Pipeline composition via MLFlow: Connecting the dots

Tommaso Tedeschi
tommaso.tedeschi@pg.infn.it

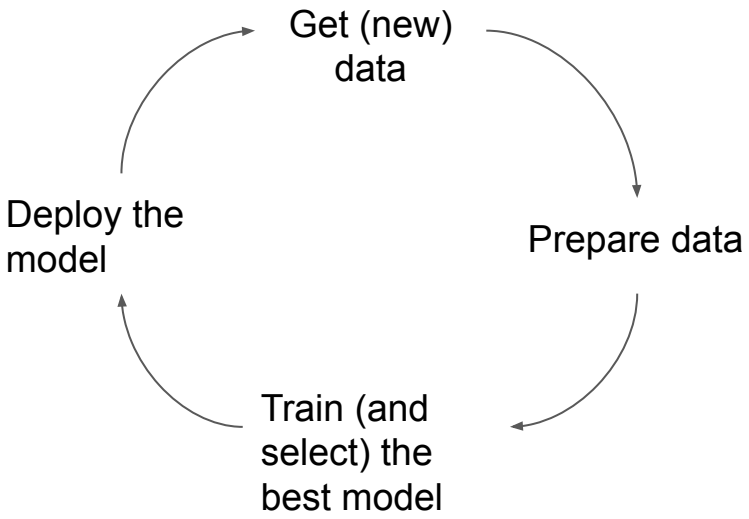
ICSC
Centro Nazionale di Ricerca in HPC,
Big Data and Quantum Computing

INFN
PERUGIA
Istituto Nazionale di Fisica Nucleare
Sezione di Perugia

Building a ML pipeline

Building machine learning pipeline is hard:

- 100s of software tools to leverage
- Hard to track and reproduce results: code, data, params, etc
- Hard to share models
- Hard to productionize models
- Needs large scale for best results



Building a ML pipeline

Building machine learning pipeline is hard:

- 100s of software tools to leverage
- Hard to track and reproduce results: code, data, params, etc
- Hard to share models
- Hard to productionize models
- Needs large scale for best results

In most cases, you end up like this!

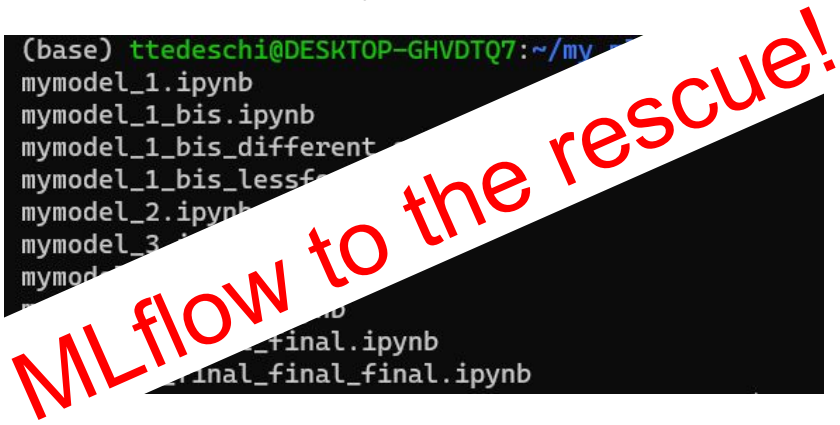
```
(base) ttedeschi@DESKTOP-GHVDTQ7:~/my_mlproject$ ls
mymodel_1.ipynb
mymodel_1_bis.ipynb
mymodel_1_bis_different_splitting.ipynb
mymodel_1_bis_lessfeatures.ipynb
mymodel_2.ipynb
mymodel_3.ipynb
mymodel_3_best.ipynb
mymodel_3_final.ipynb
mymodel_3_final_final.ipynb
mymodel_3_final_final_final.ipynb
```

Building a ML pipeline

Building machine learning pipeline is hard:

- 100s of software tools to leverage
- Hard to track and reproduce results: code, data, params, etc
- Hard to share models
- Hard to productionize models
- Needs large scale for best results

In most cases, you end up like this!



What is MLFlow?

MLflow is a versatile, expandable, open-source platform for managing workflows and artifacts across the **machine learning lifecycle (**developed by Databricks):

- “API-first” framework, built around REST APIs and CLI:
  - Allow submitting runs, models, etc from any library & language
  - Example: a model can just be a lambda function that MLflow can deploy in many places (Docker, Azure, etc…)
- Modular design: distinct components:
  - Let people use different components individually
  - Easy to integrate into existing ML platforms and workflows
- Cross-cloud
- Open and extensible
- Platform agnostic for maximum flexibility

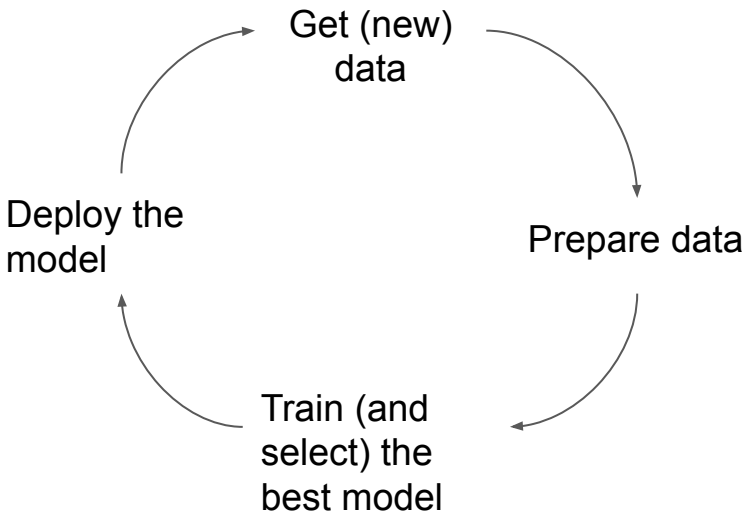
It has built-in integrations with many popular ML libraries, but can be used with any library, algorithm, or deployment tool. It is designed to be extensible, so you can write plugins to support new workflows, libraries, and tools.

