Versioning

Experiment Tracking

Agenda

- Versioning & Tools
- Experiment Tracking
- MLflow
- MLflow Components:
    - Tracking
    - Model
    - Model Registry
    - Project
- Lab

Versioning & Tools

Code Versioning

GitHub

Bitbucket

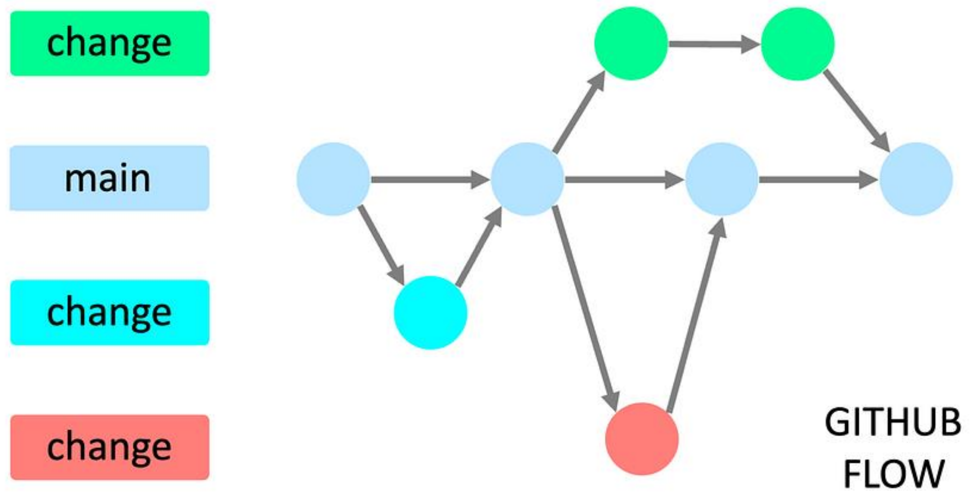
Azure Repos

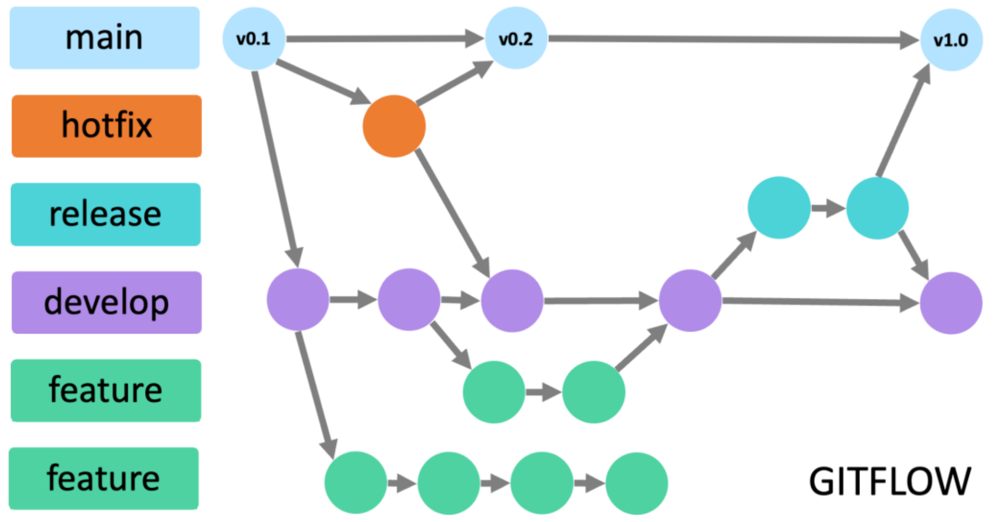
pyter Source File	3,961 KB
pyter Source File	3,962 KB
pyter Source File	4,097 KB

Code Versioning - Git

As computer engineers, we recognize the importance of code versioning and branching strategies like Git Flow and GitHub Flow in streamlining software development.

Both approaches provide a structured way to manage code changes through **feature branches**, ensure the **stability** of the **main branch**, and support collaboration via best practices such as meaningful commits and **code reviews** through **pull requests**. Despite differences in complexity, they share a common goal of enabling parallel development and integrating new features without disrupting the main codebase.





Code Versioning - Git

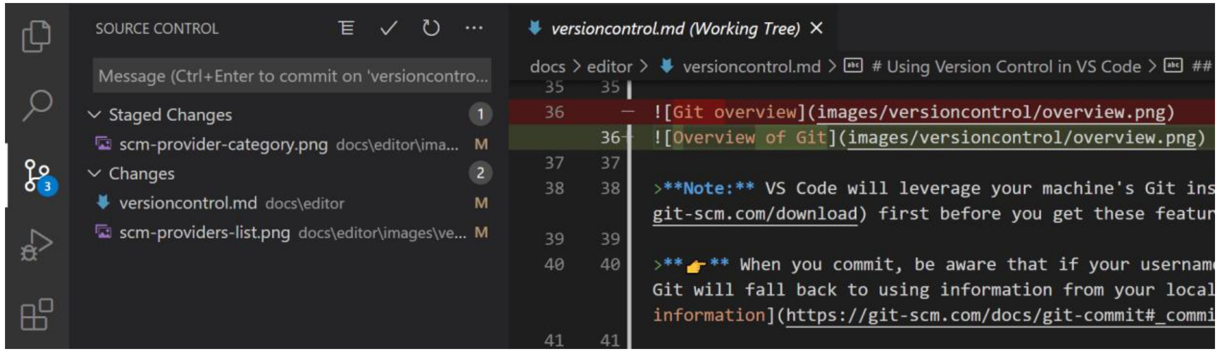
For more information on branching and commit conventions, refer to:

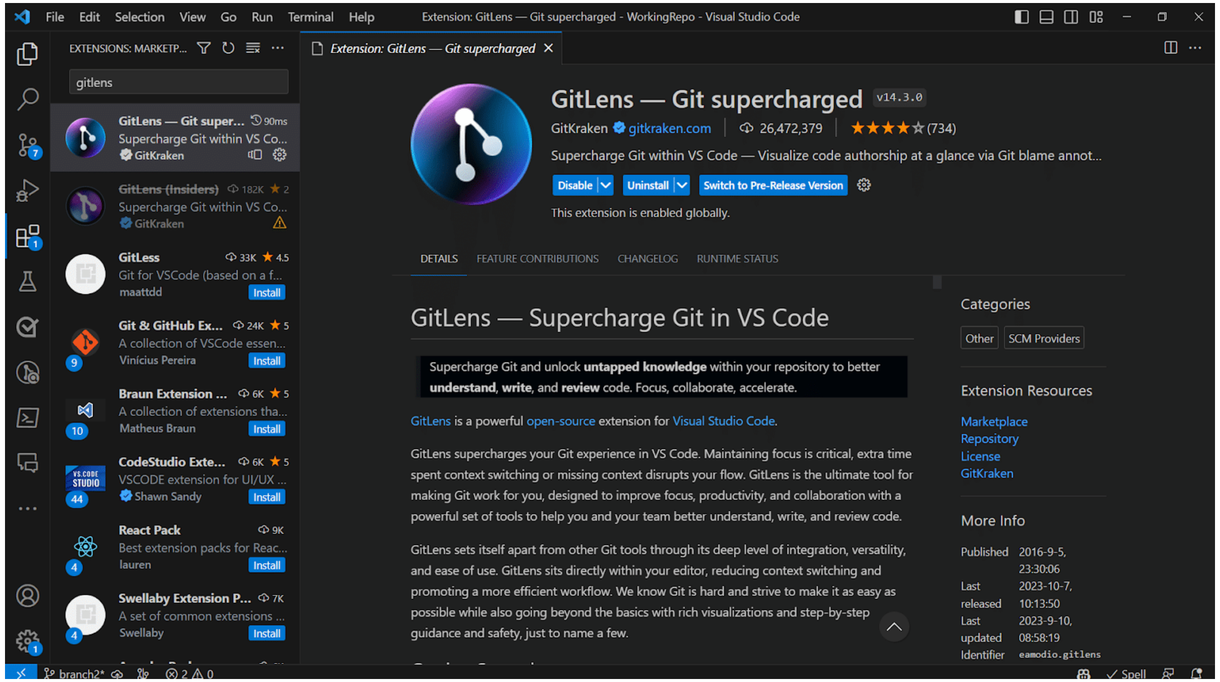
- Git Branching And Commit Message Convention
- Conventional Commits

