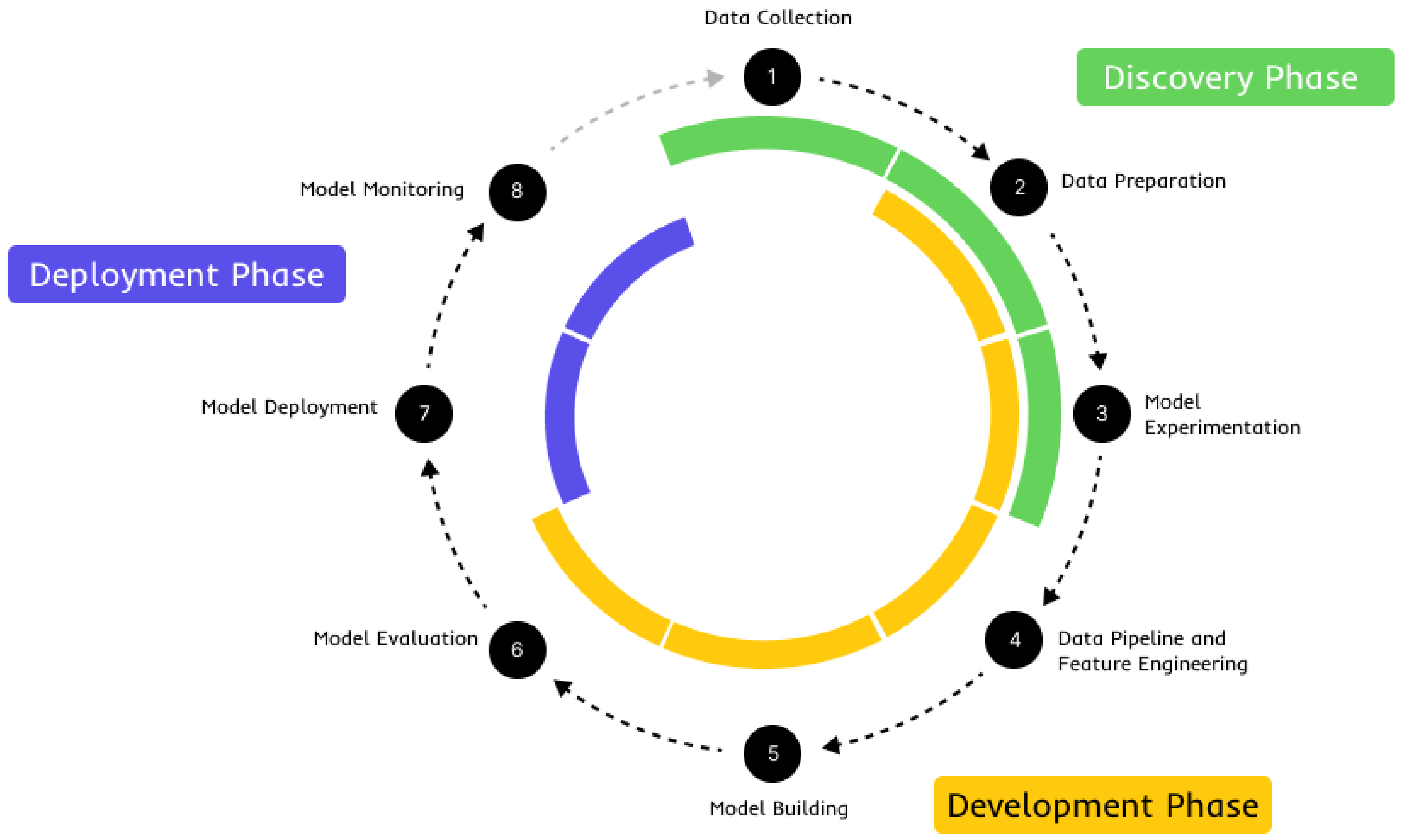
Profiling
Logging
Modelling
Organization
Continues X
Monitor
Deploy
Version Control
Reproduceability
Distruted apps
Package
Cloud
</>

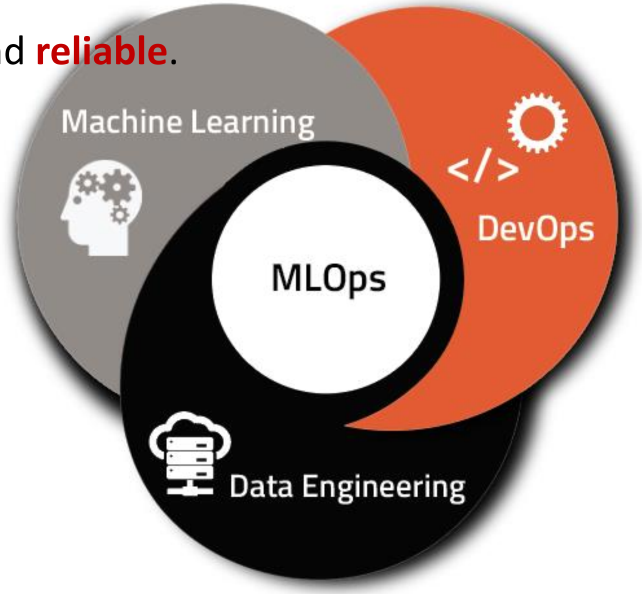
Eman Raslan

Machine Learning Lifecycle



What is MLOps?

