



# Comparison checklist

---

Neda Sultova

- General
- Version management and reproducibility
- Experiment tracking and comparison
- Pipeline execution and code invasiveness
- Scaling and backend
- Interaction with HPC
- Storage
- Logging and debugging
- Visualization
- Collaboration
- Compatibility
- Model serving
- Open source
- Costs
- Application to real world example

# Short definitions

## Metaflow

[Metaflow](#) is a python library, originally developed at Netflix, that helps building and managing data science projects.

## MLflow

[MLflow](#) is an open source platform to manage machine learning life-cycles. The platform offers four distinct components, which can be used in either in stand-alone mode or together.

## DVC

[DVC](#) is an open-source version control system for ML projects which is build around existing engineering toolsets and workflows

# General

## Comparison Factors

### General

- design decisions
- emphasis
- components

## Metaflow

Build around [dataflow paradigm](#)

High emphasis on seamless scalability and runtime agnostic (code can run on a laptop in parallel over multiple processes or in the cloud over multiple batch jobs)

Strong [testing philosophy](#)  
Object Hierarchy:

Metaflow -> Flow -> Run -> Step  
-> Task -> Data Artifact

**Flows:** graph of operations

**Steps:** describe operations within the flow

**Run:** execution of flows

Workflow implemented via [namespaces](#)  
Tightly coupled with [Amazon Web Services](#)

Designed around failure cases

## MLflow

Organized into four components:

- [Tracking](#)
- [Projects](#)
- [Models](#)
- [Model Registry](#)

Components can be used on their own

Emphasis on maximum flexibility and customization

Out-of-the box with [Microsoft Azure](#)

Deployable on [Amazon Sagemaker](#) with one API call

## DVC

Runs on top of any Git repository

Compatible with GitHub

Closely interacting with git (not necessary but recommended)

High emphasis on handling large datasets

Keep everything simple and accessible

Steps are connected into a [DAG](#)

Build around [pipelines and stages](#)

Minimal code invasiveness

# Version management and reproducibility

## Comparison Factors

Version management + reproducibility

- code
- model
- data

## Metaflow

Built-in versioning via [tags and namespaces](#)

Production namespace and user namespace are separated (a bit like working on different branches)

Creates snapshots of runs

Environment is managed via [conda](#) (@conda decorator is implemented)

## MLflow

Handled via [MLflow projects](#)  
API and command-line tools for running projects, can be chained together to workflows

- build-in support of [conda](#)
- build-in support for [docker-containers](#)

**Project:** a directory with code/a git repository  
- uses a descriptor file/ convention to specify dependencies/runs.

Code can be published on GitHub in the MLflow Project format

MLflow Model Registry manages model lifecycles. (especially build for large collaborations and many models from many teams involved)

## DVC

Version management works together with git

- like a distributed version control system
- lock-free, local branching, versioning

Can be used together with [CML](#) for [further improving CI/CD](#)

Dvc manages the remote storage of data and creates files which then can be tracked via git

Due to its structure one can use different virtual environments as with other code projects, but no special integration (found so far)

# Experiment tracking and comparison

## Comparison Factors

Experiment tracking + comparison

parameter handling  
metadata  
data artifacts

## Metaflow

[client API](#) is used to inspect results of past runs and some other parameters

Can be used within code and notebooks

[data artifacts](#): container object, produced by runs. Contains actual value + metadata

## MLflow

Realized with MLflow Tracking API+UI

Can be used within code and notebooks

Automatically keeps track of parameters, metrics, code, and models from each experiment

## DVC

Realized with [DVC Experiments](#)

Designed for exploration of many different configurations and scenarios

Organize experiments and keep only the best ones

Tracking, scheduling, comparison integrated

Run experiments via ``dvc exp run``

Known cases where experiment data has been fed into external tools ([weights&biases](#))

# Pipeline execution and code invasiveness

## Comparison Factors

Pipeline execution + code invasiveness

## Metaflow

Integration:

- form the ML-Process code into [Flows](#)
- a flow is implemented by subclassing FlowSpec and implementing steps as methods

Run:

- execution of user-defined flows
- also executable via CLI

## MLflow

Integration:

- mlflow via the [Python API](#) into existing code

Run :

- via [CLI tool](#), or via [mlflow.projects.run\(\)](#) Python API

## DVC

Integration:

- as underlying mechanism, (no code needs to be altered), or via [Python API](#)
- (it can be helpful to structure code around input/output paradigms)

Run:

- via execution of ``dvc run`` and/or ``dvc repro``

# Scaling and backend

## Comparison Factors

### Scaling + Backend

## Metaflow

Integrates with Batch from AWS  
- directly accessible via [@ressources](#) decorator

Languages such as C++ can be called from within a step for direct performance tuning

Can work with [Amazon Sagemaker](#)

Works well with [meson](#) (workflow orchestration + scheduling)

Also mention of [scaling up on larger machines](#) but no further specs

The [foreach](#) branch offers a mechanism for independent/parallel executions

## MLflow

Integrates with [Databricks](#) as [Managed MLflow](#), [Kubernetes](#), [Spark](#)

Custom backends and needs can be implemented and integrated via [mlflow community plugins](#)

