An In-depth Comparison of Experiment Tracking Tools for Machine Learning Applications

Tim Budras*, Maximilian Blanck†, Tilman Berger†, and Andreas Schmidt*‡,
* Department of Computer Science and Business Information Systems,
Karlsruhe University of Applied Sciences
Karlsruhe, Germany
Email: {buti1021, andreas.schmidt}@h-ka.de
† inovex GmbH, Karlsruhe, Germany
Email: {mblanck, tberger}@inovex.de
‡ Institute for Automation and Applied Computer Science
Karlsruhe Institute of Technology
Karlsruhe, Germany
Email: andreas.schmidt@kit.edu

***Abstract*—As the machine learning market is growing strongly and machine learning is increasingly being used productively, new challenges for developers and operators arise that haven't been existing in traditional software development. One of these challenges is the versioning and reproducibility of models. To help solve this challenge *experiment tracking tools* exist, which keep track of the experimental development process of machine learning models. This paper describes the process of bringing a machine learning model to production and emphasizes its experimental nature and the challenges arising with it. Following the definition of a set of requirements for experiment tracking tools, 20 tools found in a market research are presented. Four of those tools are analysed in-depth, showing that differences between tools exist especially for advanced requirements. This paper also includes the progress the tools have made within the last year.**

***Index Terms*—Machine Learning; Experiment Tracking; Development Environment; MLOps**

I. Introduction

This paper is an extended version of a conference paper [1], published in 2022 at the Fourteenth International Conference on Advances in Databases, Knowledge, and Data (DBKDA-2022) conference in Venice/Italy. In this extended paper, we go into more detail about the various tools that we were able to consider in the previously mentioned conference paper. We have also examined the tools in terms of their current enhancements.

The machine learning market is growing strongly. According to MarketsandMarkets [2], it is "expected to grow from USD 1.03 billion in 2016 to USD 8.81 billion by 2022". As a result of this growth, tools have been developed in recent years to help develop machine learning models and put them into production. However, due to the fact that the use of machine learning in productive software is relatively new, tools and conventions are less settled and less commonly applied than in traditional software development.

Warden [3] uses the term "machine-learning-reproducibility-crisis" to describe that the tools to meet these needs are often not deployed or used in practice. With regard to tracking data, parameters, models and results, numerous products with different focuses and strengths have been developed. Tools that focus on saving information around the model training and development process are often referred to as *experiment tracking tools*. But as stated in a Kaggle survey [4], in a large amount of scenarios these relatively new tools remain unused and tracking is either done manually or not done at all.

But without experiment tracking, the information under which circumstances an AI model was created is missing. It is therefore a black box model, which contradicts the High Level Expert Group on Artificial Intelligence (HLEG AI) [5] demand for *transparency*. An important criterion according to the HLEG demand for transparency is *explainability*: decisions made by an AI system must be understandable and comprehensible to humans. This requires information about the underlying datasets, the algorithms, parameters and data processing pipelines used, and the results obtained, which then serve as the basis for selecting specific algorithms/parameter sets.

The information gathered in this way will further enable repeatability of the experiments for both the current and future development teams, and also for other working groups dealing with the same or similar issues. Experiments by Alahmari et al. [6] show that the tracking information collected is sometimes not sufficient for the repeatability of an experiment. They demonstrated that running the same experiment several times can lead to different results. Reasons for this are, for example, a random selection of the training and test data, different libraries used or also hardware. In [7], for example, it was shown from Nagarjan et al. that when switching from CPU to GPU, different but deterministic results were obtained for the respective processor unit. More information about reproducibility and traceability to achive trustworthy AI can be found in [8], [9].

The paper is structured as follows: In Section II we explain the *machine learning lifecycle* and what artifacts, i.e., code, data, environment, parameter settings need to be tracked in the context of an experiment. Based on these findings we present in Section III the general architecture for experiment tracking tools and formulate the most important requirements. In Section IV four tools are presented and compared in detail. The paper is finished with a conclusion and outlook to further research directions in Section V.

II. Background

In this section, a set of basic insights required for understanding tracking tools in the field of machine learning will be presented as well as information about research related to experiment tracking tools.

A. The Machine Learning Lifecycle

The different phases and steps around the productive use of a machine learning model have been described by different authors using different terms. One of these terms is the *machine learning lifecycle*. Garcia et al. [10] describe the machine learning lifecycle as a three-phase process as shown in Figure 1.

The first phase is the *pipeline development*. During this iterative phase, the data preprocessing, exploration and visualization is done, model designs are chosen and models get trained with different configurations and hyperparameters. The authors emphasize that the important achievement of the first phase is not the model, but the pipeline that can be reused to create a model from a dataset. This pipeline can be used later in the second phase *training* (Figure 1, middle), to train and validate the model used for inference. The last phase (Figure 1, right) is called *inference*. Here, the prediction service (which includes the data preprocessing as well as the model used for inference) returns a prediction for a given user input. This service provides information about the predictions made, which can be used for subsequent training. The authors mention that the different stages are often managed by different teams.

Amershi et al. introduce a similar process, the *machine learning workflow* [11]. This process is divided into nine stages and is shown in Figure 2. Those nine stages can be grouped into four phases. The first phase consists of the model requirements stage, in which the objective of the machine learning task is defined. Furthermore, the type(s) of model that could be used to implement this objective get selected. The initial planning phase is followed by the data preprocessing phase, which includes the stages data collection, data cleaning and data labelling. The next phase describes the model engineering and includes potential feature engineering, as well as the model training and evaluation. If a good performing model is found, the model can be deployed into production, in the last phase, the deployment and monitoring stages in Figure 2. Once the model is used in production, the model needs to be monitored, to measure its performance and find out at which point a potential retraining is needed. The authors emphasise that – in contrast to their illustration (Figure 2) – the machine learning workflow is generally not linear, as the illustration implies. The workflow is iterative and contains multiple feedback loops, as indicated by the arrows.

B. Experiment Tracking

Langley [12] describes machine learning as an experimental science and compares the process of finding a good model to the empirical sciences of physics and chemistry. This aligns with the results from interviews Hill et al. [13] conducted with various machine learning practitioners in 2016. Seven out of seven interviewees experienced the need ”to resort to basic trial and error”. Langley defines an experiment as the process of examining the effect of varying one or more independent variables on some dependent variables [12]. Hence, an experiment consists of multiple runs. According to Vartak et al. [14], ”data scientist often built hundreds of models before arriving at one that met some acceptance criteria”. Each model built can be seen as the dependent variable of a run. However, experiment tracking tools can also be used in the pipeline development phase, introduced by Garcia et al. [10], which does not produce a model, but a training pipeline. In this case, the dependent variable would be the training pipeline. Therefore, the following definition of an experiment is used in this paper:

Definition (Experiment): A run is a part of an experiment, it has a specific set of independent variables that produces a model or a training pipeline. An experiment is a collection of runs that try to solve the same problem or business task. The objective of an experiment is to find the set of independent variables that results in the best dependent variable(s).

It should be noted that usually in practice it is not possible or at least not economically feasible to find the best independent variables [15].

Various possibilities exist to assess the quality of a model. A common approach is to calculate a metric for prediction quality (such as accuracy) on a dataset not used for training. However, additional (nonfunctional) quality measures might exist, e.g., the inference time, the training time or the explainability of a prediction.

Due to the fact that the number of experiment runs might be enormous, it is very helpful to track the experiment and its runs. The term *experiment tracking* describes the process of saving the information related to the experiment and its runs, to allow further evaluation. Although typically the verb *to track* is used in combination with experiments, some tools evaluated in this work have functionalities that use the words *log* or *logger*. Thus, both terms are treated as synonyms in this work.