- MLOps is a **set** of **practices** and **processes** that enable organizations to **effectively manage** the **development**, **deployment**, and **maintenance** of **machine learning models**.
- MLOps is like a **cookbook** that helps you make sure this **entire process** is done **consistently, efficiently,** and **effectively.**
- **MLOps is about** **automating** the entire machine learning **lifecycle** from **data collection**, **data preparation**, **model experimentation**, **training** and **evaluation**, **deployment** and **monitoring**.
- **By automating these processes, organizations can reduce the time and effort required to develop and deploy models**, while also ensuring that models are **accurate** and **reliable**.



Why do we need it?

- **Scalability**
  - **MLOps supports the standardization** of environments, optimization of resource allocation, effective management of infrastructure, seamless integration of processes, and implementation of Continuous Integration and Continuous Deployment (CI/CD), **making it easier to scale machine learning models as demand increases.**
- **Reproducibility**
  - **MLOps ensures consistency and repeatability in the training and deployment of machine learning models.** It helps to track the environment and version of each model as well as the data used for training and evaluation. **This helps to ensure that a model is reproducible and reliable.**
  - Fast-time-to-market: **MLOps simplifies and accelerates the end-to-end machine learning model development and deployment process.** Automated workflows help to speed up the process of model building, testing and deployment. **This helps to quickly launch new products and services with improved accuracy and better performance.**
- **Monitoring**
  - **MLOps provides real-time monitoring of machine learning model performance** and alerts if any discrepancies in accuracy or performance are detected. This helps to identify potential issues that could be affecting the model performance and take appropriate corrective actions.

Important Concepts in MLOps

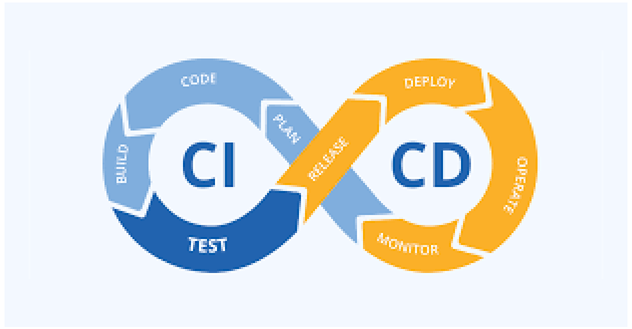
- **Data Lineage**
  - It is the ability to trace a piece of data back to its original source. **It allows you to track exactly how data moved from source to target and who made each change along the way.** The goal is to understand where your data comes from so that you can ensure it's trustworthy and accurate.
- **Data Provenance**
  - It is a concept that's similar to data lineage, but **it focuses more on how data is created and altered.** The goal is to understand the history of a piece of information so that you can determine why it changed over time and what happened as a result.
- **Metadata**
  - **It is data about your data.** It includes information like the location, type and size of a piece of information as well as who owns it and how it's being used. In addition to helping you understand what data you have, metadata can help you better manage your system.

How does applying these concepts help your system?

- It helps ensure that models are built upon quality data, preserving the accuracy of the results.
- It helps with debugging and reproducing models, as it can be used to identify where issues occurred and how to reproduce results for further analysis.
- Data provenance and data lineage enable compliance and governance, improved transparency, and accountability.
- Help with reducing the time needed for data preparation and model deployment, as it provides insight into the data used to develop models.

Continuous Integration/Continuous Delivery (CI/CD)

- The concepts of DevOps have direct application to MLOps.
- In software development, different developers do not work on the same code simultaneously. Instead, they check out code they're going to work on from a code repository and then merge it back after their task is completed. Before returning the code, the developer verifies if anything in the main version has changed and runs unit tests on their updates before merging with the main code.
- Continuous Integration (CI) and Continuous Delivery (CD) are well-established practices in software development to automate code integration, testing, and deployment. In **MLOps**, these concepts are adapted to the unique challenges of machine learning projects, where not only code but also **data, models**, and **infrastructure** must be integrated and deployed seamlessly.



Key Concepts of CI/CD in MLOps

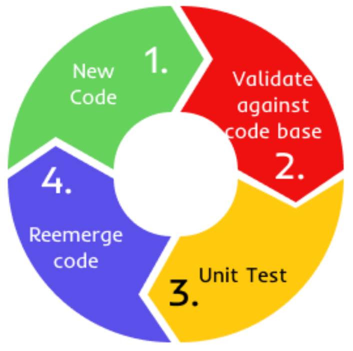
- **1. Continuous Integration (CI):**
  - **Automated Testing of Code and Models**: CI in MLOps ensures that new code, data preprocessing steps, and model changes are automatically tested whenever changes are made to the codebase. It validates the integration of all components of the ML pipeline.
  - **Version Control**: All code, data, and model updates are version-controlled using tools like Git. This helps in tracking which code changes result in new models and how they perform.
  - **Automated Data Validation**: Since data can change over time, CI also ensures that changes in data (e.g., new training data) are automatically validated to prevent data drift or broken pipelines.

