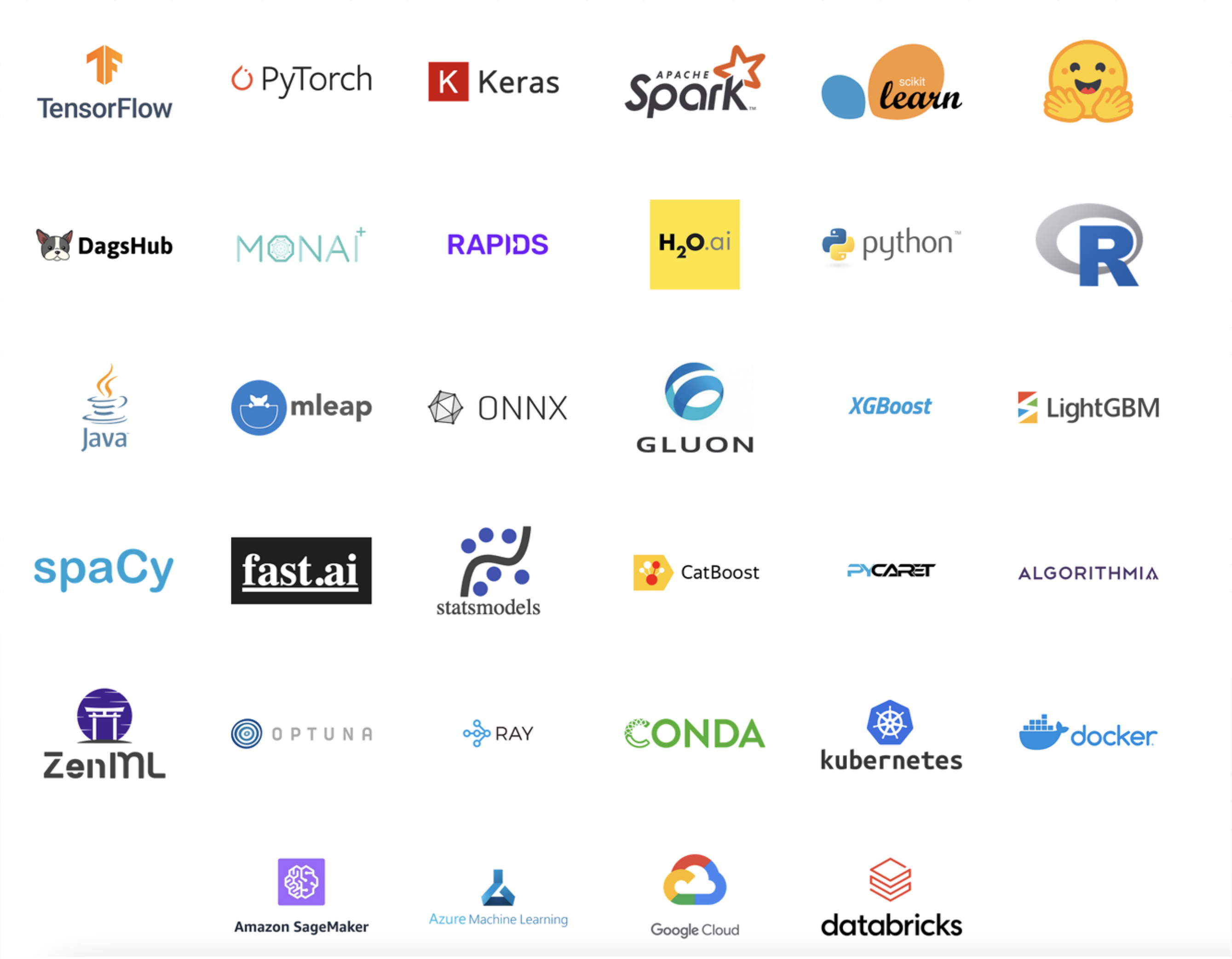
8. Recipes

- **Purpose**: Guides the structuring of ML projects for real-world deployment scenarios.
- **Features**: Provides recommendations to ensure functional end results.

Advanced Features and Use Cases

- **Nested Runs**: Organize experiments in a hierarchical structure for more structured experimentation.
- **Rich Metric Visualization**: Utilize advanced tools to analyze metrics over time and across different runs.
- **Automated ML Pipelines**: Use predefined templates for common tasks and dynamic step execution to adapt workflows based on data or previous results.
- **Multi-Model Endpoints**: Serve multiple models or versions from a single endpoint to optimize resource usage.
- **Plugin Ecosystem**: Extend MLflow's functionality with custom plugins for storage, authentication, etc..