In its easiest version, tracking can be done manually, alternatively one of the tools presented in Section IV can be used. A Kaggle survey conducted in 2020 [4, p. 27] showed that in most cases tracking is either done manually or not done at all. In theory, automating the process of tracking by using one of the tools presented later should have many

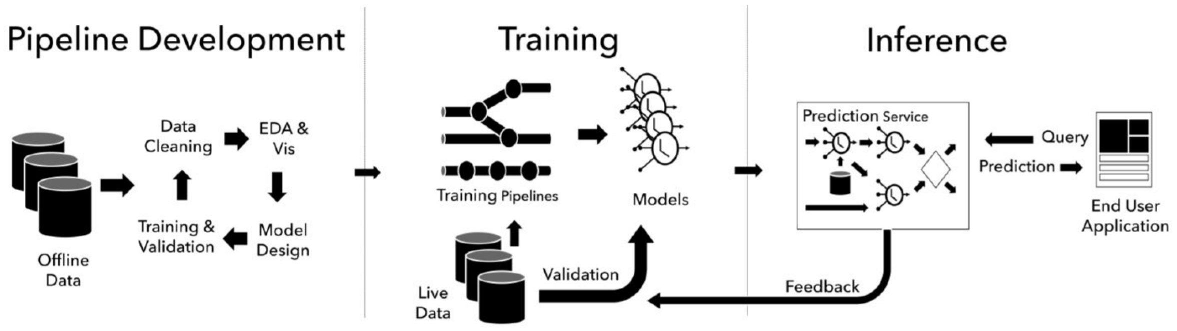


Fig. 1. Machine Learning Lifecycle (from [10])

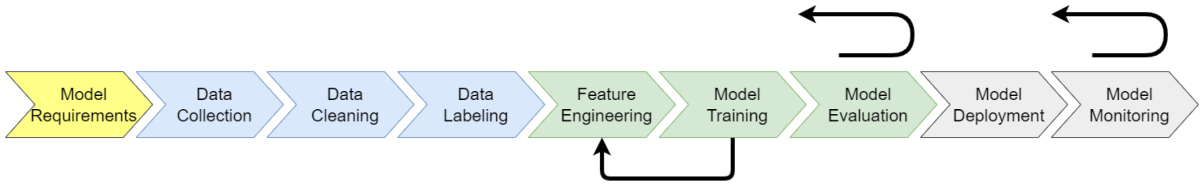


Fig. 2. Machine Learning Workflow based on [11]

advantages. Manual tracking can be error prone, wrong data may get saved or the tracking may be forgotten. Additionally tools might provide functionality to facilitate working in teams or analyzing the tracked data. Automating the process allows the machine learning practitioner to focus on developing the best model.

Either way, tracking experiments brings multiple advantages: Keeping track of all the runs makes it easy to find the best variables. Additionally, it is easy to see, which sets of independent variables have already been tried out or might be worth trying out in the future. This is especially helpful if the work is done in teams, or if the responsible person changes. With the right tool, tracked experiments can be easily compared. If a model is used in production, it can be very helpful to have the information available on how the model was created. Another advantage – which applies especially to research – is the fact that results may need to be reproduced. Furthermore, establishing the use of an experiment tracking tool in a company or a project provides the benefit of a structured way to access the data generated during experimentation, regardless of the individuals responsible for the experiments.

C. Related Work

Experiment Tracking can be seen as a part of MLOps, which can be described as a common set of practices that includes the development of a machine learning application as well as its operations [16]. 41% of business decision makers named ”versioning and reproducibility of models” as a machine learning challenge in a survey conducted by Algorithmia in 2020 [17], making it the second most often named challenge after ”scaling up” with 43%. Experiment tracking should help to tackle this challenge.

Research has been conducted and lead to the presentation of individual frameworks such as MLflow [18], [19]. Additionally, new tools have been developed or proposed based on evaluations of existing tools by Scotton or by Zárate et al. [20], [21]. Other work e.g., by Hewage and Meedeniya focuses on comparing existing experiment tracking tools [16]. The work by Hewage and Meedeniya is different to our work, as it does not include a comparison about the ease of use nor the accessibility of tracked data. Also, the selection of compared tools is different. It does not include DAGsHub or Neptune but includes other tools instead.

D. Reproducibility Requirements

In a reproducibility challenge, Pineau showed that most challenge attendees found it at least reasonably difficult to reproduce the result of a paper of the *International Conference on Learning Representations 2018* [22]. Pineau also published a machine learning reproducibility checklist [23], which is supposed to help increase the reproducibility of experiments. Tatman et al. [24] define three levels of reproducibility for research: low, medium and high reproducibility. The lowest level of reproducibility is achieved by publishing the paper. According to the authors, the medium level is achieved, when the code is published along with the used data. The highest level can be reached by additionally providing the environment.

In the following subsections the requirements for reproducibility introduced by Tatman et al. [24] as well as the terms *hyperparameters* and *metrics* will be explained in detail.

1) Code: Similar to traditional programming, machine learning highly depends on the source code. There are several tools to effectively version source code. A developer survey by StackOverflow in 2018 [25] showed that almost 90 % of the developers use Git as a version control system. There is no valid reason to not track the code used in machine learning projects with Git. However, in a fast developing process, experiment runs might be executed, without committing the code beforehand. This would lead to a lack of reproducibility, as Git needs a commit to restore a state of the code.

2) Data: Besides the code, data plays an essential role in machine learning, because different data can lead to different results. As the kind of data depends on the business task, the data format varies. Common data formats are text, image or video. Due to the partly large data resources, a suitable tool for the efficient storage of different variants of a data resource should be used.

3) Environment: Providing information about the environment is certainly only necessary for some use cases. However, it can contain important information of the original run, such as the used hardware, the used operating system or the software dependencies. Thus, keeping track of the environment can be helpful to reproduce a run. Tatman et al. [24] propose three possibilities to share the environment: Either by using a hosted service, or by providing a container or virtual machine, which includes all dependencies. At minimum, the used libraries and their versions should be tracked.

4) Hyperparameters: According to Bergstra et al. [26], hyperparameters configure the machine learning algorithm before training, whereas, in the present paper, any kind of configuration parameters of the experiment run (not only the machine learning algorithm) will be considered as hyperparameters. As any change in configuration might result in different results, it is recommended to track as many hyperparameters as possible. Although hyperparameters are often tracked implicitly when they are defined in the code and the code is versioned, hyperparameters should be tracked explicitly to allow easier comparison.

5) Metrics: A metric is an evaluation measure calculated to quantify "the effectiveness of a complete application that includes machine learning components" [15]. Most of the times, metrics will be calculated based on a model's predictions on data that has not been used for training. Different metrics with varying strengths and weaknesses exist. For classification tasks for example, accuracy or precision can be used. Accuracy is defined as the fraction of correct predictions out of all predictions [15]. Metrics can be used to compare different runs of an experiment and can be considered as one of the dependent variables of the experiment. Whatever type of metric is used is actually not important for experiment tracking.

III. Experiment Tracking Tools

The main goal of experiment tracking is to save information during experimentation in order to be able to access it later. As a result, most experiment tracking tools consist of at least three components, as shown in Figure 3. Some kind of client software – for example a Python library – is required to store the tracked information during experimenting on a persistent data storage or send it to a server. The data can often be retrieved programmatically through the client or be viewed in a Graphical User Interface (GUI). The exact functionality of those components differs between the available tools.

A. Requirements

As already discussed in Subsection II-B, tracking of code, data, the used environment, hyperparameters, and metrics are elementary requirements for such a tool. Additional requirements examined in our research also include the following aspects:

1) Storing of Models: Training a model can take a long time. Therefore, the models should be stored and linked to the hyperparameters and metrics. This avoids time consuming retraining e.g., if a model should be evaluated on new data.

2) Accessibility of Tracked Information: Tracking is a prerequisite however, the tracked data will only provide value, if the tracked information can be accessed in a simple yet powerful way. This includes a user interface, which provides a clear and customizable overview of all runs, as well as the possibility to compare runs in depth. Filtering the runs with easy but rich querying options is also part of this requirement. Besides that, the tool should provide a possibility to create and show plots. If additional interfaces, e.g., an API, exist, they will be useful as well.

3) Collaboration: According to Tabladillo et al. [27], bringing data science projects to production requires different tasks. For this reason, data science projects are often worked on in teams composed of different roles. Therefore, the tool should facilitate collaborative work. This includes the possibility of viewing existing results of different team members and adding new results by executing new runs. To achieve this, a form of access management is required.

4) Initial Setup and Infrastructure: Because tracking machine learning experiments should facilitate the work of teams, tools will only be taken into consideration if they have low barriers to entry. Thus, this requirement describes the initial investment needed to set up and use the tool. The initial setup is everything that does not need to be repeated if the same tool is used in another project (given the projects can use the same infrastructure). As cloud tools might have an advantage concerning the initial setup, it must be kept in mind that saving data off-premises might not be a possibility in any case due to legal or corporate regulations.