Key Concepts of CI/CD in MLOps

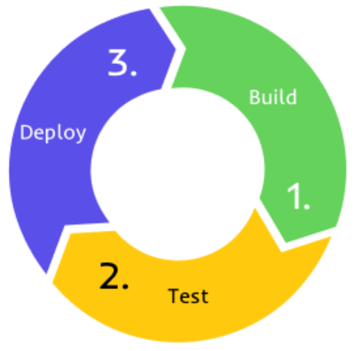
- **2. Continuous Delivery (CD):**
    - **Automated Model Deployment**: CD automates the process of delivering new models to staging or production environments. Models that pass the CI tests are automatically packaged and made ready for deployment.
    - **Infrastructure as Code**: In MLOps, infrastructure (e.g., cloud resources, pipelines, and deployment environments) is treated as code. Automated tools deploy the infrastructure required to serve the ML model.
    - **Canary Releases and Rollbacks**: New models are often deployed gradually (e.g., to a small subset of users in a canary release) to monitor their performance in production. If issues arise, the system can automatically roll back to a previous version.

Continuous Training (CT)

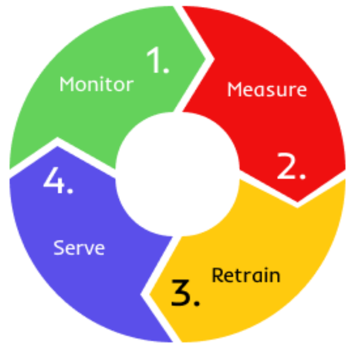
- In MLOps, continuous integration of source code, testing (unit and integration), and continuous delivery to production are crucial.
- But there is another important aspect to MLOps: **data**.
- **ML models can deviate from their intended behavior when the data profile changes**, unlike conventional software that consistently produces the same results. To address this, MLOps introduces the concept of Continuous Training (CT) in addition to continuous integration and continuous delivery.
- **Continuous Training ensures that the ML model remains effective even as the data profile evolves.**



Continuous Integration (CI)

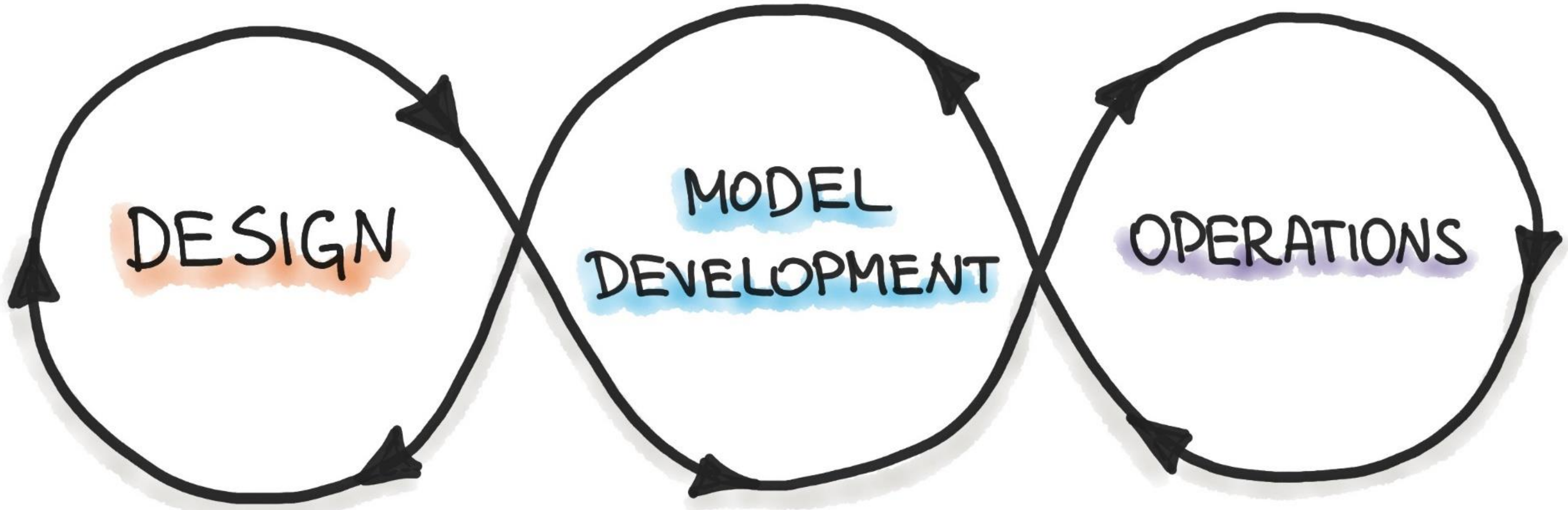


Continuous Delivery (CD)



Continuous Training (CT)