Code Versioning – VSCode Extensions

Tips & Tricks

Know more here





Workflow Orchestration

Apache Airflow

Code Versioning

GitHub

Bitbucket

Azure Repos

Data Versioning

DELTA LAKE

lakeFS

DVC

Experiment Tracking

mlflow™

W&B

- runs
  - detect
    - train
    - train2
    - yolov11l
    - yolov11m

Experiment Tracking

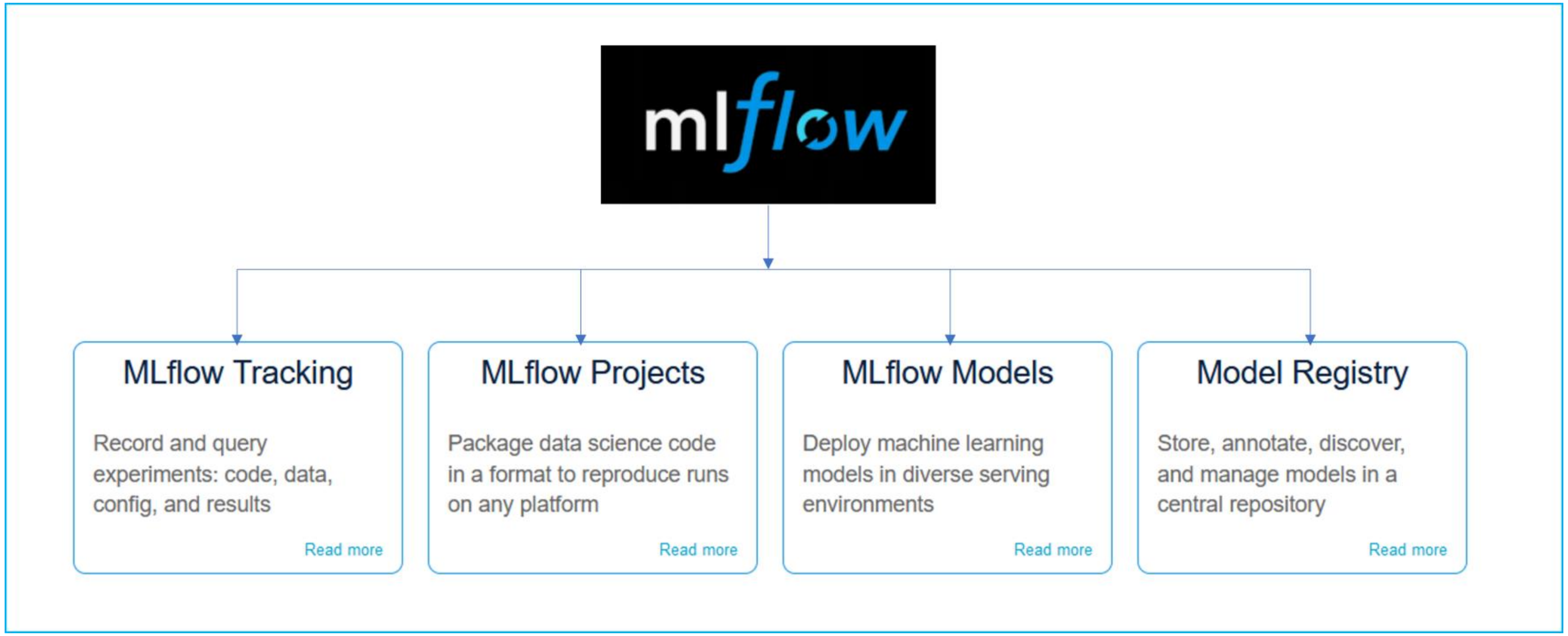
Experiment trackers let you track your ML experiments by **logging extended information** about your models, datasets, metrics, and other parameters and allowing you to browse them, visualize them and **compare them between runs.**

- Track What?
  - Track hyperparameters
  - Track model
  - Track evaluation metrics e.g. RMSE
- Why?
  - Reproduce an experiment
  - Compare different models
  - Managing the versioning and reproducibility of ML models and datasets is crucial for **ensuring the reliability and consistency of results**

MLflow

Experiment Tracking – MLflow

- What's MLflow?
  - Mlflow is an open-source platform to manage the ML lifecycle, including experimentation, reproducibility, deployment, and a central model registry
- Components of MLflow



Experiment Tracking – MLflow

Features of MLflow:

- 🪽 **Free and Open** - MLflow is **open source** and FREE. Avoid vendor lock-in and secure your ML assets on your own choice of infrastructure. You can invest in the most critical business goals without worrying about the cost of the MLOps platform.

- 👥 **Community** - MLflow boasts a vibrant Open Source community as a part of the Linux Foundation. With **19,000+** GitHub Stars and **15MM+** monthly downloads, MLflow is a trusted standard in the MLOps/LLMOps ecosystem.

- 🔄 **End-to-End** - MLflow is designed for managing the end-to-end machine learning lifecycle, eliminating the complexity of managing multiple tools and moving assets between them.

- 🌎 **Domain Agnostic** - Real world problems are not always solved by GenAI alone. MLflow provides a unified platform for managing traditional ML, deep learning, and GenAI models, making it a versatile tool for all your ML needs.

- 🛠️ **Native Cloud Integration** - MLflow is not only self-hosting, but also offered managed service on Amazon SageMaker, Azure ML, Databricks, and more. This allows you to getting started quickly and scale easily within your existing cloud infrastructure.

- 🤝 **15+ GenAI Framework Support** - MLflow integrates with more than a dozen GenAI libraries, including OpenAI, LangChain, LlamaIndex, DSPy, and more. This allows flexibility in choosing the right tool for your use case.

Experiment Tracking – MLflow

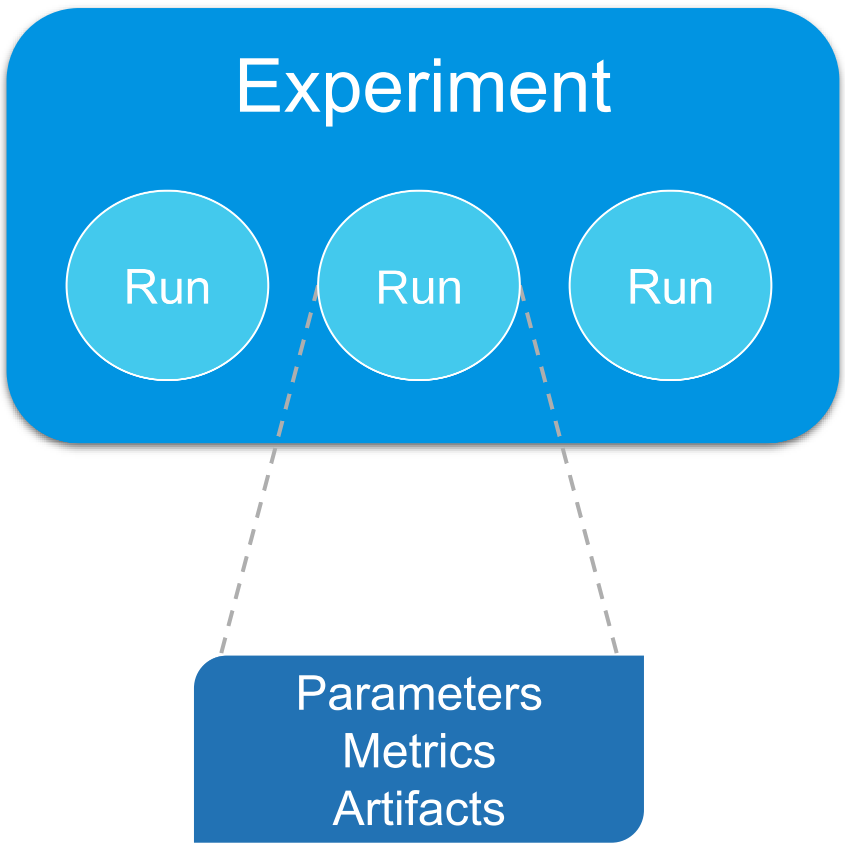
- Documentation Documentation Documentation
  - Link
  - But why do we need it? Don’t we have ChatGPT or anything else?
    - Getting familiar with information extraction from documentation is **crucial**
    - You need to practice writing the code yourself from time to time, not relying on copying code immediately.
    - You’ll get into different situations in which you need to act fast and not get into the hustle of explaining and guiding ChatGPT to get you the answer you need.
      Situation as a technical interview in which you’re allowed to use documentation, in a meeting with a senior, or a stressful work time in which you need to finish your task faster before a meeting.

MLflow Tracking

Experiment Tracking – MLflow Tracking

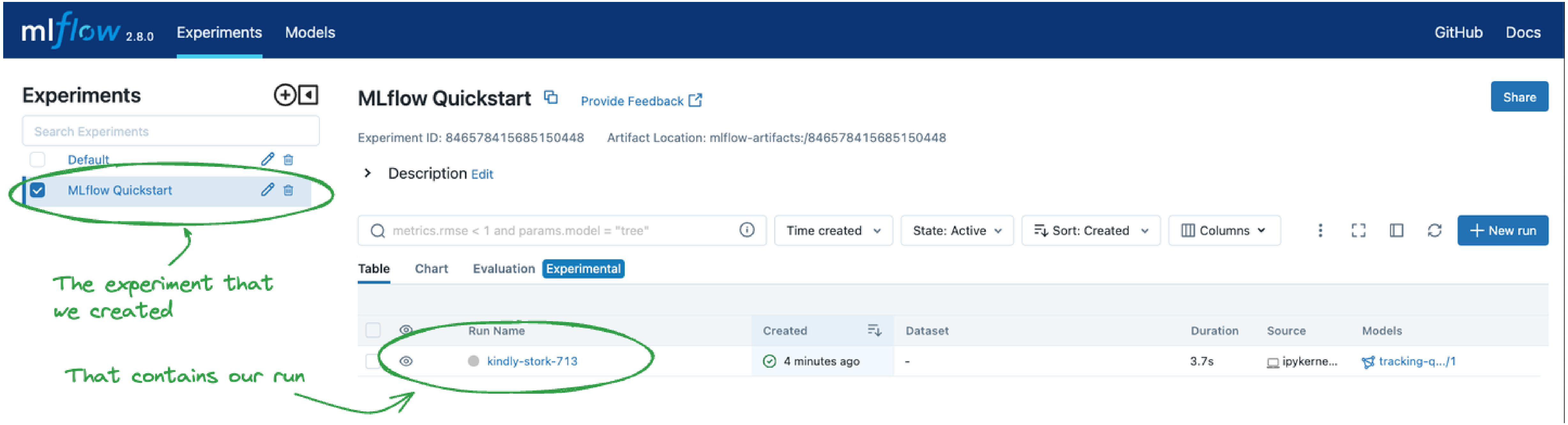
key concepts of MLflow Tracking are:

- **Experiment**: represents a specific machine learning task or project. It acts as a container for runs and helps organize and group related runs together.
- **Run**: represents a specific execution of an MLflow script or code. It captures the parameters, metrics, and artifacts generated during the run.
- **Parameters**: are inputs or configurations that define an MLflow run. They can be hyperparameters, model configurations, or any other variables that affect the experiment's outcome.
- **Metrics**: are measurements or evaluation criteria used to assess the performance of a model during training or evaluation. MLflow allows logging various metrics such as accuracy, loss, F1-score, or any other custom metric.
- **Artifacts**: are the output files generated during an MLflow run, such as trained models, visualizations, or data files.
- **Tags**: are user-defined key-value pairs that provide additional metadata for experiments and runs. They can be used to add descriptive labels, track specific attributes, or categorize experiments based on certain criteria.



