Published in Geek Culture

Neda Sultova Follow

Aug 3, 2021 · 11 min read · Listen

Comparing Metaflow, MLFlow and DVC

Space Adventures #58 (Charlton, 1964)

understanding the problem, defining criteria to the actual comparison process, the comparison itself and my conclusions. You can find the repository I created along the way and the center piece, the comparison table, here and here.
Feel free to follow along or use them as a basis for your own work. This blogpost is structured as follows:

- Defining our use case
- The machine learning lifecycle
- Comparison checkpoints
- Examination setup
- Recap
- Conclusion
- Outlook
- Glossary

Before examining or testing any tooling, it was important to gain an overview about the broader concept of machine learning engineering itself. (See Rules of ML, CD4ML and Googles Hidden Technical Debt in Machine Learning Systems as good first impressions about what this area is about and why it is important).

After roughly situating our use case within the emerging MLOps landscape, I converged towards examining three tools: Metaflow, MLflow and DVC.

> *Metaflow is a python library, originally developed at Netflix that helps building and managing data science projects.*
>
> *MLflow is an open source platform to manage machine learning life-cycles. The platform offers*




Defining our use case

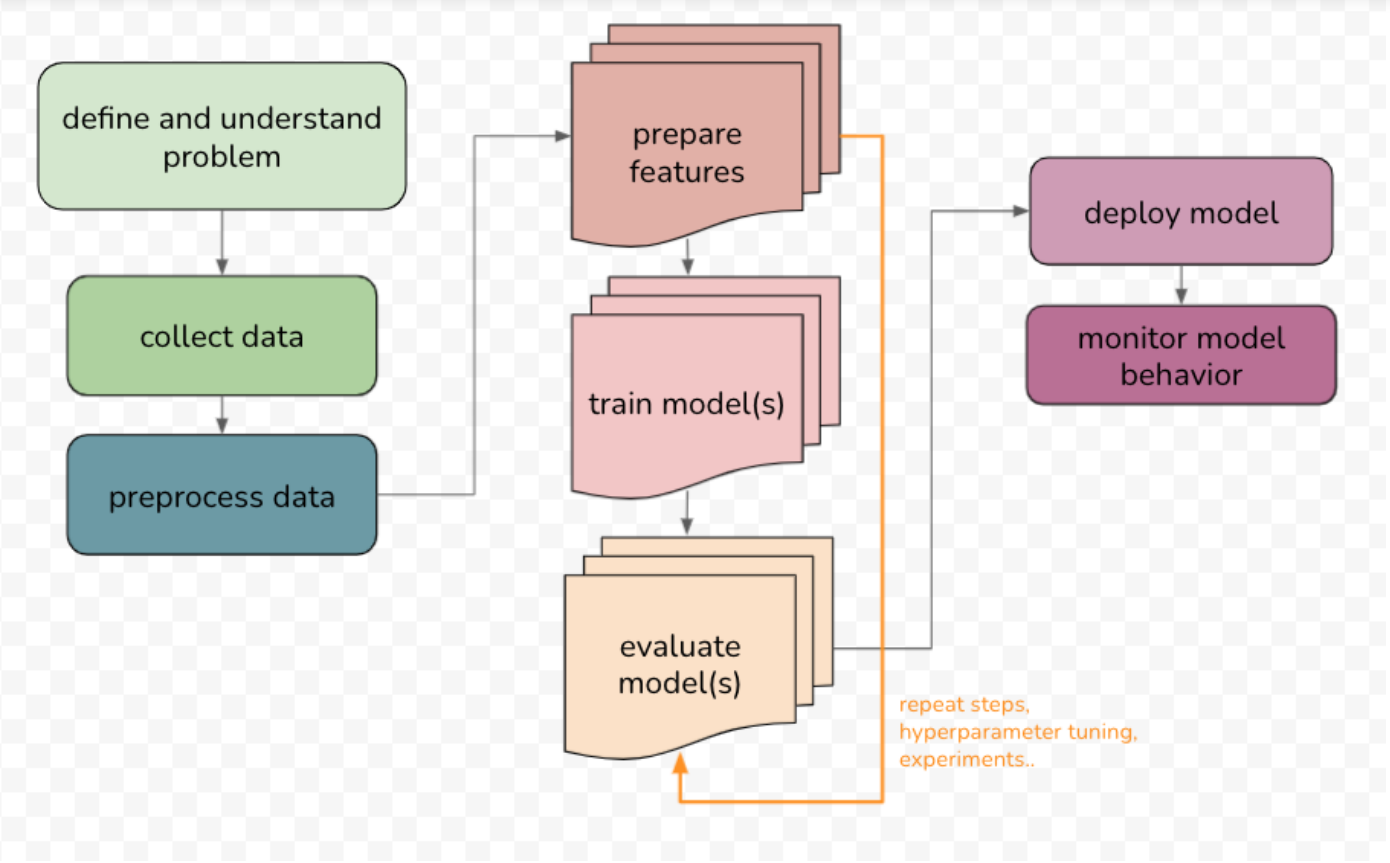
As a research facility we have various aspects to consider when we choose new tools. Our use case revolves around large scientific datasets, custom infrastructure (own data centres, HPC) and a particular need for reproducibility.