5) Ease of Integration: Similar to the previous requirement this requirement concerns user-friendliness. Yet, unlike the initial setup and infrastructure, the ease of integration describes how easy it is to include the tool into a specific project. This means, for example, project-specific configuration or source code changes.

IV. Examined Tools

In a market research, the following tools with experiment tracking functionality were identified.

- Aim [28]
- Amazon SageMaker Experiments [29]
- Azure Machine Learning [30]

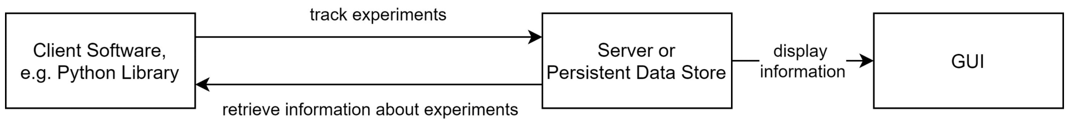


Fig. 3. General Architecture of an Experiment-Tracking-Tool

- ClearML [31]
- Comet [32]
- DAGsHub [33]
- DominoDataLab [34]
- Guild AI [35]
- H2O MLOps [36]
- Iterative Studio [37]
- MLflow [38]
- Neptune [39]
- Paperspace Gradient [40]
- Polyaxon [41]
- Sacred [41] in combination with Omniboard, Incense or Sacredboard (GUIs)
- TensorBoard [42]
- Valohai [43]
- Verta [44]
- Vertex AI [45]
- Weights & Biases [46]

The research was conducted online, using search engines, blogs, forums, as well as the websites of the respective tools.

To allow an in-depth evaluation of the tools in the scope of this work, the tools listed previously have to be limited to a reasonable amount. The tools were selected in consultation with a project team at inovex, actually developing a multilingual and multidomain Conversational AI. The selection was influenced by requirements given from the project team. First, it was required that tracking tool is not running in a cloud ecosystem only nor creates a platform lock-in, like Azure Machine Learning, Amazon SageMaker Experiments or Paperspace Gradient do. Furthermore, the tracking tool should be independent of runtime and used libraries, which did exclude Tensorboard. For the sake of brevity four tools were examined only. The selection was based, in addition to the stars on github, on the simplicity of integrating the tools into the project and the familiarity within the project team. Therefore, Aim and Polyaxon were not considered, but they are very interesting tools and should be included in future research.

In this process, MLflow, ClearML, Neptune and DAGsHub were adopted for a more detailed evaluation. MLflow was selected because it is one of the most established and widely used tools. ClearML was assessed because of its wide range of operating options. It can be used for free (even in small teams) as a hosted option, operated self-hosted for free, but also be used with a paid plan. The most important argument for choosing Neptune was that it promises an effortless setup. The last option evaluated was DAGsHub, as it makes use of Data Version Control (DVC) [47] for versioning data, like the project. In the next subsections each tool will be evaluated based on the requirements defined in Subsection III-A and an exemplary integration will be provided.

A. *MLflow*

The open-source tool MLflow is developed by Databricks and was introduced by Zahari et al. [48] and launched in June 2018. At the time of writing this paper (October, 2022) the latest released version was 1.29.0 [49]. The software itself is shipped as a Python package and can be either hosted on own server or used as a Software as a Service (SaaS) offering by Databricks called *Managed MLflow*. Since Databricks' main business is offering managed versions of Apache Spark clusters, Managed MLflow is tightly coupled to these offerings. By comparing the managed with the open source version the managed offers mostly features that allow integrations into the Databricks eco-system. In contrast to the self-hosted version the managed one offers notebook and workspace integration to Databricks, as well as a role based user management. In addition also an integration to the aforementioned clusters is offered [50].

With the version 1.27.0 MLflow introduced a new experimental feature called *MLflow Pipelines*. This feature provides a framework to structure the whole cycle of an machine learning project. Hereby the user can specify pipelines either with Python code or by configuration files. The steps of these pipelines define parts like data preprocessing, splitting data into parts, evaluating trained models or storing models. MLflow provides templates that fit for common machine learning tasks. Since the pipelines are either defined by Python code or configuration files they can be easily stored in a repository like Git [51].

As already mentioned the open-source version of MLflow is shipped as a Python package and can be easily installed by any Python package manager like pip or conda. MLflow uses a naming similar to our definition in Subsection II-B where runs are grouped into experiments. The most basic setup of MLflow just uses a local file system to store experiment and run metadata. In order to get a more scalable setup it is advised to setup a relational database as a backend. Here MLflow is able to use a varity of databases like mysql, mssql, sqlite and postgresql. Apart from experiment and run metadata MLflow uses either a local file system or a cloud object storage (like AWS S3, Google Cloud Storage or Azure Blob Storage) to store artifacts like trained models. If not configured explicitly the metadata as well as the artifact data are stored into the local file system [52].

```python
1  import mlflow
2  mlflow.set_tracking_uri("postgresql://postgres:
       postgres@172.3...")
3  mlflow.set_experiment("MyProject") #group runs
4  with mlflow.start_run() as run:
5      hyperparams = {"lr": 0.01,}
6      mlflow.log_params(hyperparams)
7      #Training placeholder, model stored in var model
8      mlflow.pytorch.log_model(model, "log_r",)
9      mlflow.log_metric("acc", 0.99)
10     mlflow.set_tag("performance", "best")
```

Listing 1. MLflow example code

To start tracking with MLflow, a run has to be started as shown in Listing 1. By using a context manager to start the run, the run will be ended automatically (line 4). MLflow differentiates between metrics and params; both can be logged to MLflow by using the respective function. With the *log_params* (line 6) function a set of values like hyperparameters describing the current run can be stored. This function takes all kinds of values, which can be stringified. In contrast the *log_metric* (line 9) function, where only numeric values can be passed. That function is used to keep track for evaluation metrics (like precision or recall) for one run. Both functions exists as singular to log one value, or as plural to log multiple values (here, a dictionary is passed, as the only parameter and the name and values of the dictionary will be used. In addition a run can be marked by using the *set_tag* (line 10) function. That function is intended to mark a run e.g., as the current best performing. Not shown in the listing is the *log_artifact* function, which can be used to store a local file attached to the current to MLflow. Grouping multiple runs together allows easy viewing and comparison in the GUI. This can be achieved by setting up an experiment (line 3).

As mentioned before, MLflow uses the local file system by default to store metadata. By passing an URI pointing to a database to MLflow these data will be stored there [52].

The MLflow GUI in Figure 4 shows all the hyperparameters and metrics in a clear table. Runs of the same experiment can be compared and metrics are automatically plotted. In addition to the GUI, data tracked with MLflow can be retrieved via Python, R, Java and REST APIs. MLflow does not provide a dedicated way to keep track of the data used for training. It does not automatically log information about the environment either. However, with *MLflow Projects*, MLflow wants the users to manually specify their environment [54]. This can be achieved by creating a conda yaml file or a docker image and structuring the project by providing entry points and default parameters.

MLflow can be used for free in teams, however, this requires shared data storage, which has to be set up by yourself.

B. Neptune

Neptune is a tool developed by Neptune Labs. It is described as a ”metadata store for MLOps” [39]. Neptune consists of a server (closed-source) and a client (Python & R packages, open-source). The first version of the Python package was released in March 2019 [55]. At the moment (October 2022), 0.16.9 is the newest version. In the last year, an R client has also been added [56]. With 0.9.0 (released end of May 2021), the API received a significant update, introducing a new and slightly different way to use Neptune, while maintaining backward compatibility.