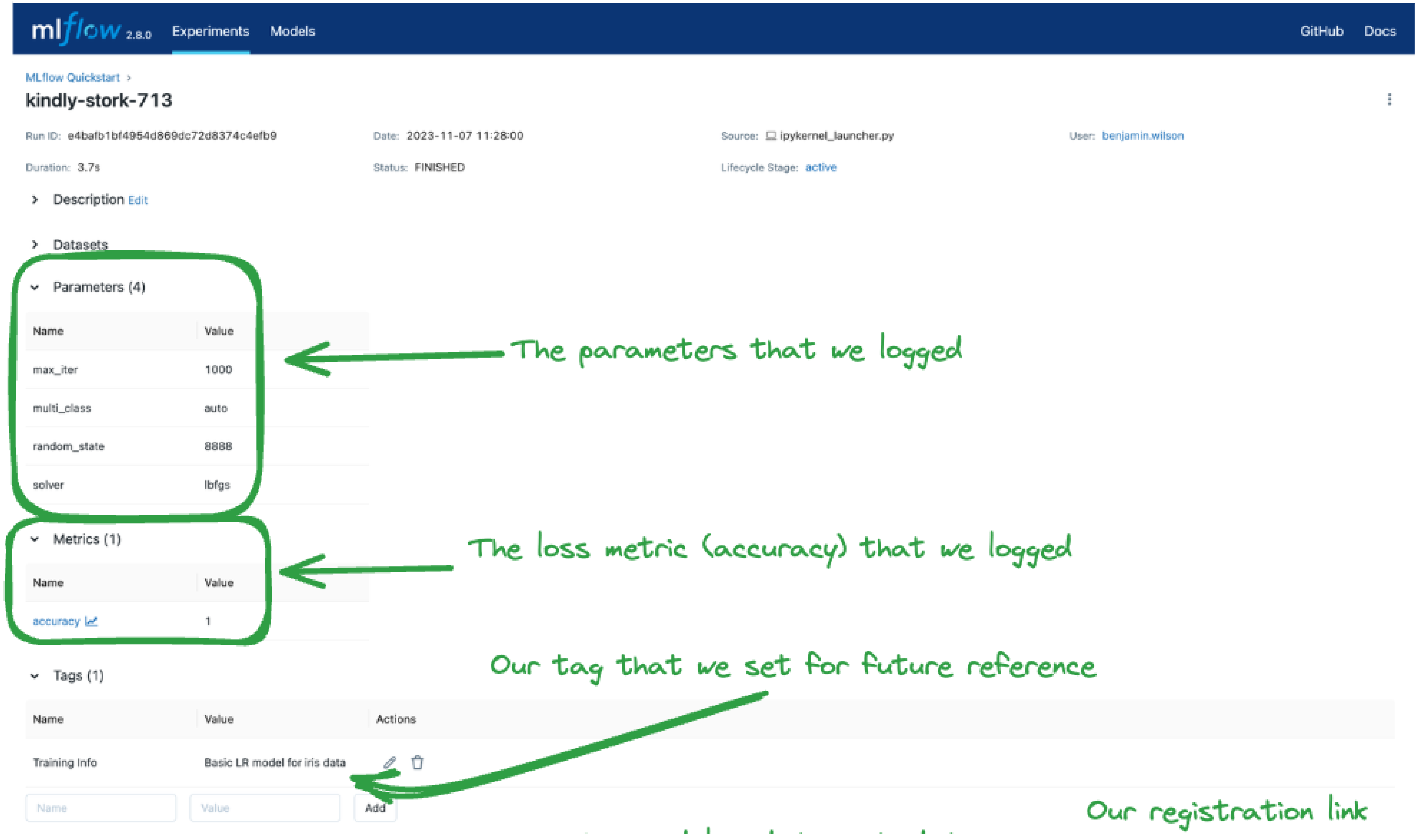
Experiment Tracking – MLflow Tracking

- **MLflow UI**
  - Command: mlflow ui



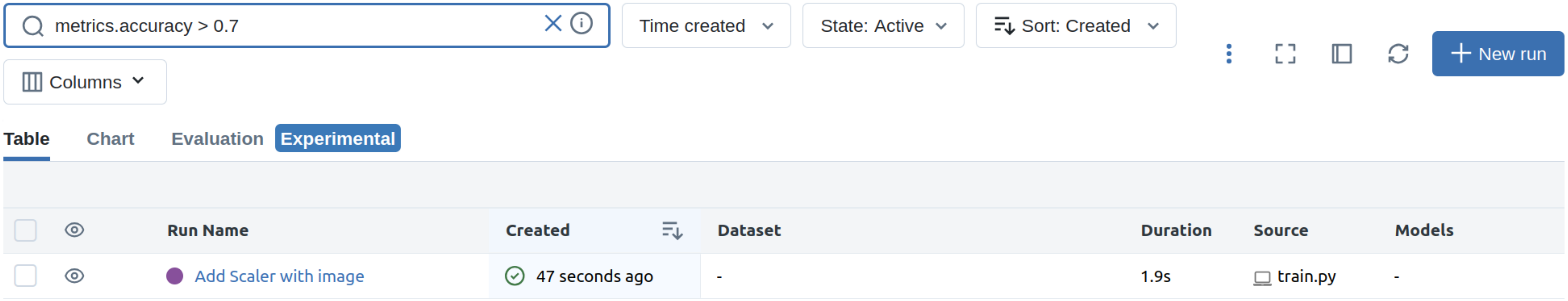
Experiment Tracking – MLflow Tracking

- **MLflow UI** of the run



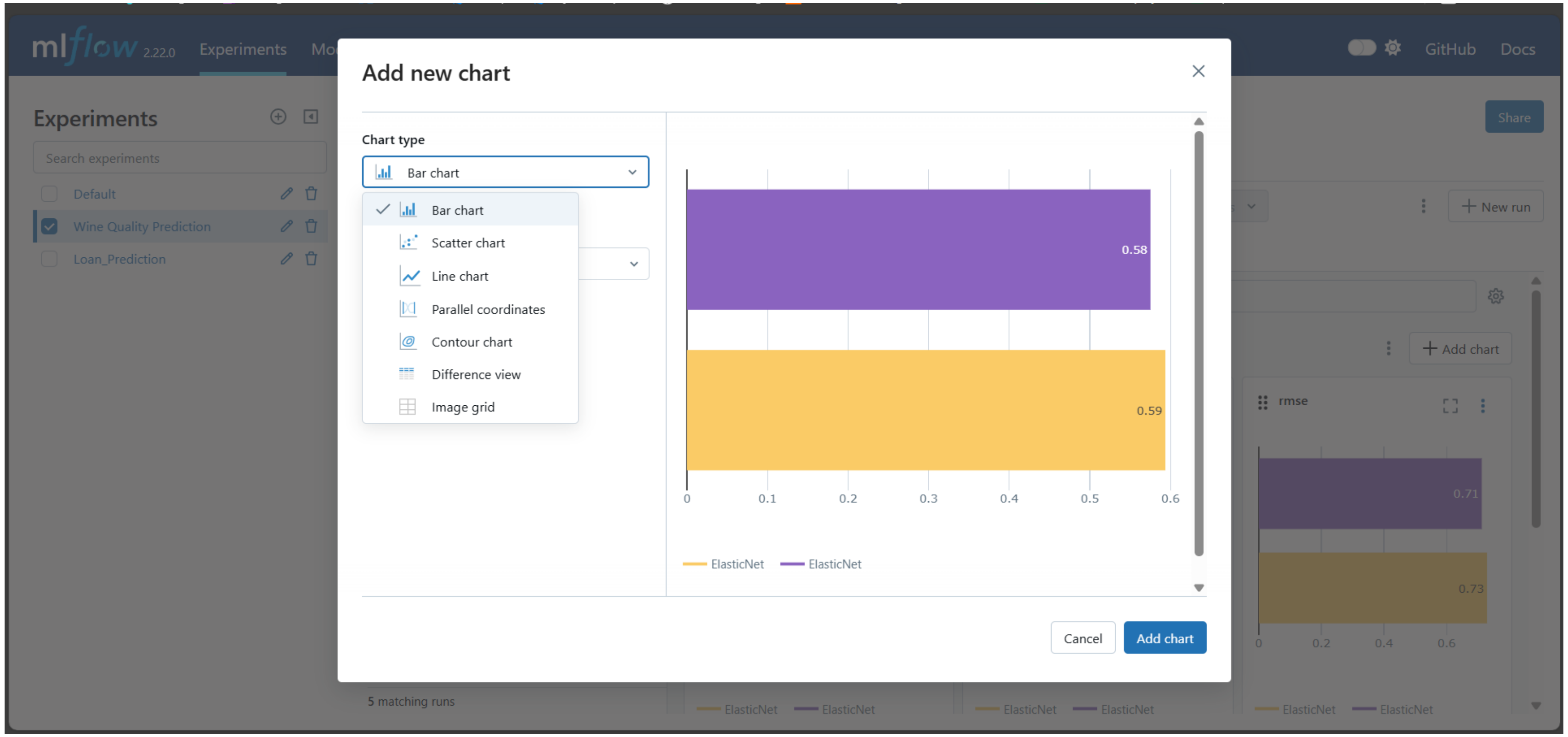
Experiment Tracking – MLflow Tracking

In **MLflow UI**, it is also possible to filter **runs**, for example, selecting those with some metric above a threshold.