https://mlflow.org/

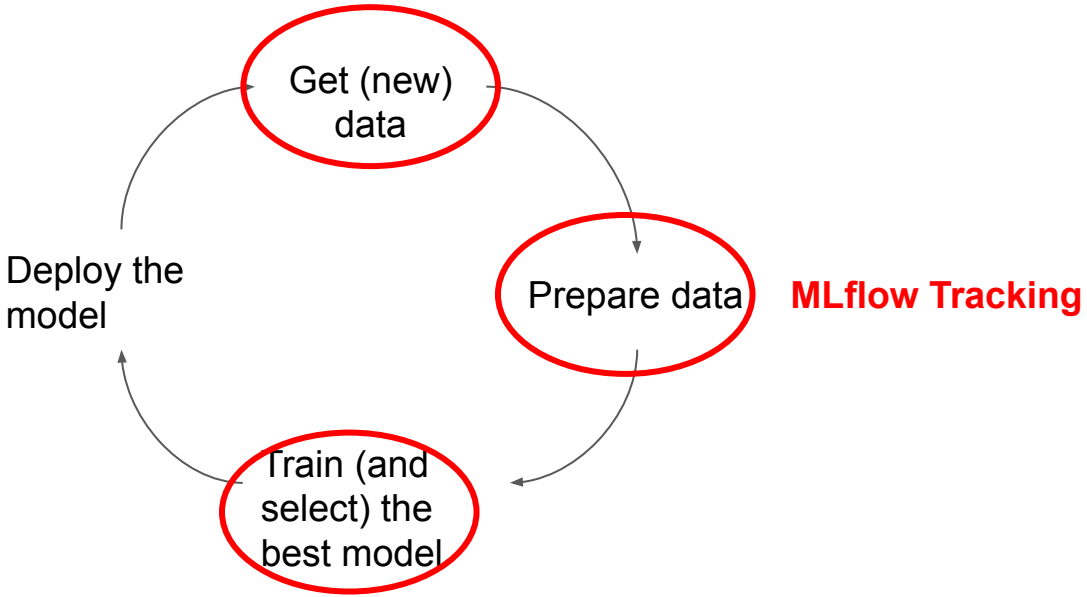
Main components

- **MLflow Tracking**:
  - Tracking ML experiments to record and compare model parameters, evaluate performance, and manage artifacts
- **MLflow Models**:
  - Packaging and deploying models from a variety of ML libraries to a variety of model serving and inference platforms
- **MLflow Model Registry**:
  - Collaboratively managing a central model store, including model versioning, stage transitions, and annotations
- **MLflow Projects**:
  - Packaging ML code in a reusable, reproducible form in order to share with other data scientists or transfer to production

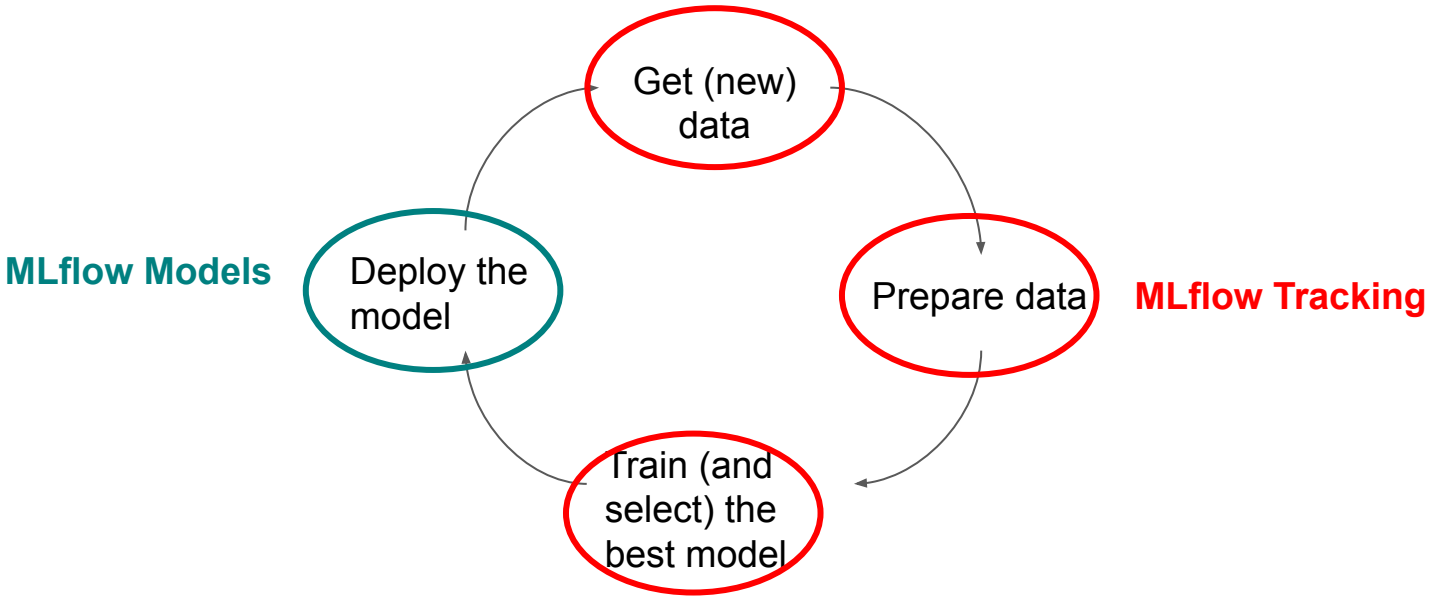
How is each component mapped?