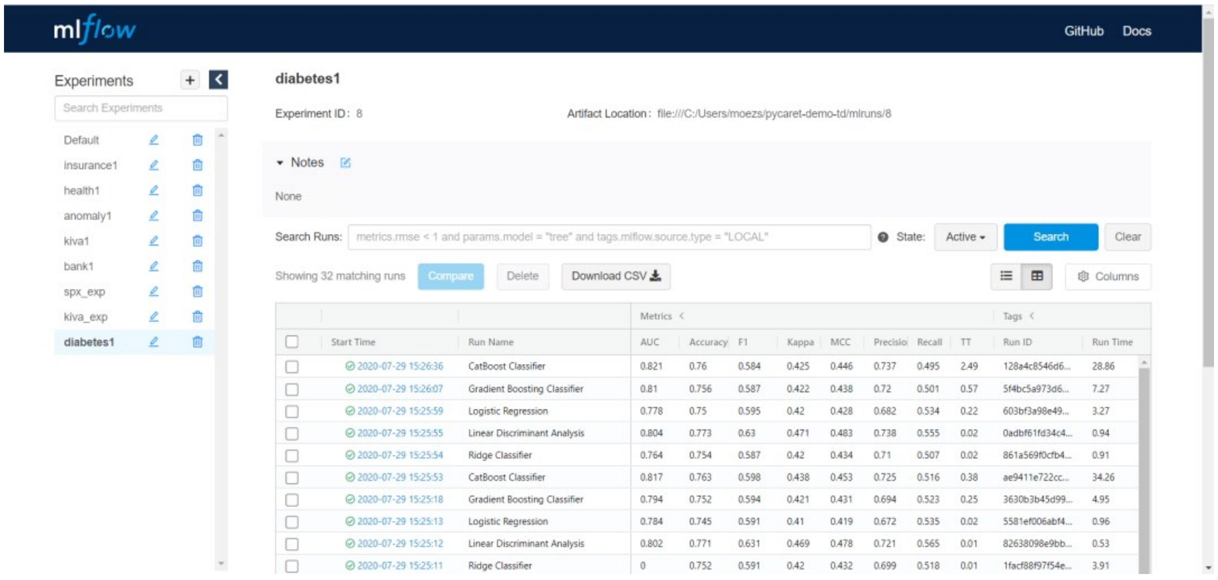
To get started with Neptune, an account has to be created at neptune.ai and an API token has to be generated. To track experiments, a project (similar to an experiment in MLflow) has to be created in the Neptune Web App or via the available management API. After those setup steps, Neptune is ready for use.

```python
1  import neptune.new as neptune
2  run = neptune.init(project="tbud/MyProject")
3  hyperparams = {"lr": 0.01,}
4  run["data/train"].track_files("./datasets/train")
5  run["hyperparams"] = hyperparams
6      #Trainingloop placeholder
7      run["loss/train"].log(the_current_loss)
8  torch.save(model, "log_r.mdl")
9  run["model"].upload("log_r.mdl")
10 run["acc"] = 0.99
11
12 run["model_pickle"].upload(neptune.types.File.as\
       _pickle(model))
```

Listing 2. Neptune example code

Listing 2 shows the integration of Neptune, after importing the new Neptune API, we can initialize a run and assign it to a project (line 2). Neptune does not differentiate between metrics and hyperparameters. To log values with Neptune, a notation with square brackets and strings as keys (e.g., `run["some_key"]`) is used, which is similar to adding new values to a dictionary in Python (line 10). To track series such as the loss, the log function has to be used (line 7). This automatically generates a plot in the GUI. To structure values, a structured namespace can be used by putting a slash in the key name (e.g., `run["namespace/some_key"]`). This concept is then used to group and structure values in the GUI. The namespace can also be used to easily distinguish between metrics and hyperparameters. To upload a trained model, it first has to be saved locally (line 8) and can then be uploaded to Neptune using the upload method (line 9). Neptune provides a dedicated model registry that shows all production-ready models in a centralized way. It further enables the user to track transitions between different development stages of a model.

Neptune provides basic functionalities to keep track of the data used in the experiment runs. Neptune can calculate the hash value of a file or folder and store it with additional metadata (path, size and the last-modified date), which allows the user to see if the dataset has changed between experiment runs. This can be achieved by using the `track_files` method as shown in line 4. However, the dataset is not stored on the Neptune server and its data can not be retrieved. If a small dataset is used, it might be as well an option to upload the whole dataset. This will than be handled as an artifact, which is similar to the handling of a model file. A dataset can be uploaded by using the `upload()` function for single files or the `upload_files()` function for multiple files or directories. Arbitrary Python objects can be uploaded directly as a pickle (a Python-specific serialization format), without the need to locally store the pickle file first, by



diabetes1

Experiment ID: 8

Artifact Location: file:///C:/Users/moezs/pycaret-demo-td/mlruns/8

Notes

None

Search Runs: metrics.rmse < 1 and params.model = "tree" and tags.mlflow.source.type = "LOCAL"

State: Active

Showing 32 matching runs

		Metrics								Tags	
Start Time	Run Name	AUC	Accuracy	F1	Kappa	MCC	Precisio	Recall	TT	Run ID	Run Time
2020-07-29 15:26:36	CatBoost Classifier	0.821	0.76	0.584	0.425	0.446	0.737	0.495	2.49	128a4c8546d6...	28.86
2020-07-29 15:26:07	Gradient Boosting Classifier	0.81	0.756	0.587	0.422	0.438	0.72	0.501	0.57	5f4bc5a973d6...	7.27
2020-07-29 15:25:59	Logistic Regression	0.778	0.75	0.595	0.42	0.428	0.682	0.534	0.22	603bf3a98e49...	3.27
2020-07-29 15:25:55	Linear Discriminant Analysis	0.804	0.773	0.63	0.471	0.483	0.738	0.555	0.02	0adbf61fd34c4...	0.94
2020-07-29 15:25:54	Ridge Classifier	0.764	0.754	0.587	0.42	0.434	0.71	0.507	0.02	861a569f0cfb4...	0.91
2020-07-29 15:25:53	CatBoost Classifier	0.817	0.763	0.598	0.438	0.453	0.725	0.516	0.38	ae9411e722cc...	34.26
2020-07-29 15:25:18	Gradient Boosting Classifier	0.794	0.752	0.594	0.421	0.431	0.694	0.523	0.25	3630b3b45d99...	4.95
2020-07-29 15:25:13	Logistic Regression	0.784	0.745	0.591	0.41	0.419	0.672	0.535	0.02	5581ef006abf4...	0.96
2020-07-29 15:25:12	Linear Discriminant Analysis	0.802	0.771	0.631	0.469	0.478	0.721	0.565	0.01	82638098e9bb...	0.53
2020-07-29 15:25:11	Ridge Classifier	0	0.752	0.591	0.42	0.432	0.699	0.518	0.01	1facf88f97f54e...	3.91

Fig. 4. MLflow GUI (from [53])

using Neptune's file type with the `as_pickle()` method, as shown in line 12. In contrast to ClearML, Neptune does not automatically keep track of the computational environment. Thus, software dependencies are not saved when running an experiment, which makes reproducibility more difficult.

The GUI of Neptune looks similar to the MLflow GUI. It includes all the basic functionalities that MLflow has, but also has additional nice-to-have features, such as query completion for filtering or an option to save customized views. Figure 5 shows the run overview of the GUI. In the table, every run is represented by one row, the displayed columns represent metadata of a run and can be easily configured and even be renamed to a custom name. Neptune provides rich filtering options and helps the user writing the query by giving fitting proposals. Besides that, it is possible to group runs together to make the table clearer. Once a table view is customized as needed, it can be saved. This allows to quickly switch between various different views. Comparing multiple runs is easily done by clicking on the eye symbol for the desired runs. Neptune can filter for differences between runs and as well shows a small indicator to quickly see if a value increased or decreased.

Beside the GUI, the data can also be retrieved through a Python and R API. As a drawback, in contrast to MLflow and ClearML, Neptune does not provide a REST API.