Experiment Tracking – MLflow Tracking

You can utilize the logged data and visualize it in a chart



Let’s Code IT

Experiment Tracking – MLflow

- Prepare your conda/virtual environment
  - Conda create –n *env-name*
  - Conda activate *env-name*
- Install
  - Pip install mlflow

Experiment Tracking – MLflow Tracking

MLFlow tracking methods

- `mlflow.set_tracking_uri()`
  connects to a tracking URI. You can also set the MLFLOW_TRACKING_URI environment variable to have MLflow find a URI from there. In both cases, the URI can either be a HTTP/HTTPS URI for a remote server, a database connection string, or a local path to log data to a directory. The URI defaults to mlruns.

- `mlflow.get_tracking_uri()`
  returns the current tracking URI.

- `mlflow.create_experiment()`
  creates a new experiment and returns its ID. Runs can be launched under the experiment by passing the experiment ID to `mlflow.start_run()`.

- `mlflow.set_experiment()`
  sets an experiment as active. If the experiment does not exist, creates a new experiment. If you do not specify an experiment in `mlflow.start_run()`, new runs are launched under this experiment.

- `mlflow.start_run()`
  returns the currently active run (if one exists) or starts a new run and returns a mlflow. ActiveRun object usable as a context manager for the current run. You do not need to call start_run explicitly: calling one of the logging functions with no active run automatically starts a new one.

- `mlflow.end_run()`
  ends the currently active run, if any, taking an optional run status.

Experiment Tracking – MLflow Tracking

MLFlow tracking methods

- `mlflow.log_param()`
  logs a single key-value param in the currently active run. The key and value are both strings. Use it to log multiple params at once.
- `mlflow.log_metric()`
  logs a single key-value metric. The value must always be a number. MLflow remembers the history of values for each metric. Use it to log multiple metrics at once.
- `mlflow.log_artifact()`
  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.set_tag()`
  sets a single key-value tag in the currently active run. The key and value are both strings. Use it to set multiple tags at once.

Experiment Tracking – MLflow Tracking

One of the tags that can be used for the run is a **version** number.

But how to specify the version Number?

- Semantic Versioning a widely used system to convey the nature and impact of changes in software. It's in the format:
- **MAJOR.MINOR.PATCH**
  - **EX. 2.1.3**
  - MAJOR version
    - This indicates **breaking changes** – changes that are **not backward compatible**.
    - Switching from version 1.x.x to 2.0.0 might mean a function was renamed or removed, or the API structure changed.
  - MINOR version
    - This represents **new features** that are **backward compatible**.
      It means new functionality was added, but nothing was removed or changed in a way that would break existing code.
    - Version 2.1.0 adds a new function or improves performance, but everything that worked in 2.0.x still works.
  - PATCH version
    - This shows **bug fixes or small changes** that are backward compatible and don't add new features.
    - Version 2.1.3 fixes a bug found in 2.1.2 without adding any new features or breaking anything.
- **Backward Compatibility Means** Older versions of the system (apps, APIs, code) can **still work** after the update **without needing changes**.

Experiment Tracking – MLflow Tracking

Facebook App – Semantic Versioning Examples

- **Version 1.0.0 – Initial Public Release**
  - Basic functionality: post, like, comment, view feed.
- **Version** [redacted]
  - Added the ability to react with ❤️ 😂 😮 😢 😡 instead of just "Like".
  - [redacted]
- **Version** [redacted]
  - Fixed a crash when uploading videos.
  - [redacted]
- **Version** [redacted]
  - Completely redesigned the app layout and user navigation.
  - Feed logic now includes Stories, Reels, and algorithmic sorting.
  - Removed support for Android versions below 8.0.
  - [redacted]
- **Version** [redacted]
  - Added “Dark Mode” toggle in Settings.
  - Introduced the Marketplace tab.
  - [redacted]
- **Version** [redacted]
  - Fixed incorrect timestamps on comments.
  - Improved photo loading speed.
  - [redacted].

Experiment Tracking – MLflow Tracking

Facebook App – Semantic Versioning Examples

- **Version 1.0.0 – Initial Public Release**
  - Basic functionality: post, like, comment, view feed.
- **Version 1.1.0 – Minor Update (New Feature)**
  - Added the ability to react with ❤️ 😂 😮 😢 😡 instead of just "Like".
  - **Why MINOR?** New feature, backward compatible.
- **Version 1.1.1 – Patch (Bug Fix)**
  - Fixed a crash when uploading videos.
  - **Why PATCH?** Bug fix, no new features or breaking changes.
- **Version 2.0.0 – Major Update (Breaking Change)**
  - Completely redesigned the app layout and user navigation.
  - Feed logic now includes Stories, Reels, and algorithmic sorting.
  - Removed support for Android versions below 8.0.
  - **Why MAJOR?** Breaking changes and removed old support.
- **Version 2.2.0 – Minor Update**
  - Added “Dark Mode” toggle in Settings.
  - Introduced the Marketplace tab.
  - **Why MINOR?** New features added in a backward-compatible way.
- **Version 2.2.5 – Patch Update**
  - Fixed incorrect timestamps on comments.
  - Improved photo loading speed.
  - **Why PATCH?** Internal improvements, no behavior changes.

Experiment Tracking – MLflow Tracking

Notice that there’s a folder called “mlruns” is created in your project directory. This folder is where the tracking will be stored.

One more interesting callout is that by default you get three way to manage your model’s environment:

- python_env.yaml (python virtualenv)
- requirements.txt (PyPi requirements)
- conda.yaml (conda env)

```
mlruns/
├── 0/                                  # Experiment ID
│   ├── bc6dc2a4f38d47b4b0c99d154bbc77ad/ # Run ID
│   │   ├── metrics/
│   │   │   └── mse                     # Example metric file for mean squared error
│   │   ├── artifacts/                  # Artifacts associated with our run
│   │   │   └── sklearn-model/
│   │   │       ├── python_env.yaml
│   │   │       ├── requirements.txt    # Python package requirements
│   │   │       ├── MLmodel             # MLflow model file with model metadata
│   │   │       ├── model.pkl           # Serialized model file
│   │   │       ├── input_example.json
│   │   │       └── conda.yaml
│   │   ├── tags/
│   │   │   ├── mlflow.user
│   │   │   ├── mlflow.source.git.commit
│   │   │   ├── mlflow.runName
│   │   │   ├── mlflow.source.name
│   │   │   ├── mlflow.log-model.history
│   │   │   └── mlflow.source.type
│   │   ├── params/
│   │   │   ├── max_depth
│   │   │   └── random_state
│   │   └── meta.yaml
│   └── meta.yaml
```


TAKE A BREAK!