MLOps

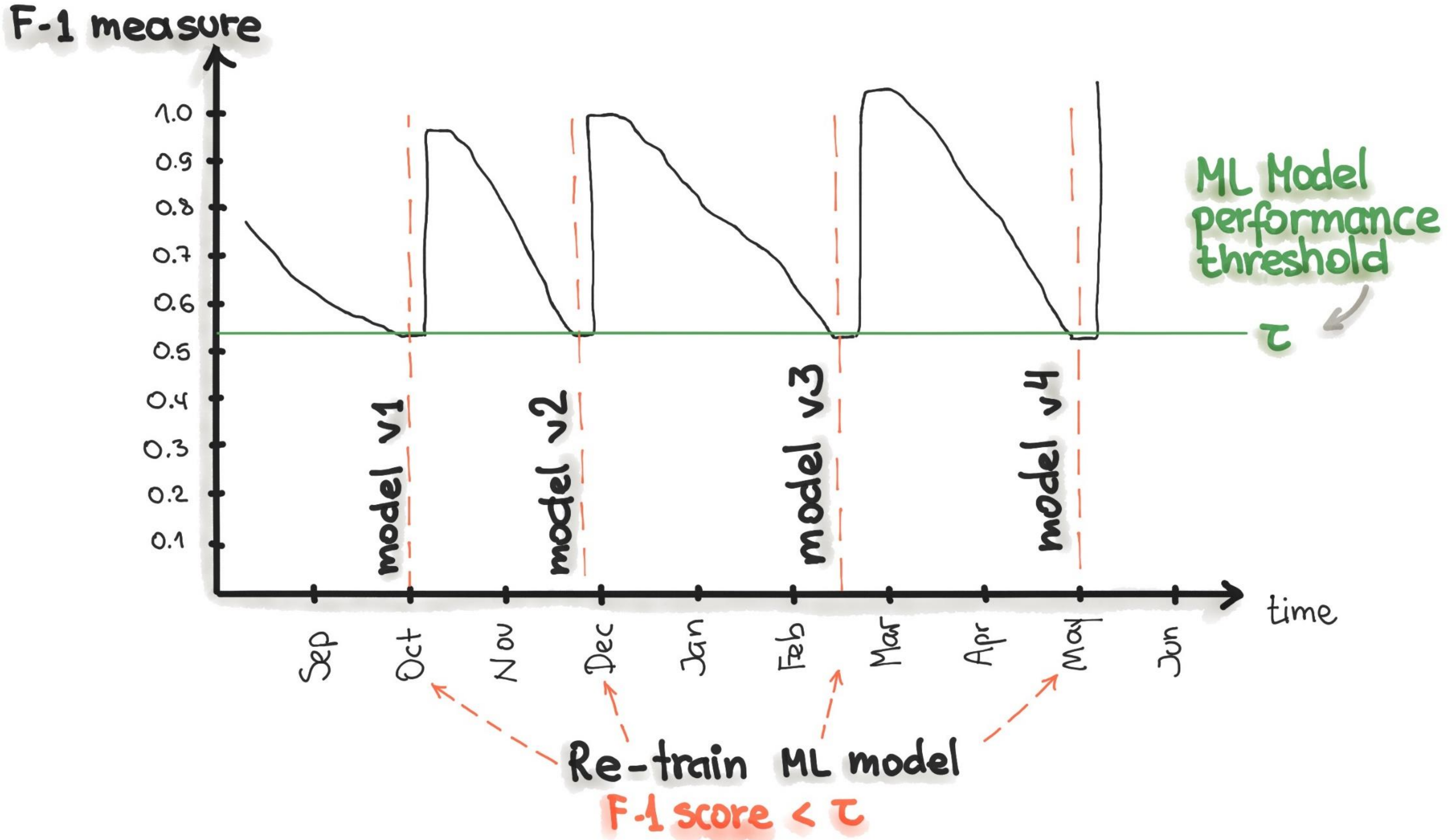


- Requirements Engineering
- ML Use-Cases Priorization
- Data Availability Check

- Data Engineering
- ML Model Engineering
- Model Testing & Validation

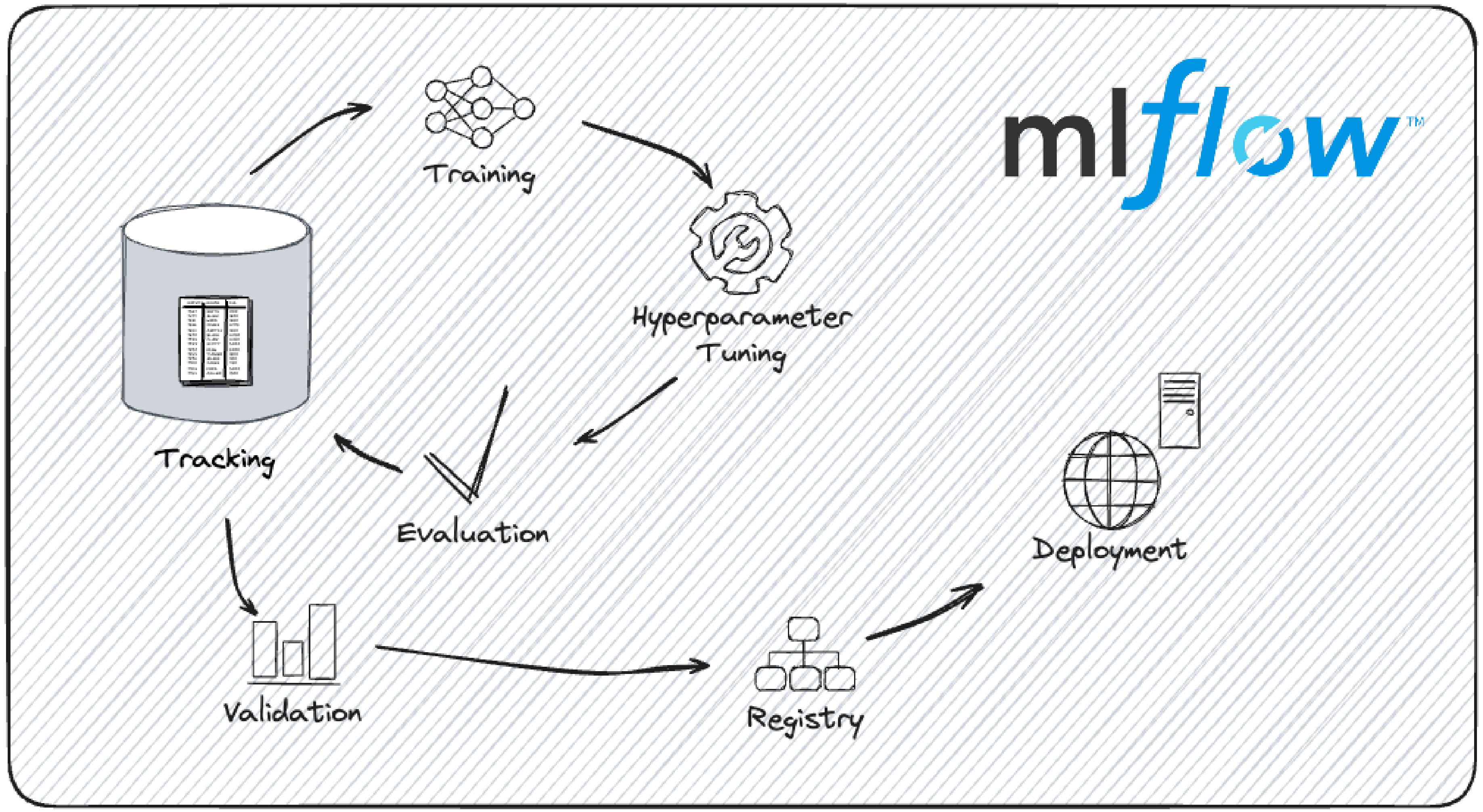
- ML Model Deployment
- CI/CD Pipelines
- Monitoring & Triggering