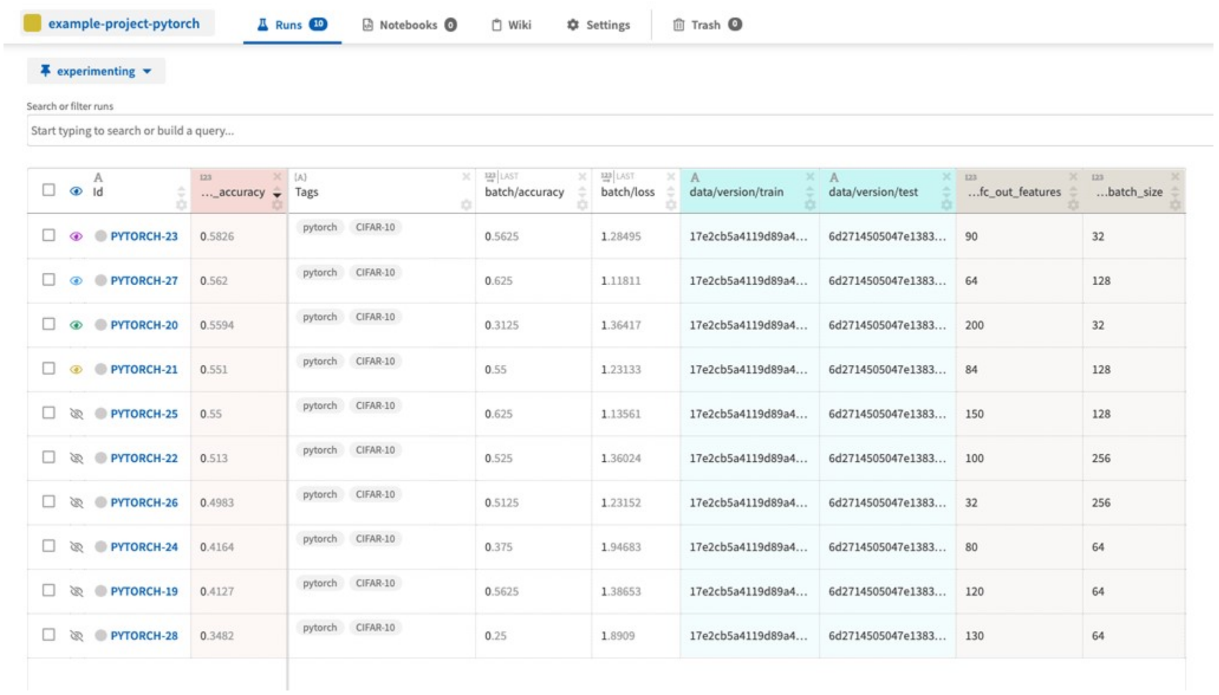
Similar to MLflow, Neptune's focus is tracking models, metrics and hyperparameters. Neptune has the opportunity to save all the tracked information on their servers, which is the most common and easiest way. However, this could raise data governance issues. Thus, Neptune does now also offer the possibility to deploy the server code on-premises or in a private cloud [57]. For single users and in special cases (e.g., academia, research) Neptune can be used for free. Paid plans have fixed prices, regardless of the amount of users. However, storage and usage limits exist (which can be increased by additional payments).

Neptune additionally provides the option to integrate Jupyter notebooks into projects. This can be done by installing the Neptune notebook extension. After enabling it, snapshots of notebooks can be created and saved to the Neptune project with the capability to compare different versions of the same notebook. This allows saving data exploration work next to the experiment runs. Furthermore, an external tool exists that allows the combination between Neptune and MLflow [58]. In this case, MLflow *runs* can be stored on a Neptune server. That way MLflow experiments can profit from the organization and collaboration features of Neptune. At the time of writing (October 2022), the new client wasn't yet supported by this tool. Also, Neptune provides a GitHub Actions template to make the data tracked in Neptune available in GitHub Pull Requests.

C. ClearML

ClearML is an open-source tool developed by Allegro AI, it was formerly known as Allegro Trains. At the time of writing, the current version of ClearML is 1.7.1. ClearML sends data to a ClearML server to store the data. This server can either be the SaaS solution provided by ClearML or a self-managed setup, which can be relatively easy set up for example with the Docker images provided by ClearML. While self operating is completely free, the free SaaS version has some limitations to it, such as the possible number of project members.

The ClearML client can be easily installed using pip. To use ClearML app credentials have to be added to the environment in which the experiment is conducted to connect the client with the account on the server. The credentials can be created in the workspace section of a ClearML account in the web interface.



Id	..._accuracy	Tags	batch/accuracy	batch/loss	data/version/train	data/version/test	...fc_out_features	...batch_size
PYTORCH-23	0.5826	pytorch CIFAR-10	0.5625	1.28495	17e2cb5a4119d89a4...	6d2714505047e1383...	90	32
PYTORCH-27	0.562	pytorch CIFAR-10	0.625	1.11811	17e2cb5a4119d89a4...	6d2714505047e1383...	64	128
PYTORCH-20	0.5594	pytorch CIFAR-10	0.3125	1.36417	17e2cb5a4119d89a4...	6d2714505047e1383...	200	32
PYTORCH-21	0.551	pytorch CIFAR-10	0.55	1.23133	17e2cb5a4119d89a4...	6d2714505047e1383...	84	128
PYTORCH-25	0.55	pytorch CIFAR-10	0.625	1.13561	17e2cb5a4119d89a4...	6d2714505047e1383...	150	128
PYTORCH-22	0.513	pytorch CIFAR-10	0.525	1.36024	17e2cb5a4119d89a4...	6d2714505047e1383...	100	256
PYTORCH-26	0.4983	pytorch CIFAR-10	0.5125	1.23152	17e2cb5a4119d89a4...	6d2714505047e1383...	32	256
PYTORCH-24	0.4164	pytorch CIFAR-10	0.375	1.94683	17e2cb5a4119d89a4...	6d2714505047e1383...	80	64
PYTORCH-19	0.4127	pytorch CIFAR-10	0.5625	1.38653	17e2cb5a4119d89a4...	6d2714505047e1383...	120	64
PYTORCH-28	0.3482	pytorch CIFAR-10	0.25	1.8909	17e2cb5a4119d89a4...	6d2714505047e1383...	130	64

Fig. 5. Neptune GUI (from [39])

```
1  from clearml import Task, Logger, Dataset
2  path = Dataset.get(dataset_project="MyProject/data",
       dataset_name="ds_1").get_local_copy()
3  task = Task.init(project_name="MyProject",
4          task_name="Task1", reuse_last_task_id=False,
5          output_uri="gs://MyProject")
6  hyperparams = {"lr": 0.01,}
7  task.connect(hyperparams)
8  cur_log = task.get_logger()
9  #Training placeholder, model stored in var model
10     cur_log.report_scalar("train", "loss", 1, ep)
11 torch.save(model, "log_r.mdl")
12 cur_log.report_single_value("accuracy", 0.99)
```

Listing 3. ClearML example code

An exemplary use of ClearML is shown in Listing 3. Initializing an object of ClearMLs Task class by calling its `init()` method, starts the tracking with ClearML, which is shown in line 3. Setting `reuse_last_task_id` to False (line 4) ensures that this task will not override an old task. The `output_uri` (line 5) specifies the location for the artifacts (e.g., the model) and is in this example set to a Google Cloud Storage. In ClearML a task is the name for everything that can be tracked, similar to a run in MLflow. By starting the tracking, ClearML automatically keeps track of a multitude of things, such as:

- Information about the Git repository, including the name of the current branch, the current commit ID and the output for the `git diff` command.
- Names and values of command line arguments that have been passed using standard Python packages, such as click or argparse.
- Plots created by libraries e.g., matplotlib, plotly or seaborn.
- Information logged by Tensorboard [42] and TensorboardX [59].
- Installed and used packages.
- Information about the resource usage (CPU, GPU, disk space, etc.).

Besides the information that is logged automatically, additional information can get logged with ClearML. This can be achieved by connecting an object to the task as shown in line 7. This object can be a Python dictionary or an object of a (custom) class.

An object of the ClearML class `Logger` is required, to log metrics. The `task.get_logger()` (line 8) and `Logger.current_logger()` (not shown in this Listing) functions return the logger object, which is is connected to the current task. To log metrics, the method `report_scalar()` of the logger object can be used as shown in line 10. This method is especially useful for metrics that change over time, as ClearML automatically creates a plot displaying the change over time in its GUI. The method requires four parameters: title, series, value and iteration. The title specifies the name of the scalar and the plot of the scalar. Multiple series can be grouped into one plot by providing the same title. Series describes the name of the series of the plot, value the value and iteration provides the x-coordinate for the plot line. For single values there is a `report_single_value()` method, which only requires the name and the value and does not result in the value being displayed in a plot. This is useful for reporting metrics that only have one value in an experiment run. ClearML also provides more sophisticated options such as `report_matrix()` to log a confusion matrix or `report_histogram()` to log a histogram.