Mind Sprint

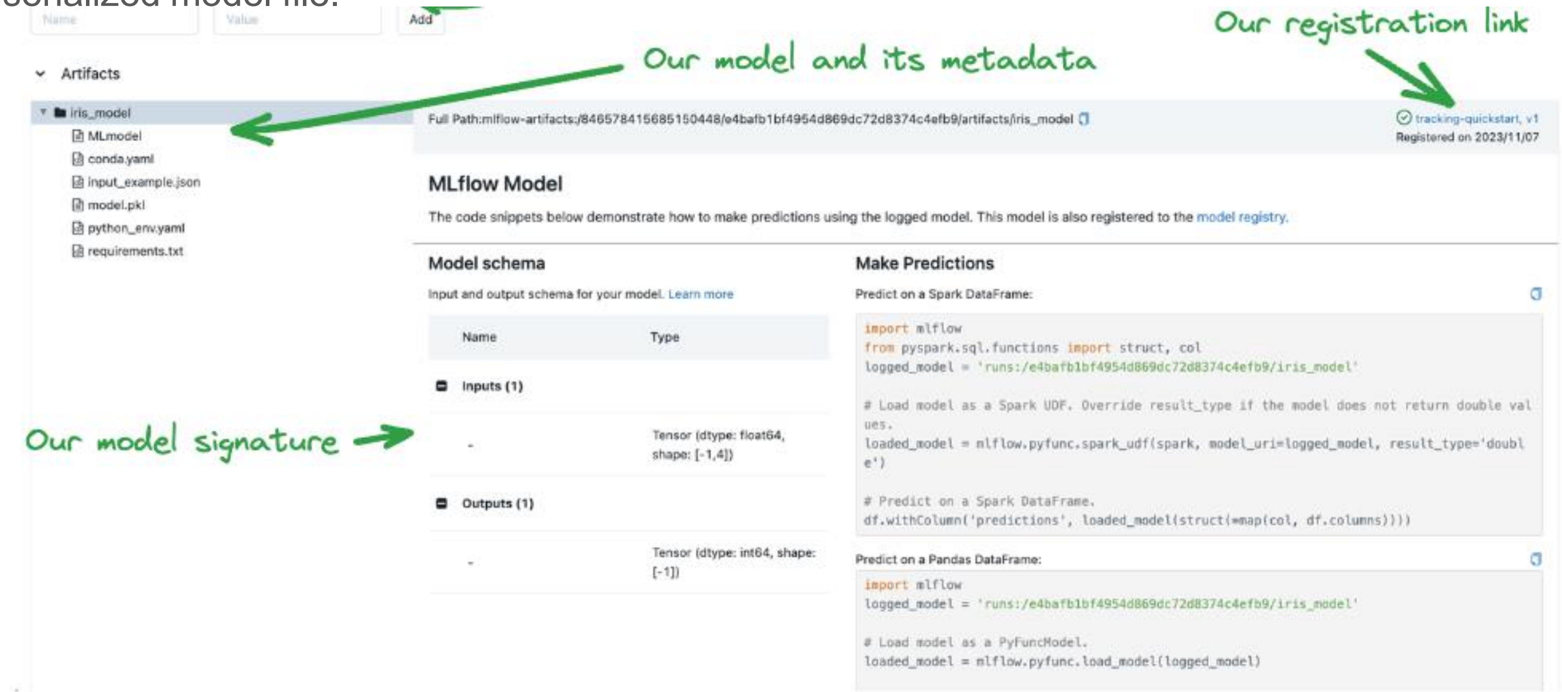
Example

Experiment Tracking – MLflow Tracking

Additionally, we can log the model using `mlflow.sklearn.log_model().`
In the Artifacts tab of the run, you'll find the following contents:

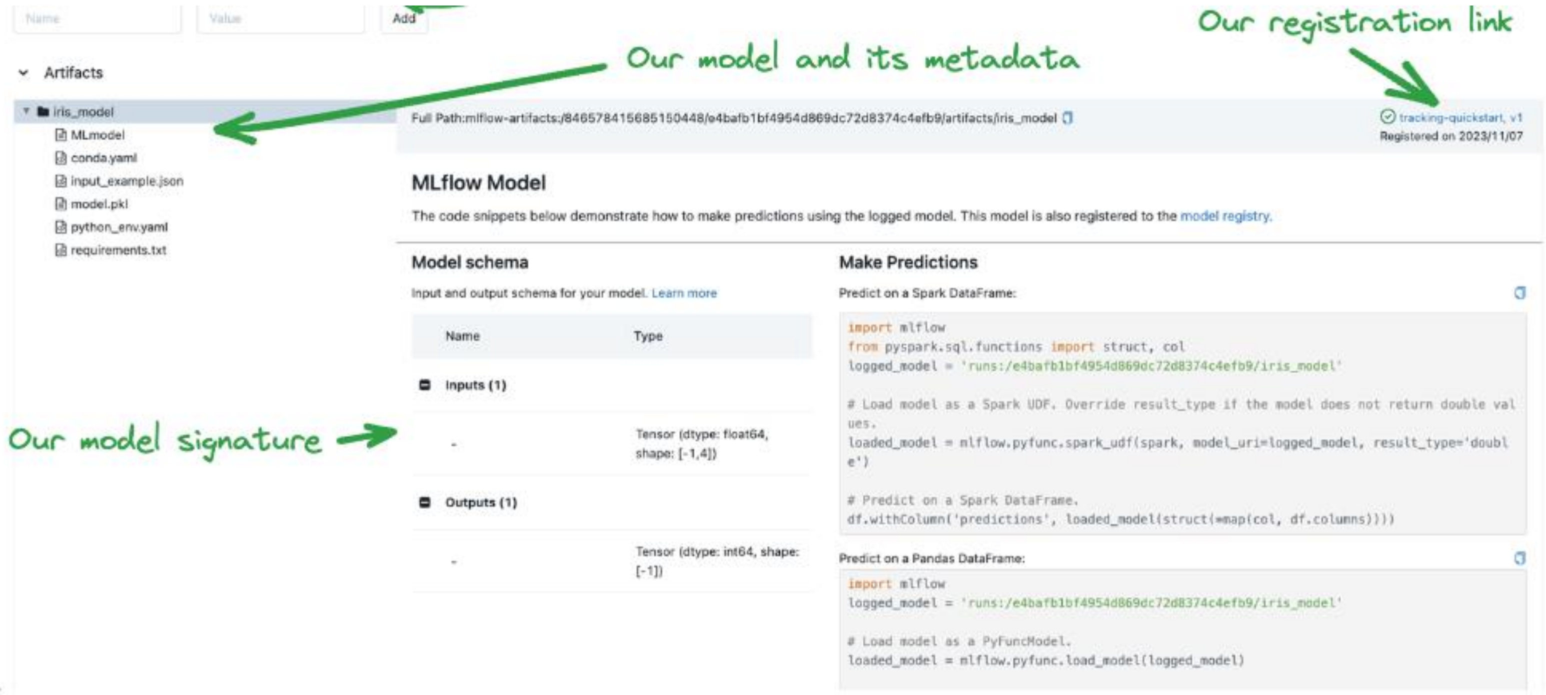
- **MLmodel**: Contains metadata about the logged model.
- **conda.yaml**, **python_env.yaml**, **requirements.txt**: Define the environment, including packages and their versions.
- **model.pkl**: The serialized model file.

MLflow also supports model signatures, which validate the model's expected inputs and outputs. By providing an input_example when using `log_model(),` the signature is automatically inferred and logged. This approach is recommended for its simplicity and clarity.



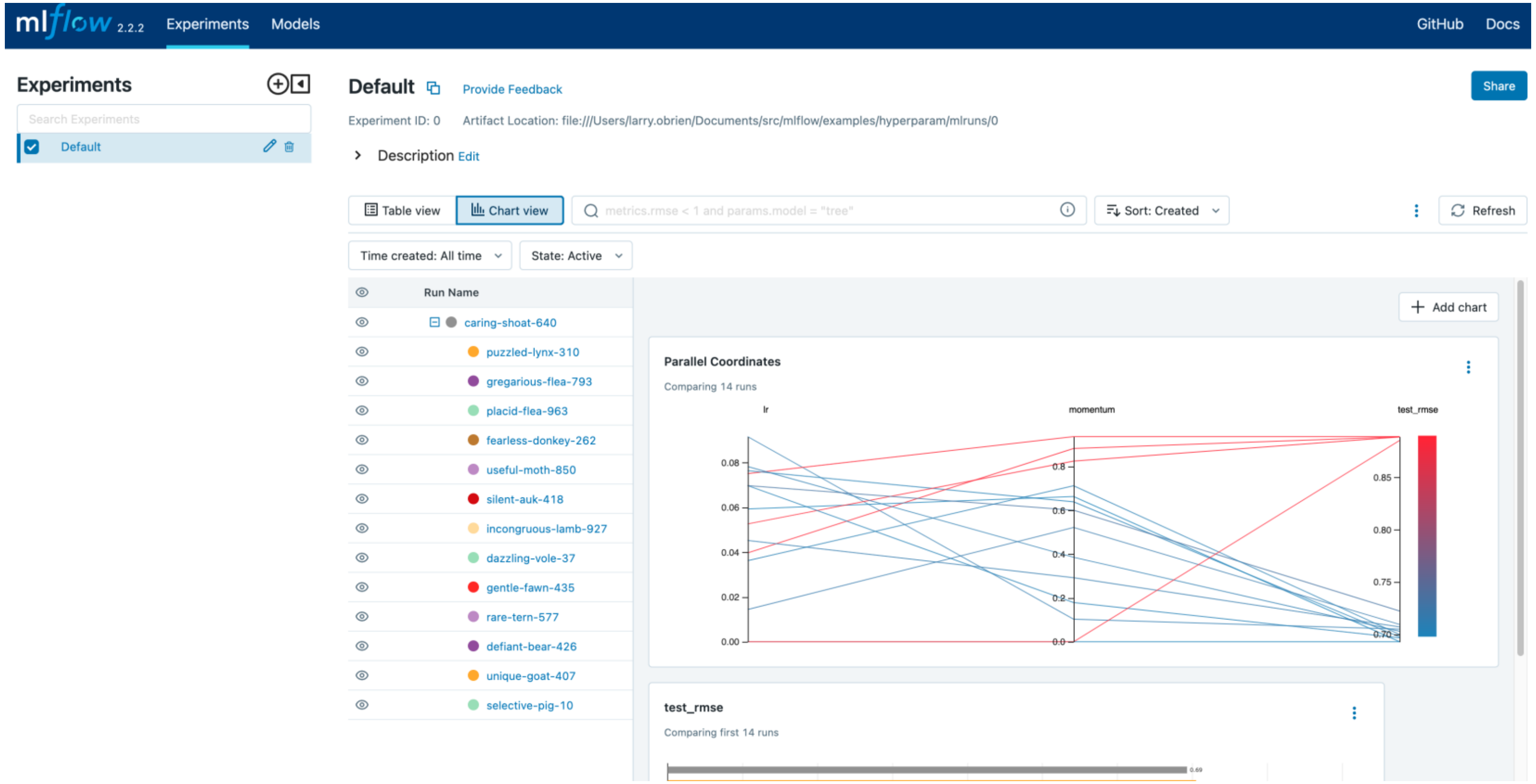
Experiment Tracking – MLflow Tracking

In MLflow, models that are logged during training are saved as artifacts, allowing for easy reuse and deployment. These logged models can later be **loaded** either from the local directory where they were saved or directly from a specific run's artifact URI. This makes it simple to reproduce results, compare models, or integrate them into production workflows without retraining.