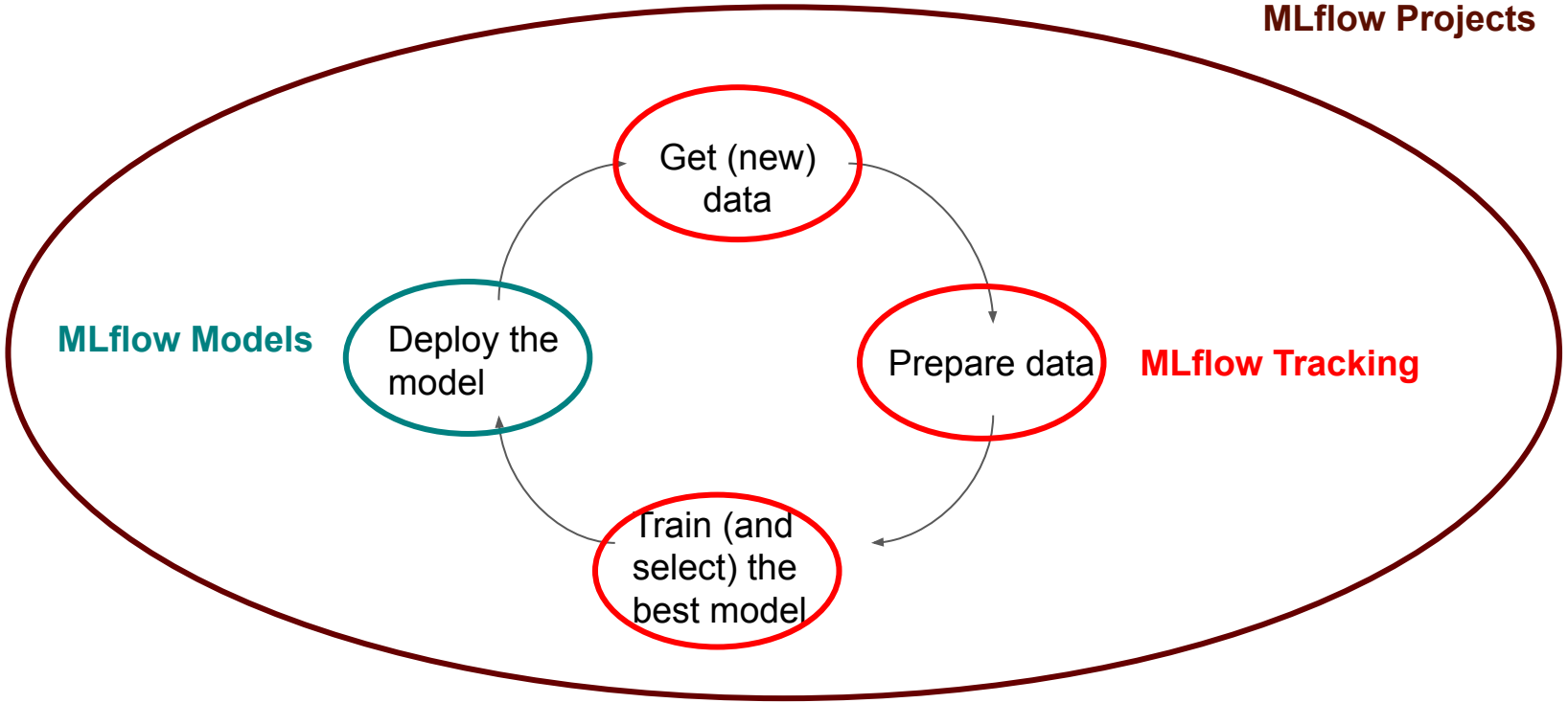
How is each component mapped?



How is each component mapped?



How is each component mapped?

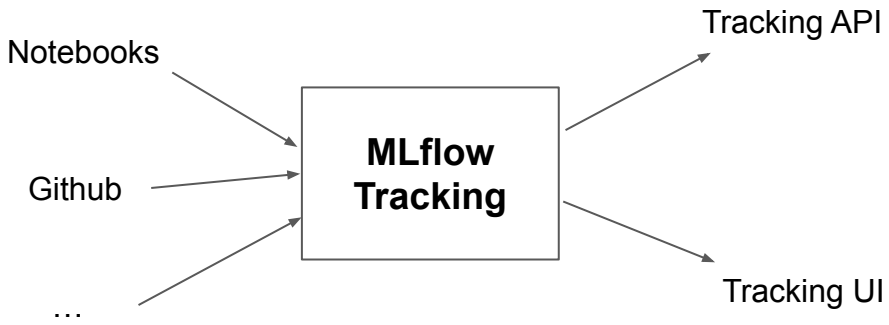


MLFlow tracking

The **MLflow Tracking** component is an **API and UI for logging** parameters, code versions, metrics, and output files when running your machine learning code and **for later visualizing** the results.