Mlflow - Integrations

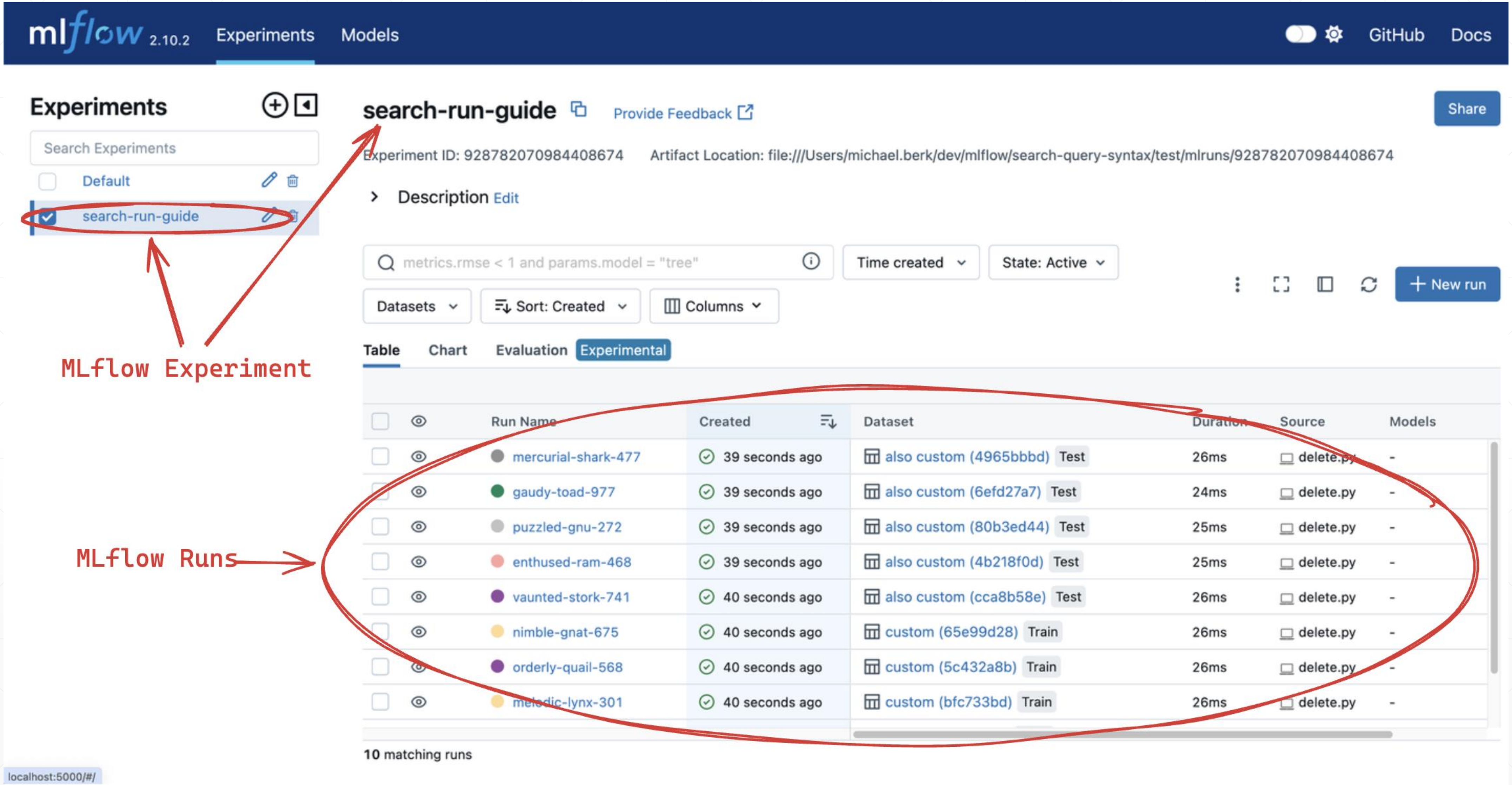


Practice: Setup Environment

Execute following commands (Linux required)