Experiment Tracking – MLflow Tracking

Nested Runs in MLflow allow you to track experiments that have a **hierarchical structure**. This is useful when you want to track individual steps (e.g., preprocessing, training, evaluation) **within a single run**.



Experiment Tracking – MLflow Tracking

MLflow allows users to log **system metrics** including CPU stats, GPU stats, memory usage, network traffic, and disk usage during the execution of an MLflow run.

Turn on/off System Metrics Logging

There are three ways to enable or disable system metrics logging:

- Set the environment variable MLFLOW_ENABLE_SYSTEM_METRICS_LOGGING to false to turn off system metrics logging, or true to enable it for all MLflow runs.
- Use `mlflow.enable_system_metrics_logging()` to enable and `mlflow.disable_system_metrics_logging()` to disable system metrics logging for all MLflow runs.
- Use log_system_metrics parameter in `mlflow.start_run()` to control system metrics logging for the current MLflow run, i.e., `mlflow.start_run(log_system_metrics=True)` will enable system metrics logging.

Name	Value
system/cpu_utilization_percentage	32.7
system/disk_available_megabytes	850367.6
system/disk_usage_megabytes	9524
system/disk_usage_percentage	1.1
system/network_receive_megabytes	0
system/network_transmit_megabytes	0
system/system_memory_usage_megabytes	24180.7
system/system_memory_usage_percentage	35.2

Experiment Tracking – MLflow Tracking

Auto logging is a powerful feature that allows you to log metrics, parameters, and models without the need for explicit log statements but just a single `mlflow.autolog()` call at the top of your ML code.

This will enable MLflow to automatically log various information about your run, including:

- **Metrics** - MLflow pre-selects a set of metrics to log, based on what model and library you use
- **Parameters** - hyper params specified for the training, plus default values provided by the library if not explicitly set
- **Model Signature** - logs Model signature instance, which describes input and output schema of the model
- **Artifacts** - e.g. model checkpoints
- **Dataset** - dataset object used for training (if applicable), such as *tensorflow.data.Dataset*

MLflow Models

Experiment Tracking – MLflow Models

- Mlflow model logging enable model re-use
- MLflow provides several standard flavors that might be useful in your applications. Specifically, many of its deployment tools support these flavors, so you can export your own model in one of these flavors to benefit from all these tools:
  - `mlflow.sklearn.log_model(model, name)`
  - `mlflow.pytorch.log_model(model, name)`
  - `mlflow.tensorflow.log_model(model, name)`
  - `mlflow.onnx.log_model(model, name)`

Experiment Tracking – MLflow Models

MLflow simplifies model evaluation with automated tools that save time. Using `mlflow.evaluate(),` you can assess MLflow Models (with the pyfunc flavor) on various tasks including classification, regression, and language modeling. Results include performance metrics, plots, and explanations, all logged to MLflow Tracking.

Here are the different tasks and built-in metrics:

- **Regressor Models**
  - **Metrics:** example_count, mean_absolute_error, mean_squared_error, root_mean_squared_error, sum_on_target, mean_on_target, r2_score, max_error, mean_absolute_percentage_error.
- **Binary Classifiers**
  - **Metrics:** true_negatives, false_positives, false_negatives, true_positives, recall, precision, f1_score, accuracy_score, example_count, log_loss, roc_auc, precision_recall_auc.
  - **Artifacts:** Lift curve, precision-recall curve, ROC curve.
- **Multiclass Classifiers**
  - **Metrics:** accuracy_score, example_count, f1_score_micro, f1_score_macro, log_loss.
  - **Artifacts:** CSV with per-class metrics, merged precision-recall and ROC curves.
- **Question-Answering Models**
  - **Metrics:** exact_match, token_count, toxicity, flesch_kincaid_grade_level, ari_grade_level (requires additional packages).
  - **Artifacts:** JSON with inputs, outputs, targets (if provided), and row-level metrics.
- **Text-Summarization Models**
  - **Metrics:** token_count, ROUGE, toxicity, ari_grade_level, flesch_kincaid_grade_level (requires additional packages).
  - **Artifacts:** JSON with inputs, outputs, targets (if provided), and row-level metrics.

Experiment Tracking – MLflow Models

Serve the Models with Local REST server

- `mlflow models serve -m runs:/<RUN_ID>/model --port 9000`

The inference server provides 4 endpoints:

- /invocations: An inference endpoint that accepts POST requests with input data and returns predictions.
- /ping: Used for health checks.
- /health: Same as /ping
- /version: Returns the MLflow version.

MLflow Models Registry

Experiment Tracking – MLflow Models Registry

The MLflow Model Registry is useful because it provides a **centralized, organized way to manage the lifecycle of machine learning models**, especially in collaborative or production environments.

1. Version Control for Models

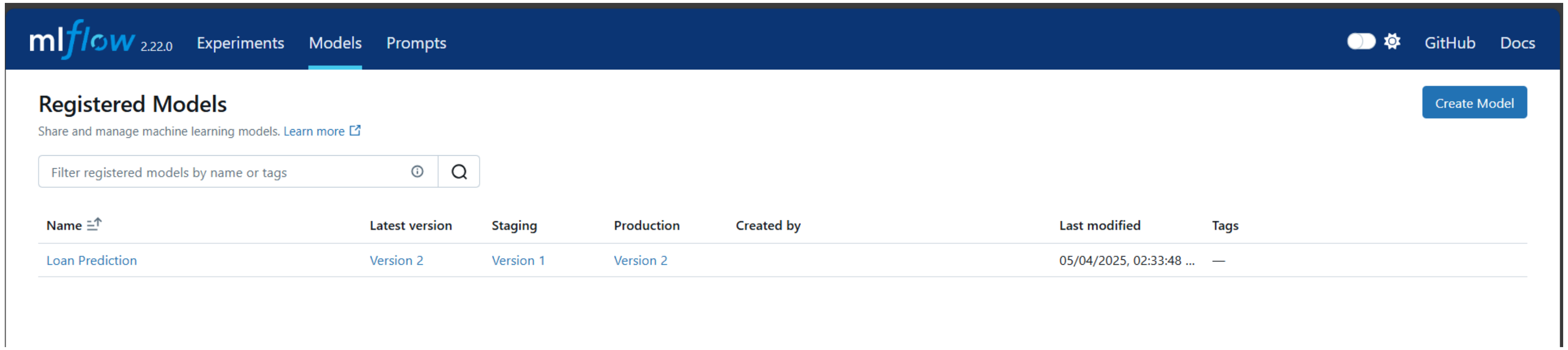
- Every model you register gets a version number.
- You can track different iterations of models trained with different data, parameters, or code.

2. Model Staging and Lifecycle Management

- Assign stages to models: None, Staging, Production, and Archived.
- This allows teams to:
  - Deploy only production models.
  - Test staging models before release.
  - Archive old models cleanly.

Typical Lifecycle:

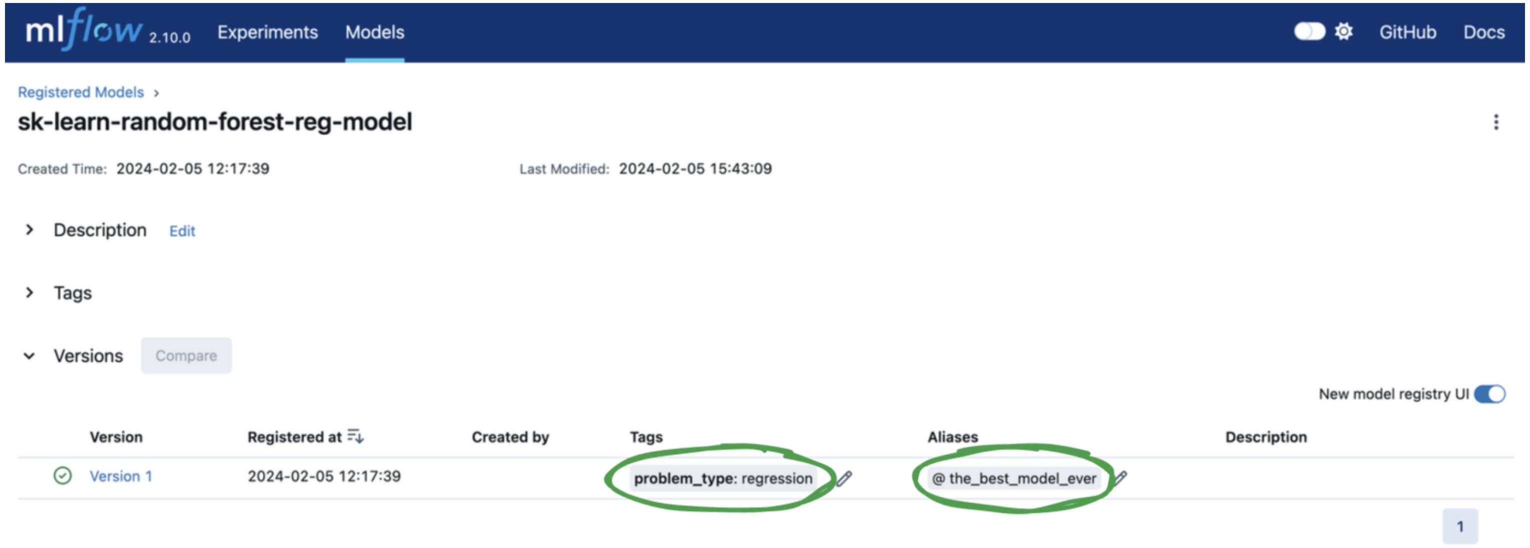
1. **None** (just registered) →
2. **Staging** (being tested) →
3. **Production** (live) →
4. **Archived** (deprecated or replaced)



Experiment Tracking – MLflow Models Registry

The MLflow Model Registry new interface provides some additional functionality that is relevant to model development and deployment:

- **Model Versioning** refers to logging different iterations of a model to facilitate comparison and serving. By default, models are versioned with a monotonically increasing ID, but you can also alias model versions.
- **Model Aliasing** allows you to assign mutable, named references to particular versions of a model, simplifying model deployment.
- **Model Tagging** allows users to label models with custom key-value pairs, facilitating documentation and categorization.
- **Model Annotations** are descriptive notes added to a model.