Besides its hyperparameter and metric tracking capabilities, ClearML provides a possibility to efficiently store and manage large datasets. It works similar to DVC [47]: Before up- or downloading files, hash sums are calculated and compared to avoid traffic in case there have been no changes. ClearML also allows to store additional metadata about the data files, which can then for example be retrieved through the GUI. This allows versioning datasets even for binary files. A simple example of the integration into code is given in Listing 3. To get the local path to a dataset managed with ClearML, the dataset has to be queried with the `Dataset.get()` function (line 2). The `get_local_copy()` (line 2) function ensures that a local copy is available and returns the path, which can then be used for training. ClearML automatically tracks and uploads (trained) models if they are saved using the respective functions of the most common Python machine learning frameworks (e.g., Tensorflow, Pytorch, scikit-learn). The destination of the upload is specified during the initialization of the task (line 3), in this case a Google Cloud Storage, but other common cloud storages are also supported. The model can then be accessed in Python by retrieving the respective task and choosing the desired model.

As mentioned earlier, initializing a ClearML Task automatically tracks the installed and used packages including their version numbers. This can help to reproduce results at a later stage.

Figure 6 shows a screenshot of the GUI. Multiple tasks are collected in a project, ClearML also allows the creation of sub-projects for projects that require more organization. While the overview table of the experiments looks similar to Neptune and MLflow, the detailed view of the task is very nested and can overwhelm new users. This is in our opinion the biggest downside of ClearML compared to the other tools: due to its huge amount of possibilities, it requires more time to familiarize. However, we think this time is well invested since ClearML provides a lot of options and possibilities for the user. Those options include possibilities to show and hide columns as well as sorting and filtering or a function to compare runs, in which case the differences between runs will be highlighted. Besides examining data in the GUI, the data tracked with ClearML can also be retrieved trough the Python API or through a REST API. Projects are organized in workspaces, team members can be added to a workspace to allow collaborative work. In the free hosted version, workspaces are limited to three members, no such limit exists on the self-managed version. ClearML provides also additional features, such as logging debug samples of images, audio, video samples, which can help understanding the data, which was used to conduct experiments. To help increasing reproducibility, by default ClearML automatically sets a random seed for Tensorflow, Pytorch, and random.

D. DAGsHub

In contrast to the other presented tools, DAGsHub pursues a different approach. It makes use of existing open-source technologies and provides unified storage and a GUI for them (however, DAGsHub itself is not open-source). The open-source tools combined by DAGsHub are:

- DVC [47] is used to keep track of the data and models.
- Git keeps track of the code.
- MLflow or the DAGsHub Client can be used to track hyperparameters and metrics.

The interaction of the different tools is presented in Figure 7.

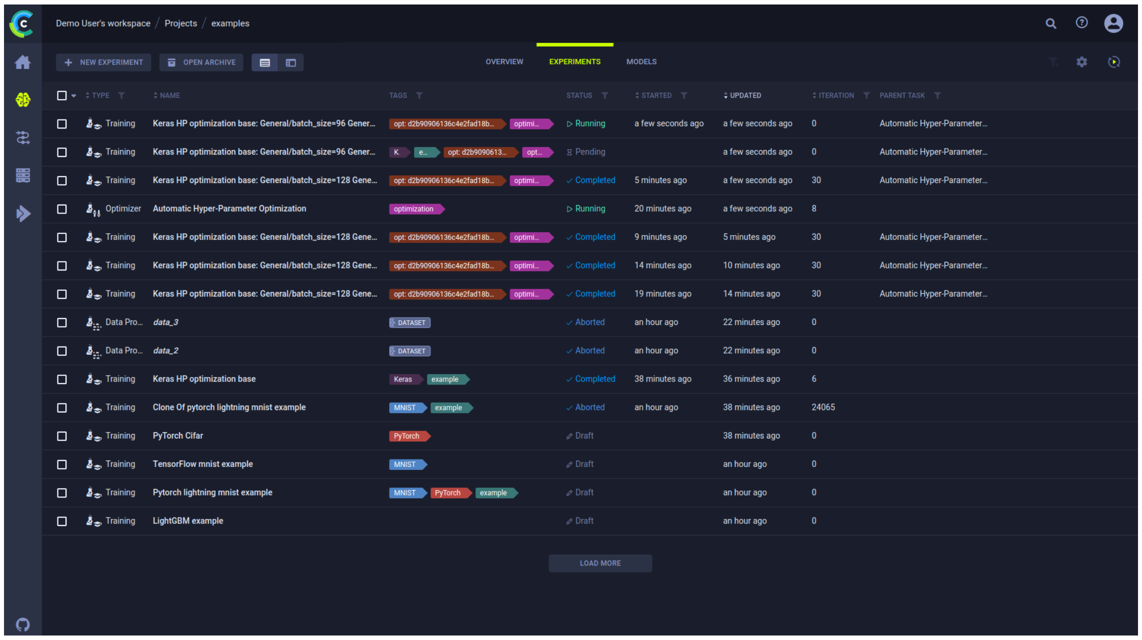
In order to take full advantage of DAGsHub, Git and DVC as well as MLflow or the DAGsHub Client should be installed on the client. In case of using MLflow to log to DAGsHub the integration into code is almost identical to the one shown in Listing 1. The only required change is to set the tracking URI specified in line 2 to the URI provided in the DAGsHub GUI. The functionality when using DAGsHub concerning the tracking of hyperparameters and metrics is similar to MLflow. An advantage of using DAGsHub in comparison to MLflow is its capability to keep track of the data. This is achieved by using DVC. Tracking data files with DVC is similar to the use of Git. Files (or even whole directories) have to be added to DVC to be tracked by using the CLI. By doing this, the files are added to the gitignore file and a small .dvc file is created. The .dvc file contains the size of the added file as well as its hash sum. Git versions the .dvc file and the data files can be pushed and pulled to a remote storage, which is better suited for handling large (amounts of) files that might not be text files.

While using DVC in combination with MLflow does not require using DAGsHub, the advantage of using DAGsHub is the unified GUI it provides.

Besides the datasets, also the created models are supposed to be tracked with DVC. However, in comparison to other tools where this can be achieved in the training code, DVC is a CLI tool and thus using it requires more effort in general. In order to facilitate using Git and DVC from the CLI, Fast Data Science (FDS) a wrapper around the two tools has been created by the DAGsHub team [61]. Using FDS combines similar commands of Git and DVC and thus accelerates the usage of both tools.

DAGsHub does not provide any functionality to keep track of the computational environment. However, the basic functionality of MLflow can also be used when using DAGsHub.

The GUI of DAGsHub shown in Figure 8 is familiar to users of the most common Git webservices, but additionally includes a data section, as well as an overview of the experiment runs as known from MLflow, thus most relevant information are combined in one place. It is the advantage of using the open-source tools without DAGsHub. However, most organizations most likely already use a different Git webservice. Especially since DAGsHub lacks functionalities, which other Git webservices provide and, which are often adapted (e.g., CI/CD functionality), organizations might not be willing to migrate to DAGsHub. For this case DAGsHub provides the functionality to mirror another Git repository, which however, contradicts DAGsHub's main advantage of having everything in one place.