1. cd ~/dev
2. mkdir mlflow-practice
3. python –m venv venv
4. echo mlflow > requirements.txt
5. source venv/bin/activate
6. pip install –r requirements.txst
7. mkdir logs
8. mlflow ui >> logs/log_file.txt 2>&1 &

Mlflow Experiments



Working with Experiments

MLflow Client

- Create Experiments

```
client.create_experiment("Name")
```

- Tag Experiments

```
client.set_experiment_tag("Name",
k, v)
```

- Delete Experiments

```
client.delete_experiment("Name")
```

MLflow module

- Create Experiments

```
mlflow.create_experiment("Name")
```

- Tag Experiments

```
mlflow.set_experiment_tag(k, v)
```

- Delete Experiments

```
mlflow.delete_experiment("Name")
```

- Set Experiment

```
mlflow.set_experiment("Name")
```

Starting a New Experiment

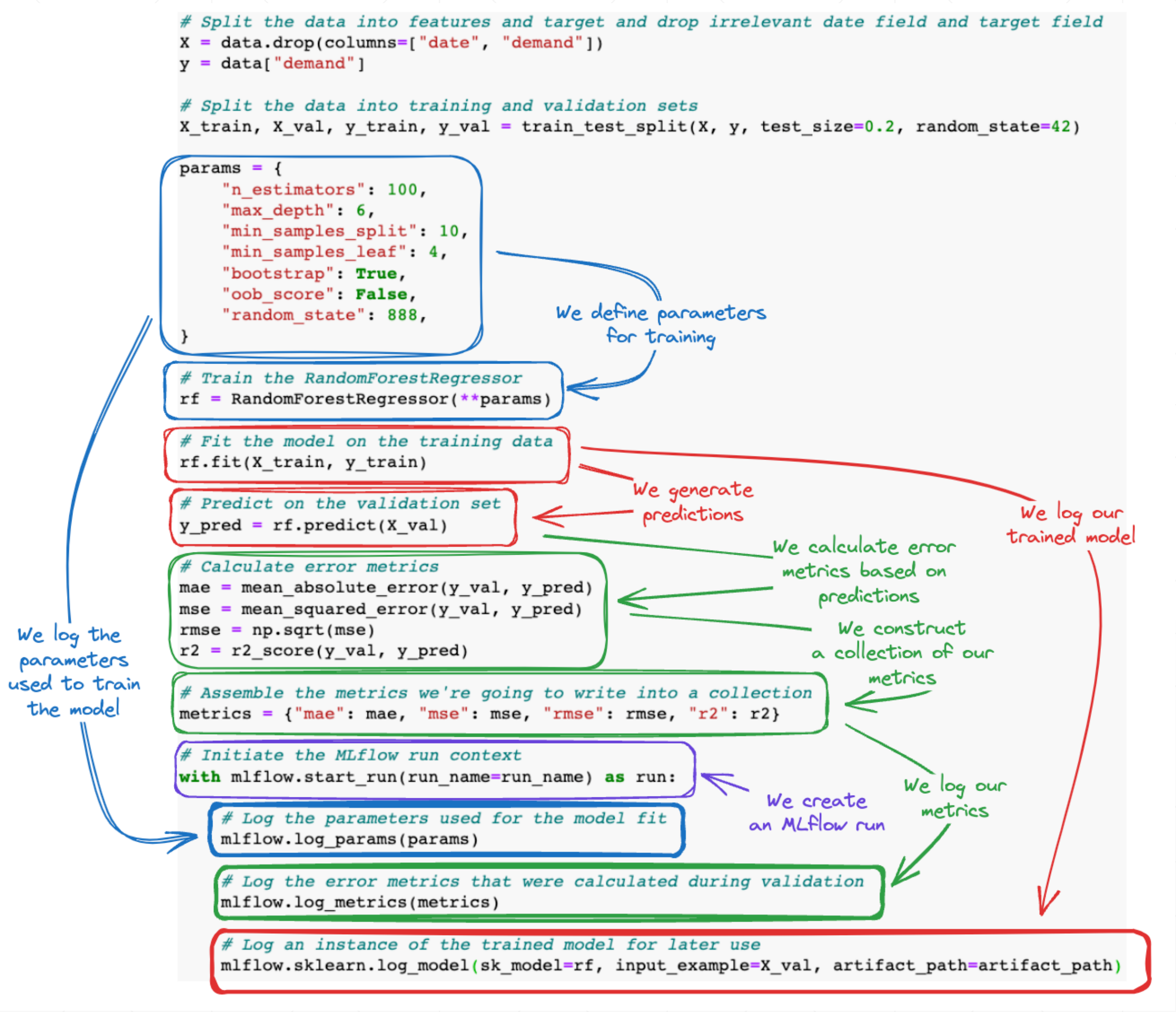
```python
import mlflow
# Create new Experiment
mlflow.create_experiment("My Experiment")
# Tag new experiment
mlflow.set_experiment_tag("scikit-learn", "lr")
# Set the experiment
mlflow.set_experiment("My Experiment")
```