Experiment Tracking – MLflow Models Registry

To register a model, you can leverage the registered_model_name parameter in the `mlflow.sklearn.log_model()` or call `mlflow.register_model()` after logging the model. Generally, we suggest the former because it's more concise.

Model Serving:

```
mlflow models serve -m "models:/{Model Name}/{Stage Name}" -p port_number
```

Experiment Tracking – MLflow Models Registry

Model Loading:

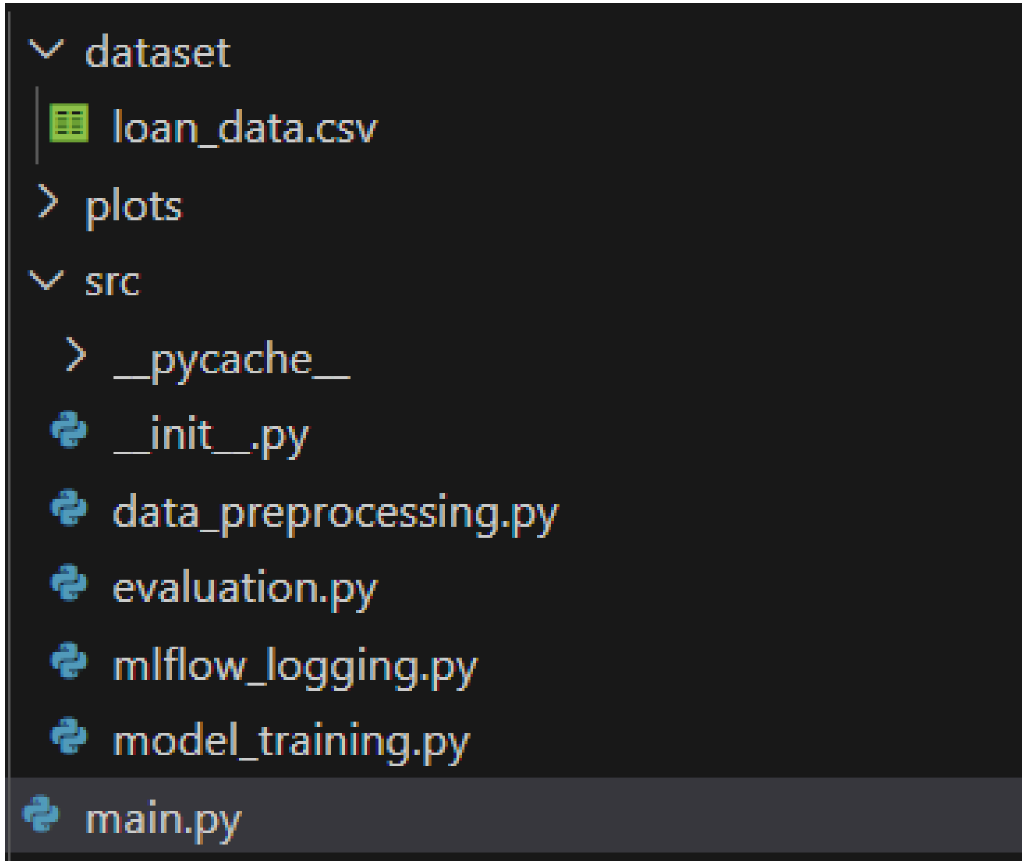
- **Load via Tracking Server:**
  `mlflow.sklearn.load_model(f"runs:/{mlflow_run_id}/{run_relative_path_to_model}")`
- **Load via Name and Version**
  `mlflow.sklearn.load_model(f"models:/{model_name}/{model_version}")`
- **Load via Model Version Alias**
  `mlflow.sklearn.load_model(f"models:/{model_name}@{model_version_alias}")`

Build – Getting out of Jupyter Notebook

Experiment Tracking – MLflow Tracking

- فقرة عشان اللي بعديا ميدعيش عليا

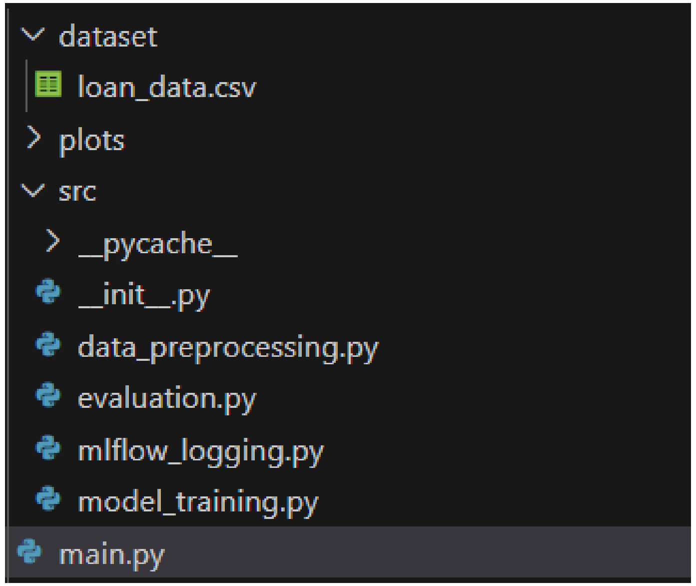
- What’s missing in this folder tree?



Experiment Tracking – MLflow Tracking

- فقرة عشان اللي بعديا ميدعيش عليا

- What’s missing in this folder tree?
  - **REQUIREMENTS FILE:** It’s crucial for maintaining the project's package versions and for ensuring consistent environments when rerunning the project after some time or sharing it with teammates. Without it, others may not know which dependencies are needed or compatible.
  - **README FILE:** This file is essential for understanding the purpose, setup, usage instructions, and overall structure of the project. It acts as a guide for new users or contributors to quickly get started and understand the context without diving into the codebase. A well-written README enhances collaboration and project maintainability.



MLflow Projects

Experiment Tracking – MLflow Projects

- MLflow Projects are just a convention for **organizing** and describing your code to let other data scientists, e.g., your teammates or yourself in the future, run it.
- Each project is simply a directory of files, or a Git repository, containing your code.
- You can run any project from a Git URI or from a local directory using the mlflow run command-line tool: `mlflow run .` , or the `mlflow.projects.run()` Python API.

Experiment Tracking – MLflow Projects

- You'll need to use an MLproject file to describe your project. This file is a YAML formatted text file. Main properties in the MLproject file are:
  - Name
    - A human-readable name for the project.
  - Entry Points
    - Commands that can be run within the project
    - It's the first instruction or reference that kicks off a process, application, or system when the YAML file is used.
  - Environment
    - The software environment that should be used to execute project entry points. This includes all library dependencies required by the project code.

```yaml
name: My Project

python_env: python_env.yaml
# or
# conda_env: my_env.yaml
# or
# docker_env:
#    image:  mlflow-docker-example

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}"
```

Experiment Tracking – MLflow Projects

- What's a YAML file?
  - A **YAML file** is a simple text file that uses **YAML format** (which stands for "YAML Ain't Markup Language") to organize data in a way that's **easy for humans to read** and **easy for machines to parse**.
  - Think of it like this:
    - It's **structured** like a list, table, or tree — but looks much cleaner than JSON or XML.
    - It's often used for **configuration**, **settings**, or **defining workflows**.
    - File extensions are usually .yaml or .yml.