MLflow Tracking lets you log and query experiments using Python, REST, R API, and Java API APIs.

https://mlflow.org/docs/latest/tracking.html



MLFlow tracking - main concepts

MLFlow tracking is organized around the concept of **runs**, which are executions of some piece of data science code, collected in **experiments** (useful for comparing runs intended to tackle a particular task). Each run records the following information:

- **Code Version**: Git commit hash used for the run, if it was run from an MLflow Project.
- **Start & End Time**: Start and end time of the run
- **Source**: Name of the file to launch the run, or the project name and entry point for the run if run from an MLflow Project.
- **Parameters**: Key-value input parameters of your choice. Both keys and values are strings.
- **Metrics**: Key-value metrics, where the value is numeric. Each metric can be updated throughout the course of the run (for example, to track how your model's loss function is converging), and MLflow records and lets you visualize the metric's full history.
- **Artifacts**: Output files in any format. For example, you can record images (for example, PNGs), models (for example, a pickled scikit-learn model), and data files (for example, a Parquet file) as artifacts.

Once your runs have been recorded, you can query them using the Tracking UI or the MLflow API.

Tracking UI

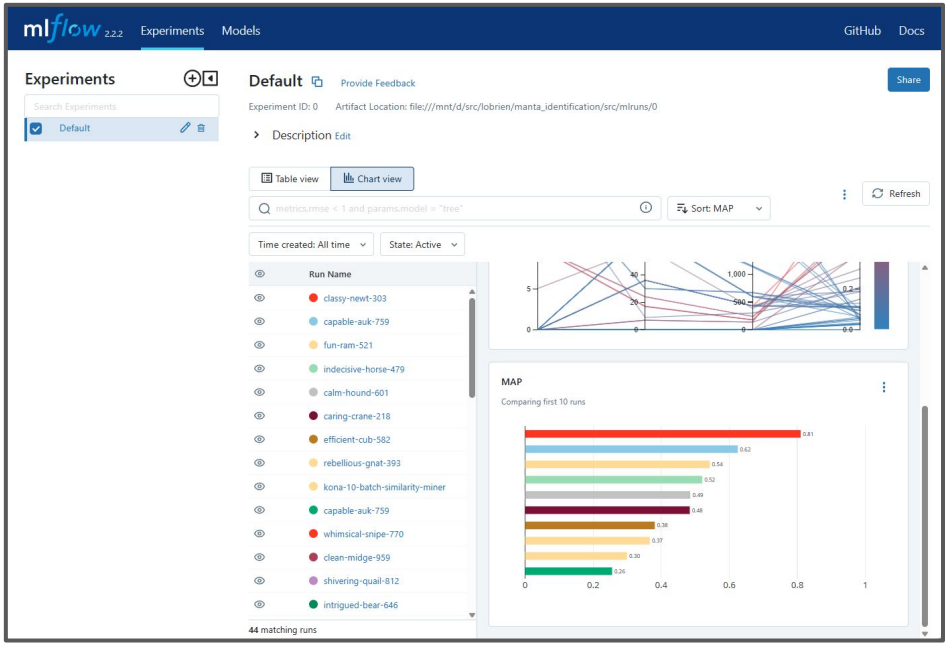
The Tracking UI lets you visualize, search and compare runs, as well as download run artifacts or metadata for analysis in other tools.

- If you log runs to a **local mlruns directory**, run `mlflow ui` in the directory above it, and it loads the corresponding runs.
- Alternatively, the **MLflow tracking server** serves the same UI and enables remote storage of run artifacts.
  - You run an MLflow tracking server using `mlflow server`
  - In that case, you can view the UI using URL http://<ip address of your MLflow tracking server>:5000 in your browser from any machine, including any remote machine that can connect to your tracking server.
  - To log to a tracking server, set the `MLFLOW_TRACKING_URI` environment variable to the server's URI, along with its scheme and port (for example, `http://10.0.0.1:5000`) or call `mlflow.set_tracking_uri()`

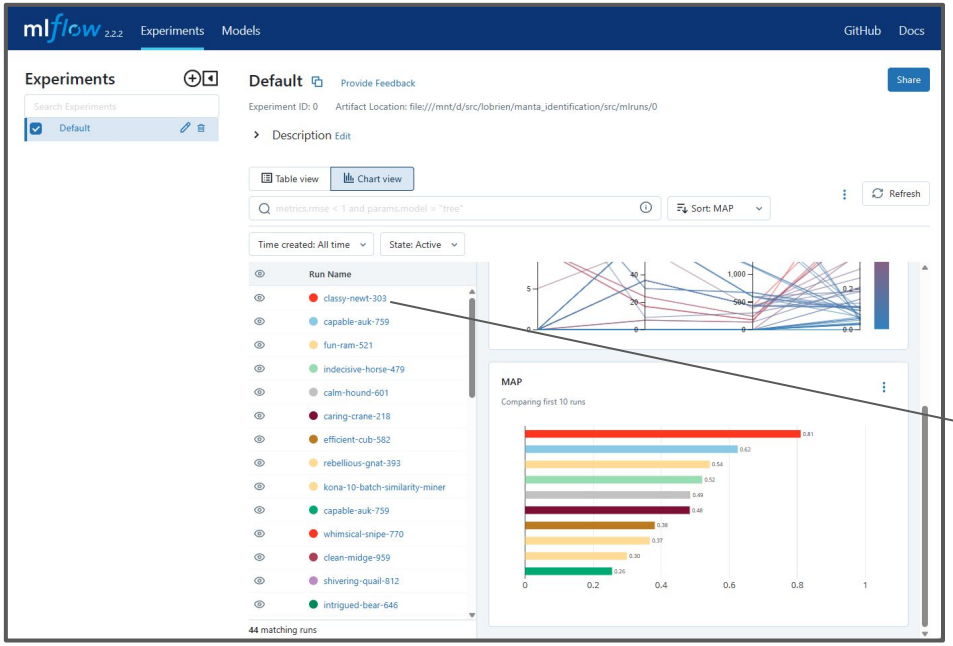
The UI contains the following key features:

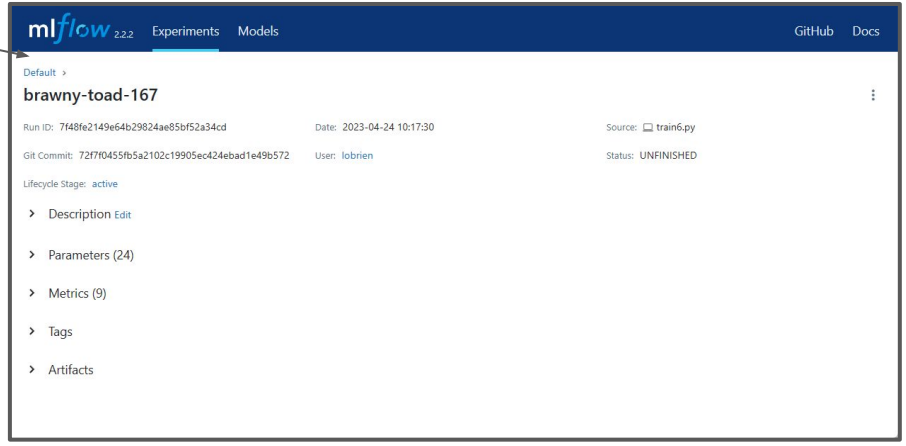
- Experiment-based run listing and comparison (including run comparison across multiple experiments)
- Searching for runs by parameter or metric value
- Visualizing run metrics
- Downloading run results

Tracking UI

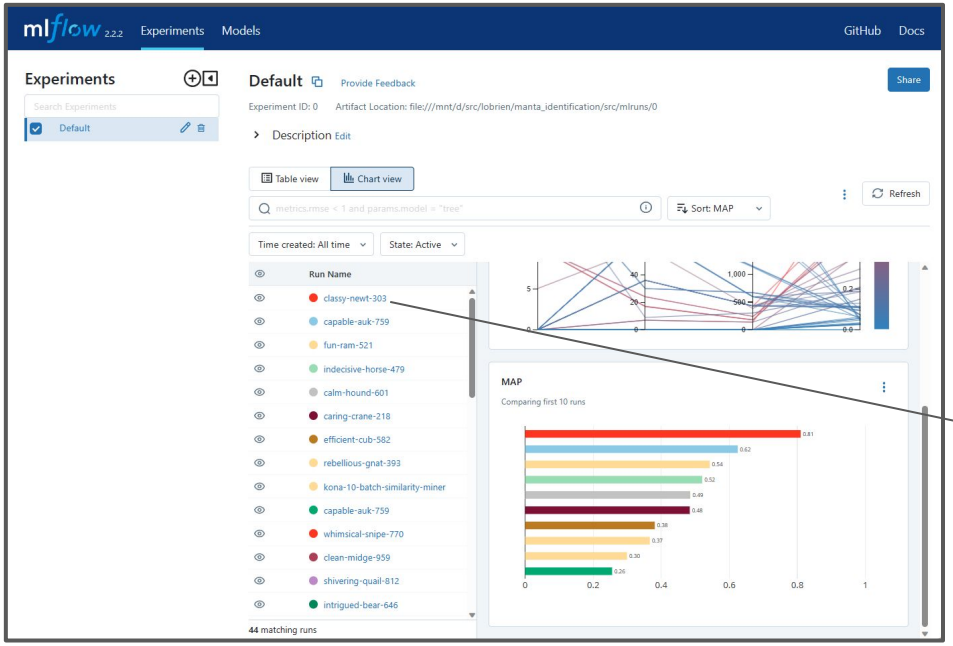

mlflow 2.2.2
Experiments
Models
GitHub
Docs
Experiments
Search Experiments
Default
Default
Provide Feedback
Share
Experiment ID: 0
Artifact Location: file:///mnt/d/src/lobrien/manta_identification/src/mlruns/0
Description Edit
Table view
Chart view
metrics.rmse < 1 and params.model = "tree"
Sort: MAP
Refresh
Time created: All time
State: Active
Run Name
classy-newt-303
capable-auk-759
fun-ram-521
indecisive-horse-479
calm-hound-601
caring-crane-218
efficient-cub-582
rebellious-gnat-393
kona-10-batch-similarity-miner
capable-auk-759
whimsical-snipe-770
clean-midge-959
shivering-quail-812
intrigued-bear-646
44 matching runs
MAP
Comparing first 10 runs