Let’s Practice – MLflow experiments

Training runs

- How MLflow is organized
- New run equals new model training
- A run is placed within an experiment
- Invoked via mlflow.start_run()

Training runs



```python
# Split the data into features and target and drop irrelevant date field and target field
X = data.drop(columns=["date", "demand"])
y = data["demand"]

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

params = {
    "n_estimators": 100,
    "max_depth": 6,
    "min_samples_split": 10,
    "min_samples_leaf": 4,
    "bootstrap": True,
    "oob_score": False,
    "random_state": 888,
}

# Train the RandomForestRegressor
rf = RandomForestRegressor(**params)

# Fit the model on the training data
rf.fit(X_train, y_train)

# Predict on the validation set
y_pred = rf.predict(X_val)

# Calculate error metrics
mae = mean_absolute_error(y_val, y_pred)
mse = mean_squared_error(y_val, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_val, y_pred)

# Assemble the metrics we're going to write into a collection
metrics = {"mae": mae, "mse": mse, "rmse": rmse, "r2": r2}

# Initiate the MLflow run context
with mlflow.start_run(run_name=run_name) as run:
    # Log the parameters used for the model fit
    mlflow.log_params(params)

    # Log the error metrics that were calculated during validation
    mlflow.log_metrics(metrics)

    # Log an instance of the trained model for later use
    mlflow.sklearn.log_model(sk_model=rf, input_example=X_val, artifact_path=artifact_path)
```

Start a training run

```python
import mlflow

# Start a run
mlflow.start_run()
```

```
<ActiveRun: >
```

```python
# End a run
mlflow.end_run()
```

Setting a training run variable

```python
import mlflow
# Set experiment
mlflow.set_experiment("My Experiment")
# Start a run
run = mlflow.start_run()
# Print run info
run.info
```

```
<RunInfo: artifact_uri='./mlruns/0/9de5df4d19994546b03dce09aefb58af/artifacts',
  end_time=None, experiment_id='31', lifecycle_stage='active',
  run_id='9de5df4d19994546b03dce09aefb58af', run_name='big-owl-145',
  run_uuid='9de5df4d19994546b03dce09aefb58af', start_time=1676838126924,
  status='RUNNING', user_id='user'>
```

Logging to MLflow Tracking

- **Metrics**
  - `log_metric("accuracy", 0.90)`
  - `log_metrics({"accuracy": 0.90, "loss": 0.50})`
- **Parameters**
  - `log_param("n_jobs", 1)`
  - `log_params({"n_jobs": 1, "fit_intercept": False})`
- **Artifacts**
  - `log_artifact("file.py")`
  - `log_artifacts("./directory/")`

Logging a run

```python
import mlflow
# Set Experiment
mlflow.set_experiment("LR Experiment")

# Start a run
mlflow.start_run()

# Model Training Code here
lr = LogisticRegression(n_jobs=1)

# Model evaluation Code here
lr.fit(X, y)
score = lr.score(X, y)
```

```python
# Log a metric
mlflow.log_metric("score", score)

# Log a parameter
mlflow.log_param("n_jobs", 1)

# Log an artifact
mlflow.log_artifact("train_code.py")
```

Searching runs

```
mlflow.search_runs()
```

Searching runs – Filter searches

- `max_results` - maximum number of results to return.
- `order_by` - column(s) to sort in `ASC` ending or `DESC` ending order.
- `filter_string` - string based query.
- `experiment_names` - name(s) of experiments to query.

Searching runs – examples

```python
import mlflow
# Filter string
f1_score_filter = "metrics.f1_score > 0.60"
# Search runs
mlflow.search_runs(experiment_names=["Insurance Experiment"],
    filter_string=f1_score_filter,
    order_by=["metrics.precision_score DESC"])
```

Let’s Practice – MLflow tracking runs

thehn/getting_started_mlflow