TYPE	NAME	TAGS	STATUS	STARTED	UPDATED	ITERATION	PARENT TASK
Training	Keras HP optimization base: General/batch_size=96 Gener...	opt: d2b90906136c4e2fad18b... optimi...	Running	a few seconds ago	a few seconds ago	0	Automatic Hyper-Parameter...
Training	Keras HP optimization base: General/batch_size=96 Gener...	K e... opt: d2b9090613... opt...	Pending		a few seconds ago	0	Automatic Hyper-Parameter...
Training	Keras HP optimization base: General/batch_size=128 Gene...	opt: d2b90906136c4e2fad18b... optimi...	Completed	5 minutes ago	a few seconds ago	30	Automatic Hyper-Parameter...
Optimizer	Automatic Hyper-Parameter Optimization	optimization	Running	20 minutes ago	a few seconds ago	8	
Training	Keras HP optimization base: General/batch_size=128 Gene...	opt: d2b90906136c4e2fad18b... optimi...	Completed	9 minutes ago	5 minutes ago	30	Automatic Hyper-Parameter...
Training	Keras HP optimization base: General/batch_size=128 Gene...	opt: d2b90906136c4e2fad18b... optimi...	Completed	14 minutes ago	10 minutes ago	30	Automatic Hyper-Parameter...
Training	Keras HP optimization base: General/batch_size=128 Gene...	opt: d2b90906136c4e2fad18b... optimi...	Completed	19 minutes ago	14 minutes ago	30	Automatic Hyper-Parameter...
Data Pro...	*data_3*	DATASET	Aborted	an hour ago	22 minutes ago	0	
Data Pro...	*data_2*	DATASET	Aborted	an hour ago	22 minutes ago	0	
Training	Keras HP optimization base	Keras example	Completed	38 minutes ago	36 minutes ago	6	
Training	Clone Of pytorch lightning mnist example	MNIST example	Aborted	an hour ago	38 minutes ago	24065	
Training	PyTorch Cifar	PyTorch	Draft		38 minutes ago	0	
Training	TensorFlow mnist example	MNIST	Draft		an hour ago	0	
Training	Pytorch lightning mnist example	MNIST PyTorch example	Draft		an hour ago	0	
Training	LightGBM example		Draft		an hour ago	0	

Fig. 6. ClearML GUI (from [60])

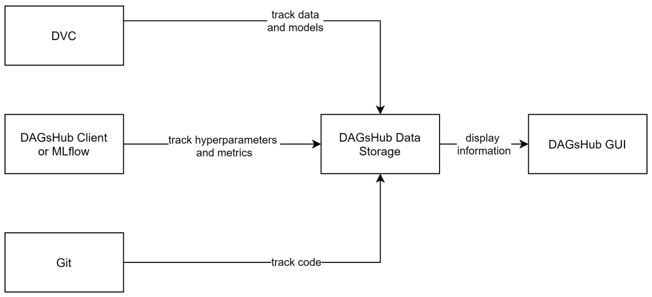


Fig. 7. DAGsHub Architecture

With a free DAGsHub plan, the number of collaborators and storage is limited. Paid plans exist, which allow working in bigger teams. DAGsHub probably has the most potential for teams that already use DVC and/or MLflow and want to keep using the tools but would benefit from unified storage and GUI.

Recent advances of DAGsHub are its integration of Label Studio [62], which can be especially helpful when annotating data in teams. As well as a commenting feature, DAGsHub Discussions, or the possibility to use the GUI familiar from MLflow.

E. Comparison

Table I shows a comparison for most of the defined requirements. As tracking the code is done with Git most of the times and tracking the hyperparameters and metrics and the ease of integration are on a similar level for all four tools, these defined requirements are not included in the table. The tools have different strengths and weaknesses when it comes to ease of use, pricing and more advanced requirements, such as tracking data or computational environment.

MLflow has a well-structured API and can be used for free however, does not provide functionality to track data and automatically keep track of the environment. Also, the effort to set up MLflow in a collaborative environment is more elaborate compared to other tools. Neptune, on the other hand, offers a simple setup and highly functional GUI but requires a paid plan when used as a team and only provides basic functionality to track data and no functionality to track the environment. In comparison to the two previous tools, ClearML handles the tracking of data and the computational environment, taking care of all requirements. Additionally, it is open-source and can be self-hosted or used as a free or paid service. The biggest weakness of ClearML based on our requirements is that because of its richness of features it might not be as easy to use as other tools that might provide less functionality. DAGsHub does not provide functionality to track the environment. However, with DVC the data can be tracked. As a result, DAGsHub can be considered as a good choice for teams already using DVC and MLflow who like to have unified storage and GUI.

V. Conclusion and Future Research Perspective

This paper provided an in-depth analysis of four tools with the focus of tracking machine learning experiments. After describing the process of bringing a machine learning model to production and emphasizing the experimental character of training a machine learning model, requirements for machine learning experiment tools were defined based on the needs of an industrial data science project as well as the research conducted in the field of reproducible machine learning. Additionally 20 tools with functionalities to track experiments,

TABLE I
COMPARITIVE OVERVIEW OF MLFLOW, NEPTUNE, CLEARML AND DAGSHUB

	MLflow	**Neptune**	**ClearML**	**DAGsHub**
Evaluated version	1.29.0	0.16.9	1.7.1	as of September 2022
Data	basic functionality to calculate the hash values and upload it alongside metadata, no way to retrieve the actual data	no dedicated functionality provided	Data Managing and Versioning with ClearML Data	Data Managing and Versioning with DVC
Environment	encourages the user to do it manually (MLflow Projects)	no dedicated functionality provided	automatically keeps track of the installed Python packages and their versions	no dedicated functionality provided
Storing models	easily possible	model has to be stored locally first and can then be uploaded	automatically uploaded if saved locally	possible to store models with DVC, commit required for every upload
GUI	basic GUI	highly customizable & advanced GUI	advanced GUI	unified GUI for data, code, and experiments
Collaboration	possible, requires a shared data storage	possible with a paid account	possible, user limit depends on the operation mode, unlimited for self-hosting	free for public repositories, not free of charge for private repositories
Initial setup and infrastructure	setting up a database or shared file storage is required for collaborative use, alternatively Managed MLflow can be used	easy setup, as the user does not have to take care of the infrastructure necessarily	hosted as well as self-hosting options exist, images to make the setup easier exist	easy setup if DAGsHub is used as Git and DVC storage
Persistence	local file, relational databases, cloud object storage providers	Neptune server, on-premise server	ClearML server, self managed server	DAGsHub Storage, MlFlow backend, cloud object storage providers
Programatic Interfaces	Python, REST-API, R, Java	Python, R	Python, REST-API	Python, REST-API, R, Java (via MLflow)
Workflow support	since 1.27.0 support for pipelines	no native support, but integration into KubeFlow possible	support for pipelines	native support for CI/CD-like pipelines
Ease of Use	intuitive Python API, easy usable by context (with-statement)	intuitive Python API, more explicit method calls necessary, Jupyter notebook integration	intuitive Python API, logs environment automatically	see MLflow

which have been identified in a market research have been presented. Ultimately, four of these tools have been evaluated in detail and compared. This comparison showed all analyzed tools function approximately equally well considering the most basic requirement of tracking hyperparameters and metrics. However, differences exist, considering the more advanced requirements such as keeping track of the data. To conclude, it can be said that the right choice of an experiment tracking tool depends on the specific requirements, and that the open source tool ClearML has been identified as meeting most of the requirements. It has to be noted that due to the quickly changing market of experiment tracking tools, new tools might be released or existing tools might receive new functionality. As a result, further research, also of tools not evaluated in this paper, might be of use.

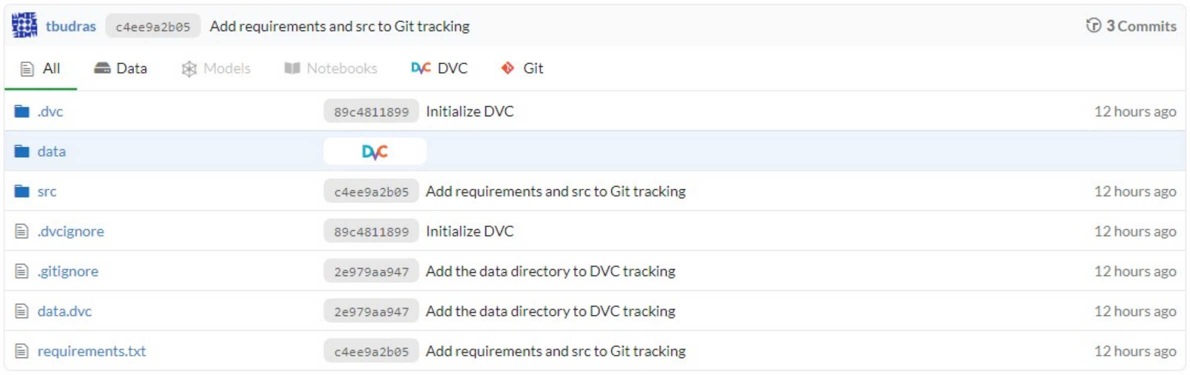
MLflow and Neptune are the tools that have developed recently the most in regard to our requirements. MLflow added a new powerful pipeline feature that eases the training process. Neptune has been adapted to R and now also offers the possibility to self-host a server. Small improvements, e.g., the integration of Label Studio, have been integrated into DAGsHub. In contrast to the other tools, ClearML, which already met most of our requirements, had no major additions. Back then, ClearMl was the favored tool that met our requirements the best. However, especially MLflow and Neptune have caught up.