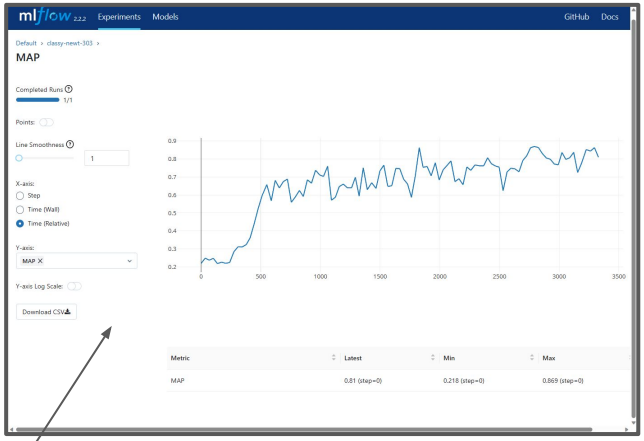
Tracking UI

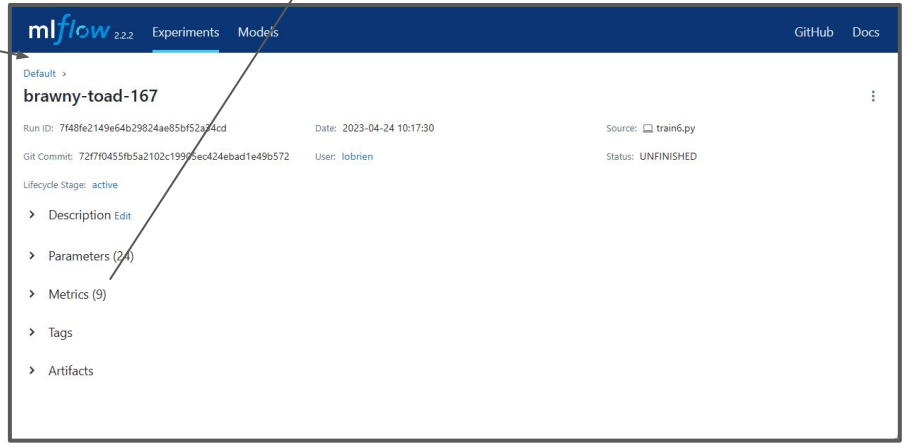

mlflow 2.2.2
Experiments
Models
GitHub
Docs
Experiments
Search Experiments
Default
Default
Provide Feedback
Share
Experiment ID: 0
Artifact Location: file:///mnt/d/src/lobrien/manta_identification/src/mlruns/0
Description Edit
Table view
Chart view
metrics.rmse < 1 and params.model = "tree"
Sort: MAP
Refresh
Time created: All time
State: Active
Run Name
classy-newt-303
capable-auk-759
fun-ram-521
indecisive-horse-479
calm-hound-601
caring-crane-218
efficient-cub-582
rebellious-gnat-393
kona-10-batch-similarity-miner
capable-auk-759
whimsical-snipe-770
clean-midge-959
shivering-quail-812
intrigued-bear-646
44 matching runs
MAP
Comparing first 10 runs


mlflow 2.2.2
Experiments
Models
GitHub
Docs
Default
brawny-toad-167
Run ID: 7f48fe2149e64b29824ae85bf52a34cd
Date: 2023-04-24 10:17:30
Source: train6.py
Git Commit: 72f7f0455fb5a2102c19905ec424ebad1e49b572
User: lobrien
Status: UNFINISHED
Lifecycle Stage: active
Description Edit
Parameters (24)
Metrics (9)
Tags
Artifacts

Tracking UI


mlflow 2.2.2
Experiments
Models
GitHub
Docs
Experiments
Default
Provide Feedback
Share
Experiment ID: 0
Artifact Location: file:///mnt/d/src/lobrien/manta_identification/src/mlruns/0
Description Edit
Table view
Chart view
Sort: MAP
Refresh
Time created: All time
State: Active
Run Name
classy-newt-303
capable-auk-759
fun-ram-521
indecisive-horse-479
calm-hound-601
caring-crane-218
efficient-cub-582
rebellious-gnat-393
kona-10-batch-similarity-miner
capable-auk-759
whimsical-snipe-770
clean-midge-959
shivering-quail-812
intrigued-bear-646
44 matching runs
MAP
Comparing first 10 runs


mlflow 2.2.2
Experiments
Models
GitHub
Docs
Default
classy-newt-303
MAP
Completed Runs
Points:
Line Smoothness
X-axis:
Step
Time (Wall)
Time (Relative)
Y-axis:
MAP
Y-axis Log Scale:
Download CSV
Metric
Latest
Min
Max
MAP
0.81 (step=0)
0.218 (step=0)
0.869 (step=0)


mlflow 2.2.2
Experiments
Models
GitHub
Docs
Default
brawny-toad-167
Run ID: 7f48fe2149e64b29824ae85bf52a74cd
Date: 2023-04-24 10:17:30
Source: train6.py
Git Commit: 72f7f0455fb5a2102c19905ec424ebad1e49b572
User: lobrien
Status: UNFINISHED
Lifecycle Stage: active
Description Edit
Parameters (24)
Metrics (9)
Tags
Artifacts

Where is data recorded?

- **MLflow runs** can be recorded to local files, to a SQLAlchemy-compatible database, or remotely to a tracking server.
- **MLflow artifacts** can be persisted to local files and a variety of remote file storage solutions.

For storing runs and artifacts, MLflow uses two components for storage: backend store and artifact store.

- **backend store** persists MLflow entities (runs, parameters, metrics, tags, notes, metadata, etc)
- **artifact store** persists artifacts (files, models, images, in-memory objects, or model summary, etc)

How to log?

“Manual” logging:

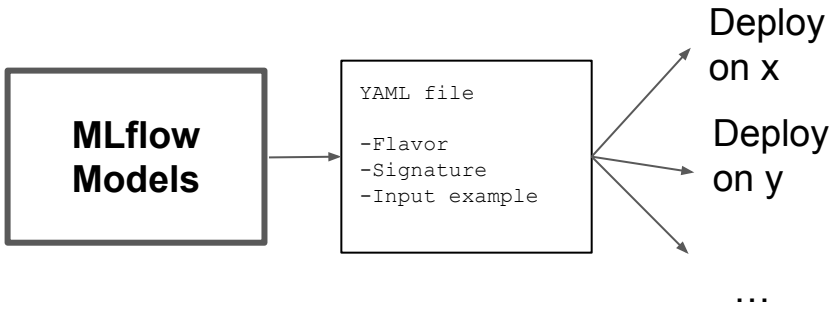
- **`mlflow.log_param()/mlflow.log_params()`**
  - logs a single key-value param in the currently active run. The key and value are both strings.
  - Use `mlflow.log_params()` to log multiple params at once.
- **`mlflow.log_metric() / mlflow.log_metrics()`**
  - logs a single key-value metric. The value must always be a number.
  - MLflow remembers the history of values for each metric (supports two alternative methods for distinguishing metric values on the x-axis: `timestamp` and `step`)
  - Use `mlflow.log_metrics()` to log multiple metrics at once.
- **`mlflow.log_input()`**
  - logs a single `mlflow.data.dataset.Dataset` object corresponding to the currently active run.
  - You may also log a dataset context string and a dict of key-value tags.
- **`mlflow.log_artifact()/mlflow.log_artifacts()`**
  - logs a local file or directory as an artifact, optionally taking an `artifact_path` to place it in within the run’s artifact URI.
  - Run artifacts can be organized into directories, so you can place the artifact in a directory this way.
  - `mlflow.log_artifacts()` logs all the files in a given directory as artifacts, again taking an optional `artifact_path`.