Based on this, the key-points that must be met are following:

- **Reproducibility:** If scientific results were obtained by using ML along the way, this part must also be comprehensible for independent researchers, thus reproducibility of code, data, parameters and models is crucial.
- **Workflow integration:** Another thing to keep in mind is that our potential user base will include scientists, whose main job is not to know their way around complicated tooling or advanced coding — thus, the easier a tool integrates into already established workflows (e.g using git, python scripts..) the better.
- **Exchangeable backend:** We have our own data centres and cloud infrastructure, thus the tool must be able to work with custom backends
- **Framework agnostic:** As preferred libraries can change rapidly and even preferred programming languages a tool shouldn't care about which libraries and flavours were used for the actual machine learning code.
- **Open source:** We think it should be preferred and also usually it's much easier to adapt for custom needs
- **Cost factor:** As a public institution we have to keep our budget in mind and prefer non-enterprise solutions

The machine learning lifecycle

There are various ways to break down a machine learning process, depending on one's priorities. The image below depicts the machine learning lifecycle as seen from our perspective:



This graphic is separated into various steps, which are connected by arrows denoting the execution order. The steps are further grouped into blocks which are inspired by commonly found coding structures.

First of all, we need to understand and define our problem. What is it about? What do we expect to learn when applying machine learning? Is this a problem for machine learning at all? After answering these questions, the next step is about data collection (usually done by the scientists) and preprocessing the collected data (usually done by our team). Sometimes, depending on data quality, the preprocessing step must be repeated several times. (But that's for another story ^^)

The second block shows the steps — these usually are executed multiple times for experimenting and tuning purposes.

Once a suitable model has been determined, the model needs to be made available for usage and monitored to detect skews over time.

Comparison checkpoints

- 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

You can find the comparison table here

Examination setup

Many of the insights were gained through extensive reading of the documentation and execution of official tutorials but in order to get a feeling for each tool, one needs hands-on experience.

In order to be able to compare things as usability between tools, a common base is needed. Thus the idea was to create a repository containing an adapted version of the same machine learning example, as every tool integrates differently into the code, furthermore this very




could and should be extended according to the checkpoints)

Here is a rough overview of basic testing steps for each tool:

```
Pipeline:
- integrate $tool into workflow
- run ml-pipeline with $tool

Parameters:
Test logging, altering and retrieving with:
- parameter 1 (val0, val1, val2)
- parameter 2 (val0, val1, val2)

Metrics:
Test logging, exchanging and retrieving with:
- metric 1 .. metric n

Models:
Test exchangeability of models
(pending)
- model 1 (type, parameters, metrics)
- model 2 (type, parameters, metrics)

Dataset:
Storing, altering, versioning and retrieving
- small dataset (PoC)
- changed dataset
- large dataset (pending)

Storage:
- local
- remote (pending)

HPC:
- interaction with HPC environment (pending)
```

Additional information about the example itself can be found in Content/Example.md.

The whole examination setup + example code + additional information can be found here

Below follows a quick recap of each tool, with focus on aspects important to us.




Metaflow is a python library, originally developed at Netflix, that helps building and managing data science projects. The library is designed around the dataflow paradigm. Code and workflow are managed through Flows. Working with Flows enforces a strong object oriented approach towards code.

When a Flow is applied to production for the first time, Metaflow creates a new, isolated production namespace. Tags and namespaces are another powerful concept which is used for various tasks, ranging from managing states to version control and collaboration with others.

Metaflow aims to structure code as a graph, build from steps. Steps are indivisible units of execution and can be used to determine code granularity. Steps also mark [checkpoints](a checkpoint) and thus are used for tracking of various states and their according parameters.

Parallel and dynamic code execution can be handled via foreach branches. Space requirements, e.g for checkpoints and data are handled by close integration with Amazon Web services. While theoretically Metaflow can be used with different backends as for now only AWS is supported.

MLflow

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

- MLflow Tracking
- MLflow Projects
- MLflow Models
- Model Registry

There is an additional sub-component which is found in the DOCs and important for our use case:




everything accessible via the UI can also be accessed programmatically. The component itself is organized around the concept of *runs*, which can be seen as execution steps of machine learning code — *runs* are also recorded and can be seen similar to Metaflows *steps*.

An MLflow Project is a format for packaging code in a reusable and reproducible way. The component further includes an API and command line tools for executing projects and chaining several projects into workflows. Workflows can also be used for parallel execution or experimenting with parameters, offering some functionalities like DVCs experiments feature.

The MLflow Model component is a standard format for packaging machine learning models in order to use them within different downstream tools, e.g real-time serving through a REST API or batch inference on Apache Spark. Models can be exported in different *flavors* for further usage.

The MLflow Model Registry component is a centralized model store including a set of APIs and an UI for collaboratively managing the full lifecycle of an MLflow Model. It seems to act as a component for managing the other three in larger setups.

The MLflow Python API enables writing plugins that integrate with different ML frameworks and backends. Mlflow plugins provide a mechanism for customizing the behavior of the MLflow Python client and integrating third-party tools. One important use case for us is the possible integration of custom storage solutions.