ML MODEL DECAY MONITORING



What is MLflow?

Managing the Machine Learning Lifecycle



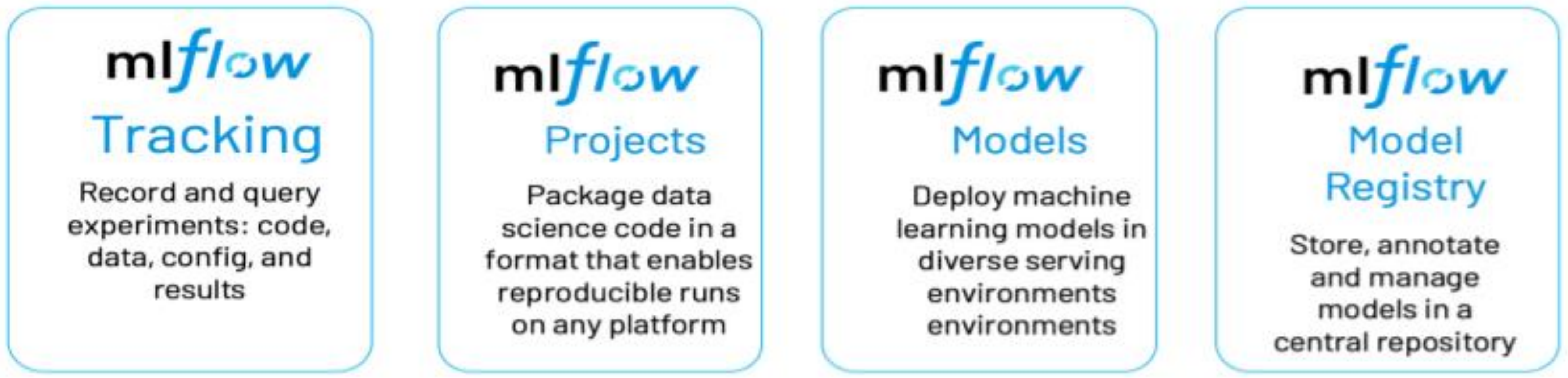
Why do we need MLflow in the ML lifecycle?

- Managing machine learning (ML) projects is challenging because of the various stages involved, from development to deployment. Without proper tools, teams often face several key issues:
- **1. Complexity in Managing ML Experiments:**
  - **Tracking Experiments**: When experimenting with different hyperparameters, models, and datasets, keeping track of which combination yields the best performance is difficult. Manual tracking can lead to confusion, errors, and duplication of efforts.
  - **Versioning Chaos**: ML experiments often involve various versions of data, code, and models. Without version control, it's easy to lose track of changes, making it harder to reproduce successful results later.
- **2. Need for Reproducibility:**
  - **Reproducing Results**: In ML, reproducing results is critical, especially when moving from research to production or sharing findings with other teams. Without proper logging of every experiment's parameters, metrics, and artifacts (e.g., models, plots), it can be impossible to retrace steps and replicate experiments.
  - **Consistent Environments**: Machine learning projects involve dependencies and configurations that can vary across environments (e.g., local, cloud, or containerized). Ensuring that code runs the same way in different environments is essential but hard to guarantee without standardized project structures.
- **3. Challenges in Model Deployment:**
  - **Moving Models to Production**: After training, deploying models to production requires handling dependencies, packaging models properly, and ensuring that the model works consistently across platforms. Without an integrated solution, this process can be time-consuming and prone to errors.
  - **Model Lifecycle Management**: Once deployed, models need monitoring, versioning, and possible updates. Managing different versions (e.g., staging, production) and transitioning between them is complex without a systematic framework.

Why do we need MLflow in the ML lifecycle?