Autolog:

- Automatic logging allows you to log metrics, parameters, and models without the need for explicit log statements.
- There are two ways to use autologging:
  - Call `mlflow.autolog()` before your training code. This works for each supported library you have installed as soon as you import it.
  - Use library-specific autolog calls for each library you use in your code
    - available: Scikit-learn, Keras, Gluon, XGBoost, LightGBM, Statsmodels, Spark, Fastai, Pytorch

MLflow Models

https://mlflow.org/docs/latest/models.html



A standard format for **packaging machine learning models that can be used in a variety of downstream tools**

- Each MLflow Model is a directory containing arbitrary files, together with an MLmodel file in the root of the directory that can define multiple flavors that the model can be viewed in

- **Flavors**: Modules/frameworks that can be interpreted by deployment tools to understand the model without any separate integration mechanism for each tool or library used in the model.
  - Standard flavors provided by MLflow include python_function, sklearn, xgboost, etc
- **Model signature**: Defines the schema of the model's input and output parameters that can either be column-based or tensor-based
- **Input example**: Defines an instance of valid model input and stored as separate artifacts
- **MLmodel file**: a text file in YAML format that outlines multiple flavors the model can be viewed in, model signature and input example

MLflow Model Registry

The **MLflow Model Registry** component is a **centralized model store, set of APIs, and UI, to collaboratively manage the full lifecycle of an MLflow Model**. It provides model lineage (which MLflow experiment and run produced the model), model versioning, stage transitions (for example from staging to production), and annotations.

The Model Registry introduces a few concepts that describe and facilitate the full lifecycle of an MLflow Model.

- **Model**:
  - An MLflow Model is created from an experiment or run that is logged with one of the model flavor's mlflow.<model_flavor>.log_model() methods.
  - Once logged, this model can then be registered with the Model Registry.
- **Registered Model**:
  - An MLflow Model can be registered with the Model Registry.
  - A registered model has a unique name, contains versions, associated transitional stages, model lineage, and other metadata.
- **Model Version**:
  - Each registered model can have one or many versions.
  - When a new model is added to the Model Registry, it is added as version 1. Each new model registered to the same model name increments the version number.
- **Model Stage**:
  - Each distinct model version can be assigned one stage at any given time.
  - MLflow provides predefined stages for common use-cases (Staging, Production or Archived).
  - You can transition a model version from one stage to another stage.
- **Annotations and Descriptions**:
  - You can annotate the top-level model and each version individually using Markdown
- **Model Alias**:
  - You can create an alias for a registered model that points to a specific model version

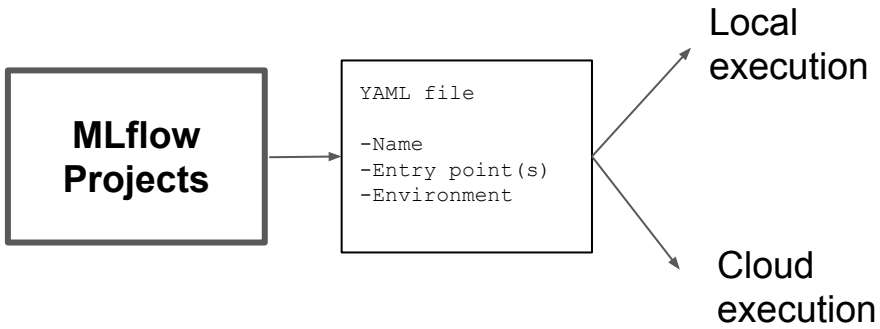
MLflow Projects

MLflow Projects are just a convention for organizing and describing your code (packaging it in a reusable and reproducible way) to let other data scientists (or automated tools) run it

A project is simply a directory of files, or a Git repository, containing your code + conventions for placing files in this directory and a MLproject file (YAML formatted). Each project can specify several properties:

- **Name**: A human-readable name for the project.
- **Entry Points**: Commands that can be run within the project, and information about their parameters. If you list your entry points in a MLproject file, however, you can also specify parameters for them, including data types and default values.
- **Environment**: The software environment that should be used to execute project entry points. This includes all library dependencies required by the project code (local, Conda, Virtualenv, and Docker)

https://mlflow.org/docs/latest/projects.html



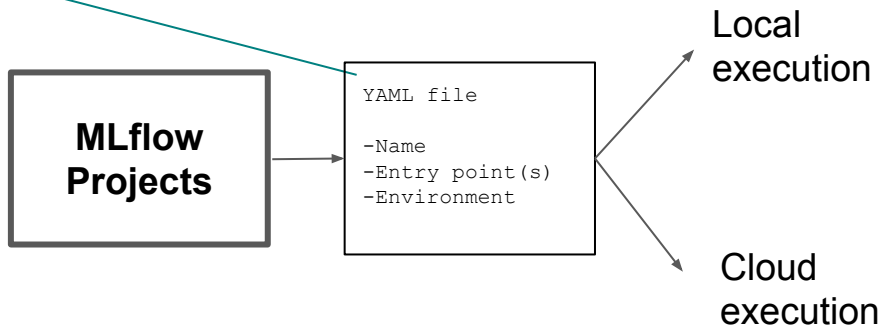
MLflow Projects

```yaml
name: My Project

python_env: python_env.yaml

entry_points:
  main:
    parameters:
      data_file: path
      regularization: {type: float, default: 0.1}
    command: "python train.py -r {regularization}
{data_file}"
  validate:
    parameters:
      data_file: path
      command: "python validate.py {data_file}"
```