DVC

DVC is an open-source version control system for ML projects which is built around existing engineering toolsets and workflows (Git, CI/CD, etc.). DVC can be divided into following components:

- Versioning + access
- Pipelines
- Experiments




Data and Model versioning is the component responsible for handling and tracking large files, datasets, and machine learning models. This is achieved by commands that are executed along with `git` ("DVC == Git for data"). Information about added data is stored in `.dvc files` .This metadata file is a placeholder for the original data and can be versioned like source code with Git. The data itself can be stored remotely and access handled via `dvc push` and `dvc pull` .

Code and workflow are organized within Pipelines. The idea behind pipelines is to structure code around data *input* and *output* and making execution steps reproducible — similar to other workflow management tools (e.g Snakemake).

With `dvc run` one can create *stages*, which form the steps of a *pipeline*. Python scripts can easily be added to a stage.

DVC offers the `dvc experiments` feature, which addresses the need of hyperparameter tuning and model testing. It uses the same pipeline principle but offers additional functionalities like dry-runs, queueing and comparison between different experiment runs. The best ones can be applied to the actual workflow whereas others can be discarded, thus not polluting the repository. DVC output seems to be able to be fed into other tools like Weights&Biases.

Visualizing results can be achieved via the various diff commands or DVC Studio which offers similar functionality to MLflows tracking UI.

Conclusion

I included conclusion and outlook for those interested in the process and my opinion on that matter

Metaflow is built with scaling and collaboration in mind which is visible from both the design concepts (heavy utilization of namespaces) and the docs. Code is structured as graph flow, features are controlled via decorators, iterators are used for chaining everything together (instead of the classical return statements). The approach is very powerful and elegant but also requires sufficient computer science familiarity in order to be able to make

management tool. Some people have tried to use Metaflow and Mlflow in combination as a workaround attempt and there are also community efforts to add different backends. As for now it's not foreseeable when there will be official and easy-to-use support.

MLflow offers the most diverse functionality from the inspected tools. It includes a graphical UI, different APIs for integrating with different backends or serving models, a machine learning lifecycle management component and various other features. The documentation is very detailed which is great for DevOps and Admins but might be overwhelming for some of our potential users. For our use-case the MLflow Tracking component together with its UI is a huge plus. On the other side the vastness of the framework and non-trivial collaboration options are a serious disadvantage. A high learning curve and initial code investment (eg. writing plugins for backend) are to be expected.

DVC is smaller and built upon existing and widespread toolsets, especially around Git. In comparison to the other two tools DVC is mostly operated by a set of CLI tools or DVC studio (despite it also offers an API for direct code integration if one wishes to do so). It seems like the least code-invasive tool, there are no substantial code adjustments needed in order to integrate the tool, which is a huge plus for our potential user-base. The documentation is straightforward and offers a lower entry-point in terms of required computer-science background.

Generally, Metaflow seems like a great choice for large in-production use-cases with large teams but might be too difficult to apply for our intended user-base and custom environment. MLflow should be a serious consideration if the Institute is considering standardizing the whole ML workflow and handling it with the same importance as other infrastructure. It probably can be used by single researchers or small teams, if they focus solely on MLflow Tracking to track experiments locally on their machine and organize code into projects for future reuse. MLflow Tracking reads and writes files to the local file system by default, so there is no need to deploy a server. DVC is more narrow than the other tools (eg. scaling must be handled by backends) and has the best balance between advantages and disadvantages for our use-case. It covers needs such as experiment tracking, comparison and data-versioning and builds around well elaborated tools like git which makes it easier accessible and usable within already existing workflows. Also it's the least

The current recommendation based on this work would be to look closer towards either a boiled-down usage of MLflow or going with DVC. In both cases the usage of custom backend, remote storage and a more thorough examination of the UI is yet to be estimated (hopefully coming soon, experiments with a real scientific use case are on their way.)

In terms of community support, all three tools offer some ways to interact and get help.

Use cases and requirements will change over time and with gaining experience thus this blogpost should be seen as first attempt to find my way around this vast topic.

I hope you enjoyed your read :) Constructive criticism and suggestions are always welcome!

Update: I was told by the DVC community that running MLflow and DVC together is also an option.(MLFlow for visualization, DVC for versioning). This approach could help mitigating the shortcomings of MLFlow regarding collaboration whilst using its powerful UI.

Glossary

(To be taken with a grain of salt as always)

(python) library: A set of useful functions that eliminate the need for writing code from scratch.

(open source) platform: A group of technologies that are used as a base upon which other applications, processes or technologies are developed.

Framework: A platform for developing software applications. A framework is similar to an API , though technically a framework includes an API. As the name suggests, a framework serves as a foundation for programming, while an API provides access to the elements supported by the framework. A framework may also include code libraries, a compiler and other programs used in the software development process.

API: application programming interface — a set of commands, functions, protocols, and

(G)UI: A graphical user interface through which users interact with electronic devices and software via visual indicator representations.

HPC: High performance computing

CLI: Command line interface — a text-based interface that is used to operate software and operating systems