- **Experiment Tracking**: MLflow provides tools to automatically log parameters, metrics, and artifacts, allowing for easy tracking of experiments.
- **Reproducibility**: MLflow Projects helps package code with environment specifications, ensuring reproducibility across different systems.
- **Model Deployment**: MLflow Models and Model Registry make it easier to package models and deploy them on various platforms while managing their lifecycle (versioning, transitions between staging and production).

Core components of MLflow



mlflow
Tracking
Record and query experiments: code, data, config, and results

mlflow
Projects
Package data science code in a format that enables reproducible runs on any platform

mlflow
Models
Deploy machine learning models in diverse serving environments environments

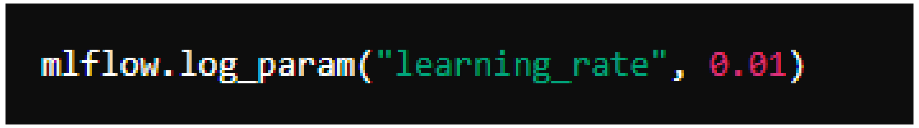
mlflow
Model Registry
Store, annotate and manage models in a central repository

What is MLflow Tracking?

- MLflow Tracking is a tool that helps you **log and record key aspects of machine learning experiments** such as hyperparameters, metrics, and artifacts.
- This allows for systematic **experiment management**, making it easier to compare results, reproduce experiments, and share findings.

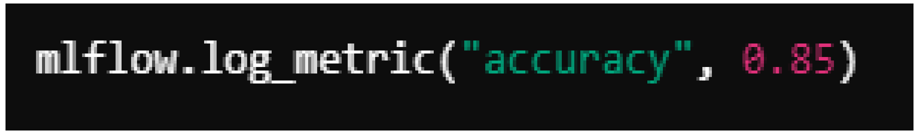
How MLflow Tracking Works?

- **Parameters**: Configuration values like hyperparameters (e.g., learning rate, batch size) that are used in training the model.



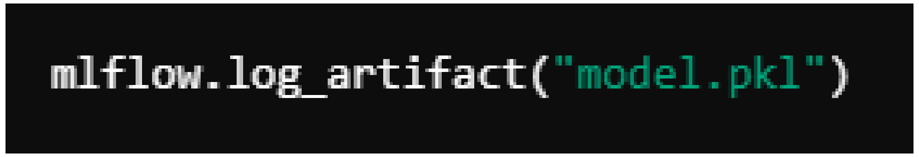
```
mlflow.log_param("learning_rate", 0.01)
```

- **Metrics**: Quantitative measurements such as accuracy, loss, precision, etc., used to evaluate the performance of the model.



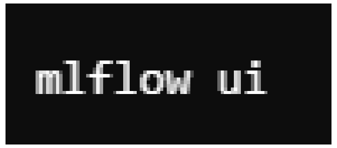
```
mlflow.log_metric("accuracy", 0.85)
```

- **Artifacts**: Files generated during the training process such as models, datasets, and plots.



```
mlflow.log_artifact("model.pkl")
```

- These components are logged during a run, which can later be visualized and compared using the **MLflow UI**.



```
mlflow ui
```

Demo

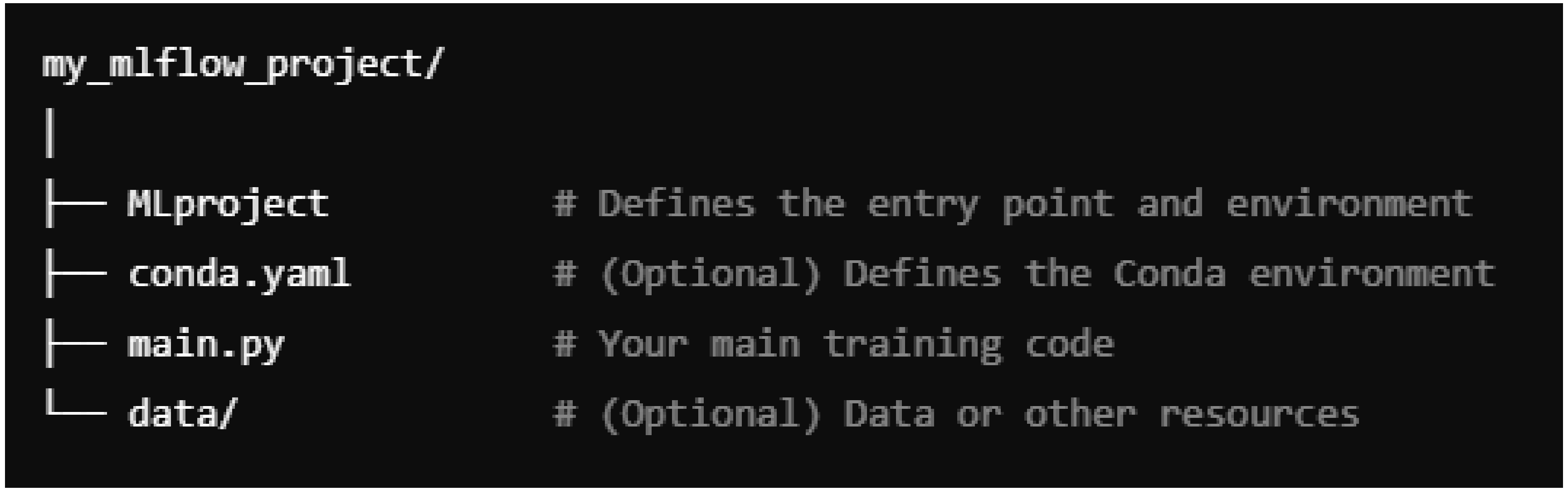
MLflow Projects

- An **MLflow Project** is a standardized format for packaging and sharing machine learning code. Each project includes all the dependencies and configurations required to run your code, making it easy to reproduce and share your work.
- A project can be as simple as a directory with your code and a descriptor file called an **MLproject file**. It also supports complex workflows, including dependencies managed with **Conda** or Docker, making it highly flexible.