https://mlflow.org/docs/latest/projects.html



MLflow Projects

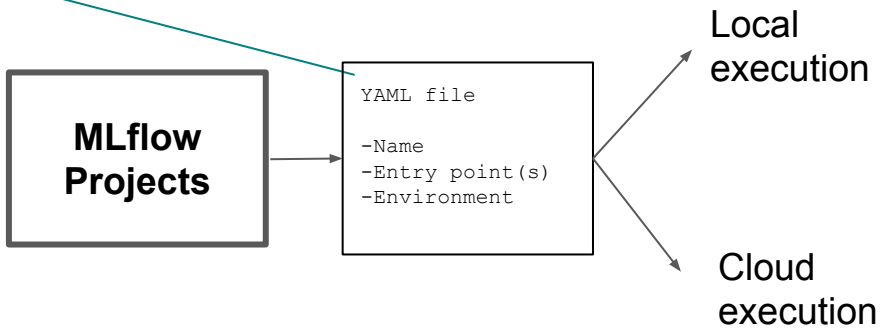
```yaml
name: My Project

python_env: python_env.yaml

entry_points:
  main:
    parameters:
      data_file: path
      regularization: {type: float, default: 0.1}
    command: "python train.py -r {regularization}
{data_file}"
  validate:
    parameters:
      data_file: path
      command: "python validate.py {data_file}"
```

```yaml
# Python version required to run the project.
python: "3.8.15"
# Dependencies required to build packages. This
field is optional.
build_dependencies:
  - pip
  - setuptools
  - wheel==0.37.1
# Dependencies required to run the project.
dependencies:
  - mlflow==2.3
    - scikit-learn==1.0.2
```

https://mlflow.org/docs/latest/projects.html



Building pipelines with MLflow Projects

SOSC23

The `mlflow.projects.run()` API, combined with other functions, makes it possible to build multi-step workflows with separate projects (or entry points in the same project) as the individual steps.

- Each call to `mlflow.projects.run()` returns a run object, that you can use with `mlflow.client` to determine when the run has ended and get its output artifacts
- These artifacts can then be passed into another step that takes path or uri parameters.
- You can coordinate all of the workflow in a single Python program that looks at the results of each step and decides what to submit next using custom code.
  - Modularizing Your Data Science Code
    - Different users can publish reusable steps for data featurization, training, validation, and so on, that other users or team can run in their workflows
    - Sometimes you want to run the same training code on different random splits of training and validation data
  - Hyperparameter Tuning
    - Using `mlflow.projects.run()` you can launch multiple runs in parallel
    - Your driver program can then inspect the metrics from each run in real time to cancel runs, launch new ones, or select the best performing run on a target metric

Papermill

Papermill is a tool for parameterizing, executing, and analyzing Jupyter Notebooks.

Papermill lets you:

- parameterize notebooks
- execute notebooks

You can programmatically execute a workflow without having to copy and paste from notebook to notebook manually

To parameterize your notebook designate a cell with the tag parameters:

- Papermill looks for the parameters cell and treats this cell as defaults for the parameters passed in at execution time. Papermill will add a new cell tagged with injected-parameters with input parameters in order to overwrite the values in parameters. If no cell is tagged with parameters the injected cell will be inserted at the top of the notebook.

https://github.com/nteract/papermill

Worldwide integration/adoption

- MLflow is well integrated with most of the Machine Learning ecosystem:
  - Tensorflow
  - Pythorch
  - Keras
  - Scikit-learn
  - XGBoost
  - Onnx
  - …
- Also integrated with different computing environments: docker, kubernetes, commercial clouds…
- Adopted by 80+ companies

Let’s start by examples

- We already set up an MLflow tracking server for you at: https://<your-username>-mlflow.131.154.99.220.myip.cloud.infn.it

Let’s start by examples

- We already set up an MLflow tracking server for you at: https://<your-username>-mlflow.131.154.99.220.myip.cloud.infn.it
- Didactic examples: https://github.com/SOSC-School/SOSC23-livesessions/tree/main/day4/MLflow

Let’s start by examples

- We already set up an MLflow tracking server for you at: https://<your-username>-mlflow.131.154.99.220.myip.cloud.infn.it
- Didactic examples: https://github.com/SOSC-School/SOSC23-livesessions/tree/main/day4/MLflow
  - Exercise for you:
    - Try and add a test for the trained model on a dummy pandas dataframe
    - Bonus:
      - Try and modify the main.py function in order to make a grid search on the parameters of the elasticnet model and then test the best model on a dummy pandas dataframe

Let’s start by examples

- We already set up an MLflow tracking server for you at: https://<your-username>-mlflow.131.154.99.220.myip.cloud.infn.it
- Didactic examples: https://github.com/SOSC-School/SOSC23-livesessions/tree/main/day4/MLflow
  - Exercise for you:
    - Try and add a test for the trained model on a dummy pandas dataframe
    - Bonus:
      - Try and modify the main.py function in order to make a grid search on the parameters of the elasticnet model and then test the best model on a dummy pandas dataframe
- More sophisticated example: https://github.com/mlflow/mlflow/tree/master/examples/hyperparam