## DVC

Integrates with [Spark](#), [Hive](#)

Heavy cluster jobs can be decomposed into smaller DVC pipeline steps (parallel execution)

# Interaction with HPC cluster

## Comparison Factors

### Interaction with HPC cluster

- installation
- requirements
- challenges

## Metaflow

To be tested

## MLflow

To be tested

## DVC

Work in progress



# Storage

## Comparison Factors

### Storage

## Metaflow

Local file path

[Amazon S3](#)

## MLflow

[MLflow Tracking Server](#) offers two types of storage

- database-backed store: experiment and run metadata as well as params, metrics, and tags for runs
- file store: large data, log of artifact output (eg models).

supports:

- local file paths, NFS, SFTP
- Amazon S3, Azure Blob Storage, Google Cloud Storage

## DVC

Local file path, SH/SFTP, HDFS, HTTP, NFS, disc to store data

Amazon S3, Microsoft Azure Blob Storage, Google Drive, Google Cloud Storage, Aliyun OSS ..

# Logging and debugging

## Comparison Factors

### Logging and debugging

what  
where  
how

## Metaflow

Persists all runs + resulting data artifacts

Results can be accessed via unique run ID

Retrieval generally via namespaces + tags via [Client API](#), see [debugging](#)

Resume execution of a past run at a failed step. Iterate quickly on step code

Python stack trace available

Plugin for VScode

..some other IDE's also work (PyCharm)

Explicit [Metaflow Test Suite](#) for test-driven development

## MLflow

### [MLflow Tracking](#)

Automatic logging via [mlflow.autolog\(\)](#)

Library-specific autolog calls available (eg pytorch, fastai, scikit-learn..)

Runs can be annotated via [tags](#)

No specific debugging information found

## DVC

Realized via [Checkpoints](#) (checkpoints can be registered during your code or script runtime (similar to a logger) to track successive steps in a longer experiment)

[Automatic log of stage runs \(run cache\)](#)

VScode plugin currently in development

# Visualization

## Comparison Factors

Visualization of results

Graphical user interface

## Metaflow

[flow graphs](#) are visualized

Didn't find anything comparable for specific parameters or trainings

## MLflow

[MLflow Tracking UI](#)

Good GUI to compare and select models

Experiment-based run listing and comparison

Searching for runs by parameter or metric value

Visualizing run metrics

Downloading run results

## DVC

[DVC Studio](#)

As it just launched (as of this writing) yet to be tested

Seems to offer similar functionality to MLflows GUI

Plotting of metrics and results can be handled via [`dvc plots`](#)

# Collaboration

## Comparison Factors

### Collaboration with others

## Metaflow

Collaboration is enabled and achieved by using [tags and namespaces](#)

People create different flows in their user namespaces, results can be accessed by other members

## MLflow

MLflow supports git, collaborative work on [MLflow Projects workflows](#) possible

Collaboration via [Model Registry](#) possible (suitable for large scale setups)

Community efforts to add collaboration possibilities: combining MLflow with [neptune](#) -> WIP

## DVC

Achievable by utilizing [remote storage capabilities](#), also similar to collaboration in github/gitlab

[DVC Studio](#) also seems to offer some collaboration specific functionalities

# Compatibility

## Comparison Factors

### Compatibility

- Pytorch
- Keras
- other languages

## Metaflow

Different python libraries are supported

Languages such as C++ can be called within a step

Jupyter notebooks are supported

## MLflow

Different python libraries are supported

Languages such as C++ can be called within a step

## DVC

Pipelines are defined around input/output, thus language-agnostic, support for Python, R, Julia, Scala

Spark, custom binary, TensorFlow, PyTorch, etc

Jupyter notebooks are supported

# Model serving

## Comparison Factors

### Model serving

## Metaflow

[pickle](#) for object serialization/deserialization as default

Others possible

## MLflow

### [MLflow Models](#)

Served via concept of [flavors](#), thus models can be processed by different downstream tools

Variety of tools to deploy models (eg. a TensorFlow model can be loaded as a TensorFlow DAG, or as a Python function to apply to input data.)

Tools to deploy many common model types to diverse platforms: e.g model supporting the "Python function" flavor can be deployed to a Docker-based REST server or to cloud platforms such as Azure ML and AWS

## DVC

No concrete defaults as dvc is mainly concerned with the data storage/retrieval part

Get models via ``dvc get`` or [dvc.api.open\(\)](#)

# Open source

Comparison Factors	Metaflow	MLflow	DVC
Open Source version available?	Yes	Yes	Yes

# Costs

Comparison Factors	Metaflow	MLflow	DVC
Costs	<p>No enterprise plan or anything, usage of metaflow itself is free</p> <p>You pay for the AWS service in case of usage</p>	<p>No enterprise plan or anything, usage of MLflow itself is free</p> <p>You pay for additional services in case of usage (<a href="#">Managed MLflow</a>, Cloud storage..)</p>	<p>Usage of DVC itself is free</p> <p>DVC studio free up to 5 team members</p>



# Real world application

## Comparison Factors

Tested with real-world example?

## Metaflow

To be tested

## MLflow

To be tested

## DVC

Work in progress  
(see Interaction with HPC)