Why is it Essential for Reproducibility?

- **Reproducibility**: MLflow Projects help you **capture the environment** in which your experiments were run. This makes it possible for others (or you) to reproduce your results consistently, even on different machines or cloud environments.
- **Portability**: Projects are self-contained, meaning they package code, environment settings, and dependencies, allowing for easy collaboration and model deployment.
- **Consistency Across Platforms**: Since MLflow Projects define the environment explicitly (through Conda or Docker), it ensures consistency when running experiments on different platforms.

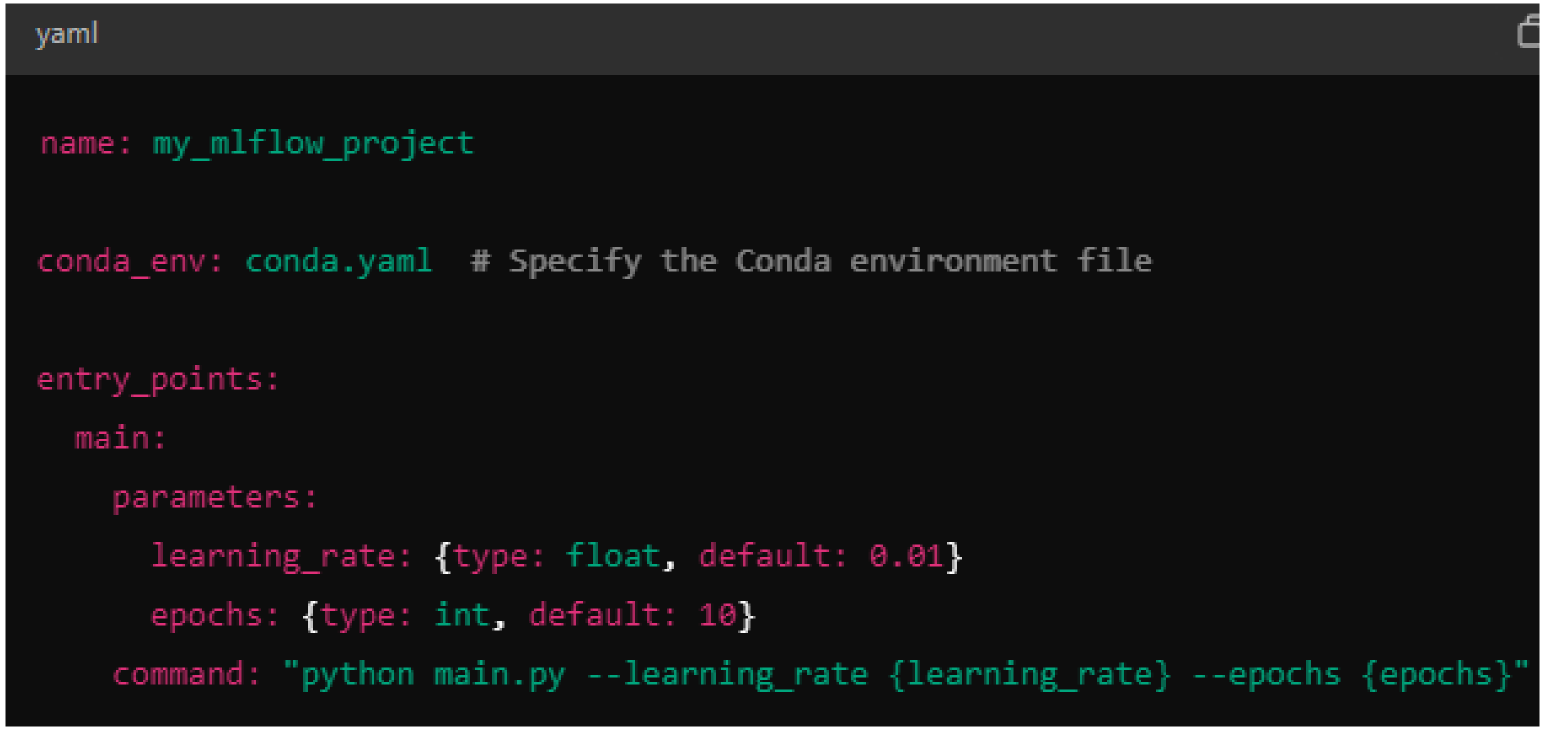
Structure of an MLflow Project Folder



```
my_mlflow_project/
│
├── MLproject          # Defines the entry point and environment
├── conda.yaml         # (Optional) Defines the Conda environment
├── main.py            # Your main training code
└── data/              # (Optional) Data or other resources
```

MLproject File Example

The **MLproject** file is a **YAML** file that defines the project structure, including how to run the code and what dependencies it requires.