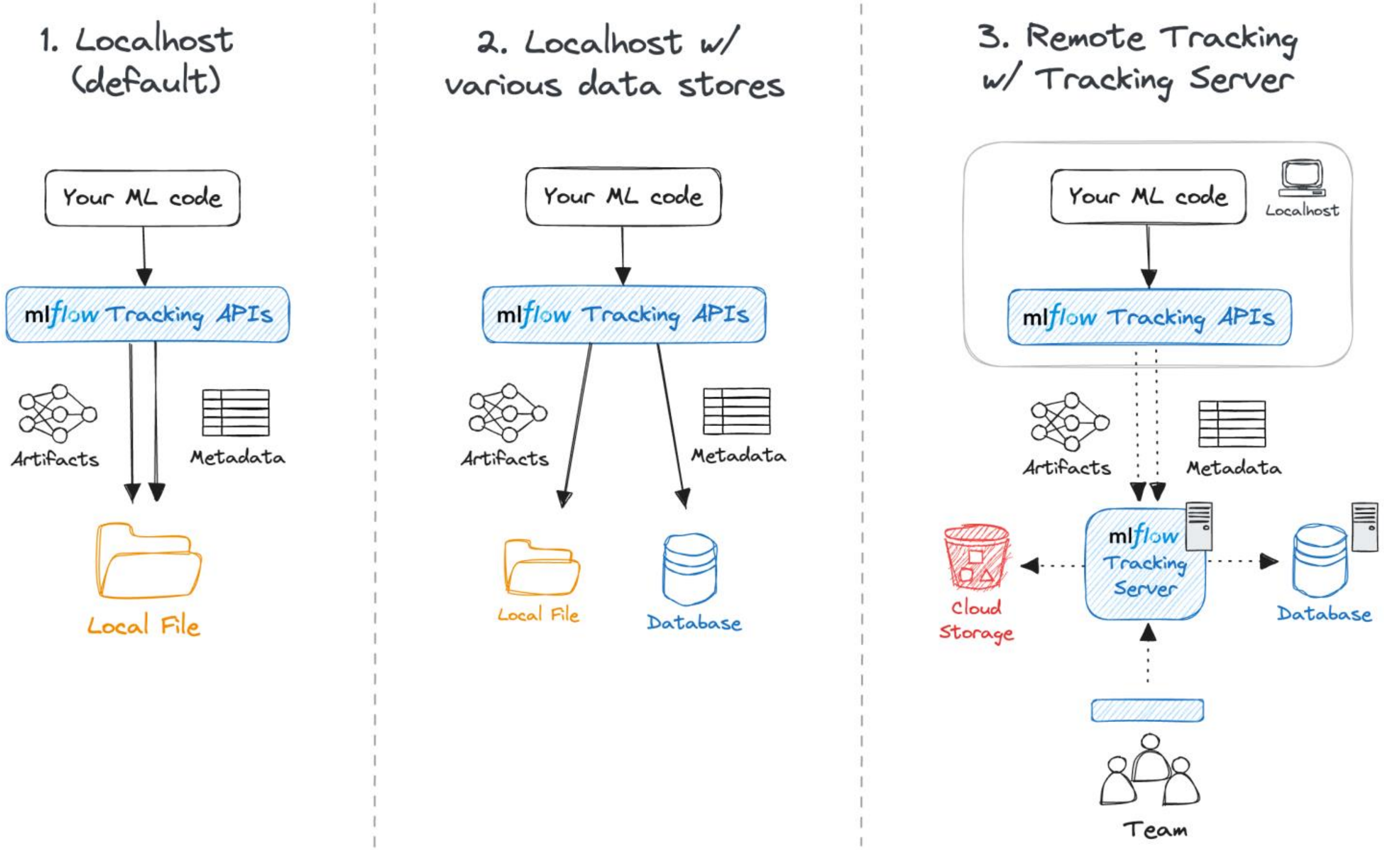
```yaml
name: Heba
is_student: false
skills:
  - Python
  - Machine Learning
  - Docker
address:
  city: Cairo
  country: Egypt

```

```json
{
  "name": "Heba",
  "is_student": false,
  "skills": ["Python", "Machine Learning", "Docker"],
  "address": {
    "city": "Cairo",
    "country": "Egypt"
  }
}
```

Experiment Tracking – MLflow

- Where was the information about the runs (metrics, parameters, artifacts) stored?
- How will local storage of experiments impact a project developed by multiple ML scientists and engineers?



Experiment Tracking – MLflow

	1. Localhost (default)	2. Local Tracking with Local Database	3. Remote Tracking with MLflow Tracking Server
Scenario	Solo development	Solo development	Team development
Use Case	By default, MLflow records metadata and artifacts for each run to a local directory, `mlruns`. This is the simplest way to get started with MLflow Tracking, without setting up any external server, database, and storage.	The MLflow client can interface with a SQLAlchemy-compatible database (e.g., SQLite, PostgreSQL, MySQL) for the backend. Saving metadata to a database allows you cleaner management of your experiment data while skipping the effort of setting up a server.	MLflow Tracking Server can be configured with an artifacts HTTP proxy, passing artifact requests through the tracking server to store and retrieve artifacts without having to interact with underlying object store services. This is particularly useful for team development scenarios where you want to store artifacts and experiment metadata in a shared location with proper access control.

Experiment Tracking

- For Vialytics use case:

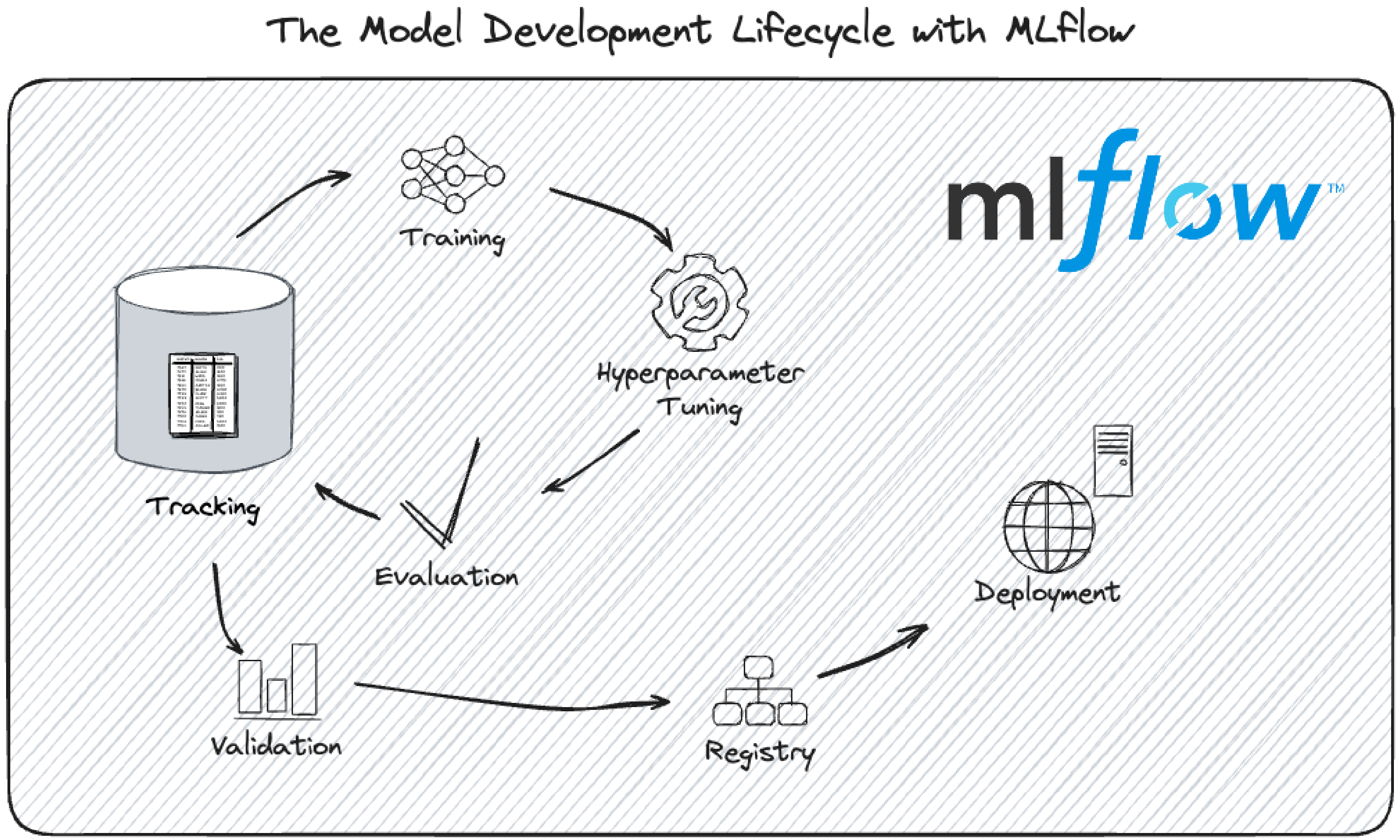
Experiment Tracking

- For Vialytics use case, which model to use among those?

Metric	YOLOv5	Mask RCNN
Accuracy (mAP@50)	58%	64%
Latency Time	25 ms	152 ms

Experiment Tracking – MLflow

- Wrap up



LAB

Experiment Tracking - Lab

- Dataset used is [Bank Customer Churn Prediction]
  - التنبؤ بانخفاض عدد عملاء البنوك
- It is the dataset of a U.S. bank customer for determining whether this particular customer will leave bank or not.
- What you'll do is:
  - Fork this [Repo]
  - Create a new conda/venv env (python version 3.12), named churn_prediction & install the requirements
  - Download the CSV dataset file into data folder
  - Add the Mlflow logging in the src/train.py file
  - Run the experiment by running the following command in the MLOps-Course-Labs directory
    - Python src/train.py
  - Try another model or any other changes you think might improve the performance
    - Run **at least two other** different experiments
  - Filter your experiments and register two chosen models one in staging and the other in production, you'll need to justify your choice :D

  - **BONUS**: Complete the readme file in the research branch.
  - **EXTRA BONUS**: Make the code cleaner, e.g., break the code into modules, add logging, .. etc

Any Questions?

Resources

- Gitflow Workflow
- Github Flow vs Git Flow
- GitHub Flow
- MLflow Documentation
- Tutorial on Tracking using MLflow

Attribution

- Testing icons created by juicy_fish - Flaticon
- Work illustrations by Storyset
- https://www.flaticon.com/free-icon/crack_16447908?term=eggshells&page=1&position=5&origin=search
- https://www.flaticon.com/free-icon/experiment_4717980?term=lab&page=1&position=46&origin=search&related_id=4717980
- https://www.flaticon.com/free-icon/take-a-break_11661572?term=break&page=1&position=12&origin=search&related_id=11661572
- https://www.flaticon.com/free-icon/amnesia_3997692?term=memory&related_id=3997692
- Work illustrations by Storyset