yaml

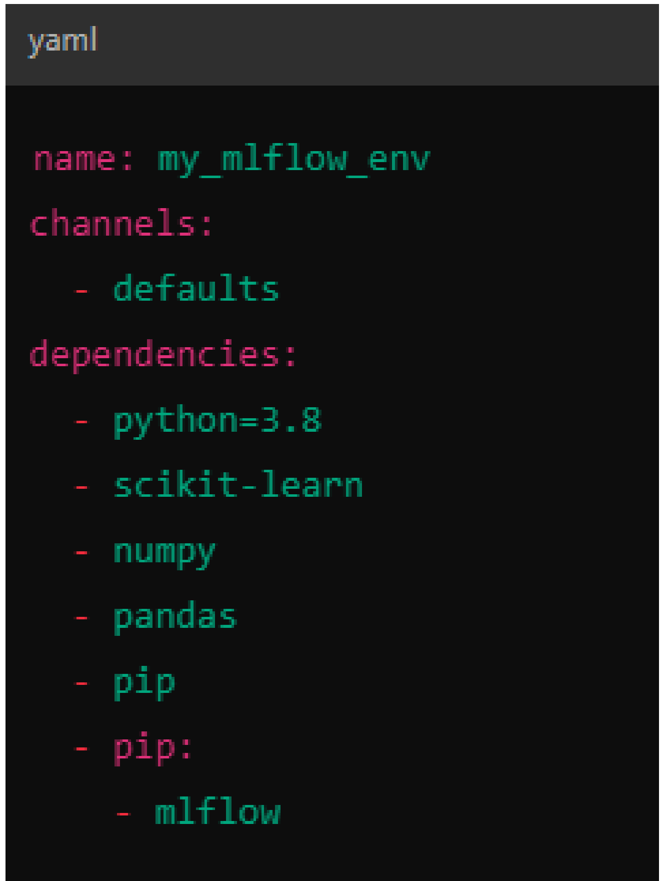
```yaml
name: my_mlflow_project

conda_env: conda.yaml  # Specify the Conda environment file

entry_points:
  main:
    parameters:
      learning_rate: {type: float, default: 0.01}
      epochs: {type: int, default: 10}
    command: "python main.py --learning_rate {learning_rate} --epochs {epochs}"
```

conda.yaml File Example

The **conda.yaml** file defines the Conda environment for your project, specifying the Python version and any libraries your project needs.



```yaml
name: my_mlflow_env
channels:
  - defaults
dependencies:
  - python=3.8
  - scikit-learn
  - numpy
  - pandas
  - pip
  - pip:
    - mlflow
```

Demo

- **Goal**: Demonstrate how to clone and run an MLflow project hosted on GitHub: https://github.com/mlflow/mlflow-example
  1. git clone https://github.com/mlflow/mlflow-example.git
  2. cd mlflow-example
  3. mlflow run . -P learning_rate=0.01 -P epochs=10
  4. mlflow ui

MLflow Models

- MLflow Models is a component of MLflow that provides a way to **log**, **package**, and **deploy** machine learning models.
- It abstracts the complexities of model management, allowing you to focus on building models without worrying about deployment details.

Model Logging and Packaging

- **Logging Models**: MLflow allows you to log models using simple commands. You can log models trained with popular libraries like TensorFlow, PyTorch, Scikit-learn, and more.
- **Model Packaging**: Once logged, models are saved in a standard format that includes all the necessary components (model artifacts, conda environments, etc.) to make them portable and easy to deploy.

Deployment Options

- MLflow supports various deployment options, including:
  - **Local deployment**: Run models on your local machine.
  - **REST API**: Serve models as REST endpoints for easy integration with applications.
  - **Cloud and container deployment**: Deploy models to cloud services or containerized environments (e.g., Docker).
  - **MLflow Model Registry**: Manage and version your models in a centralized repository.

Demo

MLflow Model Registry

- The **MLflow Model Registry** is a system for **managing the lifecycle** of machine learning models. It allows you to:
  - **Version models**: Keep track of multiple versions of the same model as you iterate over your experiments.
  - **Document models**: Add notes and metadata to models, helping teams understand the history and purpose of different models.

Why is Versioning, Staging, and Production Important?

- **Versioning**: Keep track of different iterations of your model. This is crucial for debugging, auditing, and comparing models.
- **Production**: Ensures that the best-performing, validated models are used in real-world applications.
- **Archived**: Retains older models for auditing or rollback in case newer models underperform.

Create a New Experiment

```
mlflow.set_experiment("Iris_Classification_Experiment")
```

MLflow UI

- The **MLflow UI** is a web-based interface that allows you to interact with all the experiments you’ve logged in MLflow.

What You Can Visualize in the MLflow UI?

- **Experiments**:
  - You can see a list of all experiments, where each experiment contains multiple **runs**.
  - Each run tracks model versions, parameters, metrics, and artifacts.
- **Metrics**:
  - Visualize metrics such as accuracy, precision, recall, loss, etc.
  - MLflow allows you to compare metrics across multiple runs within the same experiment.
- **Parameters**:
  - View logged hyperparameters like learning rate, number of iterations, or batch size.
  - These are logged during training and allow for easy comparison across different runs.
- **Models**:
  - Models that were logged during an MLflow run are available as **artifacts**.
  - The MLflow UI makes it easy to download or use these models for future use, and provides details such as the model's framework (e.g., Scikit-learn, PyTorch) and dependencies.

Resources

- https://arturlunardi.com/mlops-intro-why-do-we-need-it/