To better understand why the tools presented in this work have only seen little application in the past, as shown in Sections I and II, further researcher could focus on the difficulties that exist in practice while using such tools. Additionally one or multiple case studies could be conducted, comparing the evaluated tools or a different set using well-defined metrics. Another potential research question could focus on interoperability examining in which cases it makes sense to use multiple tools jointly.

Further, it would be great if standardized formats would be developed to easily switch from one tracking tool to another. This would mitigate platform lock-in effects and improve model life cycle management in the long run. Especially upcoming regulations from administrations [63] that require model documentation and reproducability for longer time periods, would drive such a development.

ACKNOWLEDGMENT

The work is based on the results of a bachelor thesis [64] of the first mentioned author at the company inovex GmbH.



tbudras c4ee9a2b05 Add requirements and src to Git tracking — 3 Commits

.dvc	89c4811899	Initialize DVC	12 hours ago
data	DVC		
src	c4ee9a2b05	Add requirements and src to Git tracking	12 hours ago
.dvcignore	89c4811899	Initialize DVC	12 hours ago
.gitignore	2e979aa947	Add the data directory to DVC tracking	12 hours ago
data.dvc	2e979aa947	Add the data directory to DVC tracking	12 hours ago
requirements.txt	c4ee9a2b05	Add requirements and src to Git tracking	12 hours ago

Fig. 8. DAGsHub GUI

References

[1] T. Budras, M. Blanck, T. Berger, and A. Schmidt, "Comparison of experiment tracking frameworks in machine learning environments," in *Proceedings of the Fourteenth International Conference on Advances in Databases, Knowledge, and Data Applications*, 2022, pp. 21–28.

[2] Machine learning market. [Online]. Available: https://www.marketsandmarkets.com/Market-Reports/machine-learning-market-263397704.html (Accessed 2022-12-13).

[3] P. Warden. The machine learning reproducibility crisis. [Online]. Available: https://petewarden.com/2018/03/19/the-machine-learning-reproducibility-crisis/ (Accessed 2022-12-13).

[4] State of data science and machine learning 2020. [Online]. Available: https://www.kaggle.com/kaggle-survey-2020 (Accessed 2022-12-13).

[5] "Ethics Guidelines For Trustworthy AI," EU Commisssion, Tech. Rep., 2019. [Online]. Available: https://ec.europa.eu/newsroom/dae/document.cfm?doc_id=60419 (Accessed 2022-12-13).

[6] S. S. Alahmari, D. B. Goldgof, P. R. Mouton, and L. O. Hall, "Challenges for the repeatability of deep learning models," *IEEE Access*, vol. 8, pp. 211 860–211 868, 2020.

[7] P. Nagarajan, G. Warnell, and P. Stone, "Deterministic implementations for reproducibility in deep reinforcement learning," *CoRR*, vol. abs/1809.05676, 2018. [Online]. Available: http://arxiv.org/abs/1809.05676

[8] O. E. Gundersen, S. Shamsaliei, and R. Isdahl, "Do machine learning platforms provide out-of-the-box reproducibility?" *Future Generation Computer Systems*, vol. 126, 07 2021.

[9] M. Mora-Cantallops, S. Sanchez-Alonso, E. Garcia-Barriocanal, and M.-A. Sicilia, "Traceability for trustworthy ai: A review of models and tools," *Big Data and Cognitive Computing*, vol. 5, no. 2, 2021. [Online]. Available: https://www.mdpi.com/2504-2289/5/2/20

[10] R. Garcia, V. Sreekanti, N. Yadwadkar, D. Crankshaw, J. E. Gonzalez, and J. M. Hellerstein, "Context: The missing piece in the machine learning lifecycle," *KDD CMI Workshop*, vol. 114, pp. 32–38, 2018.

[11] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2019, pp. 291–300.

[12] P. Langley, "Machine learning as an experimental science," *Machine Learning*, vol. 3, no. 1, pp. 5–8, 1988. [Online]. Available: https://doi.org/10.1023/A:1022623814640

[13] C. Hill, R. Bellamy, T. Erickson, and M. Burnett, "Trials and tribulations of developers of intelligent systems: A field study," in *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2016, pp. 162–170, ISSN: 1943-6106.

[14] M. Vartak, H. Subramanyam, W.-E. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia, "ModelDB: a system for machine learning model management," in *Proceedings of the Workshop on Human-In-the-Loop Data Analytics - HILDA '16*. ACM Press, 2016, pp. 1–3. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2939502.2939516 (Accessed 2022-12-13).

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[16] N. Hewage and D. Meedeniya, "Machine learning operations: A survey on mlops tool support," *CoRR*, vol. abs/2202.10169, 2022. [Online]. Available: https://arxiv.org/abs/2202.10169 (Accessed 2022-12-13).

[17] "2020 State of Enterprise Machine Learning," Algorithmia, Whitepaper, 2020. [Online]. Available: https://info.algorithmia.com/hubfs/2019/Whitepapers/The-State-of-Enterprise-ML-2020/Algorithmia_2020_State_of_Enterprise_ML.pdf (Accessed 15.12.2022).

[18] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe *et al.*, "Accelerating the machine learning lifecycle with mlflow." *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 39–45, 2018.

[19] A. Chen, A. Chow, A. Davidson, A. DCunha, A. Ghodsi, S. A. Hong, A. Konwinski, C. Mewald, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, A. Singh, F. Xie, M. Zaharia, R. Zang, J. Zheng, and C. Zumar, "Developments in mlflow: A system to accelerate the machine learning lifecycle," in *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning*, ser. DEEM'20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3399579.3399867

[20] L. Scotton, "Engineering framework for scalable machine learning operations," Master's thesis, Aalto University. School of Science, 2021. [Online]. Available: http://urn.fi/URN:NBN:fi:aalto-202101311796 (Accessed 2022-12-13).

[21] G. Zárate, R. Miñón, J. Díaz-de Arcaya, and A. I. Torre-Bastida, "K2e: Building mlops environments for governing data and models catalogues while tracking versions," in *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, 2022, pp. 206–209.

[22] J. Pineau, "Reproducibility, reusability, and robustness in deep reinforcement learning," Paper presented at the meeting of ICLR 2018, 2018. [Online]. Available: https://www.youtube.com/watch?v=Vh4H0gOwdIg (Accessed 2022-12-13).

[23] J. Pineau, "The machine learning reproducibility checklist," 2020. [Online]. Available: https://www.cs.mcgill.ca/~jpineau/ReproducibilityChecklist.pdf (Accessed 2022-12-13).

[24] R. Tatman, J. VanderPlas, and S. Dane, "A practical taxonomy of reproducibility for machine learning research," 2nd Reproducibility in Machine Learning Workshop at ICML 2018, Stockholm, Sweden., 2018.

[25] Stack Overflow, "Stack overflow developer survey results 2018," 2018. [Online]. Available: https://insights.stackoverflow.com/survey/2018/#work-_-version-control (Accessed 2022-12-13).

[26] J. Bergstra, D. Yamins, and D. D. Cox, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms," Proceedings of the 12th Python in Science Conference in Science Conference (SCIPY 2013).

[27] M. Tabladillo, A. Arora, and C. Gronlund, "What is the Team Data Science Process?" [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/overview (Accessed 2022-12-13).

[28] Aim. [Online]. Available: https://aimstack.io (Accessed 2022-12-13).

[29] Amazon sagemaker. [Online]. Available: https://aws.amazon.com/sagemaker/features/ (Accessed 2022-12-13).

[30] Azure machine learning. [Online]. Available: https://docs.microsoft.com/de-de/azure/machine-learning/how-to-track-monitor-analyze-runs?tabs=python (Accessed 2022-12-13).

[31] Clearml. [Online]. Available: https://clear.ml (Accessed 2022-12-13).

[32] Comet. [Online]. Available: https://www.comet.ml/site/ (Accessed 2022-12-13).

[33] Dagshub. [Online]. Available: https://dagshub.com (Accessed 2022-12-13).

[34] Dominodatalab. [Online]. Available: https://www.dominodatalab.com (Accessed 2022-12-13).

[35] Guild ai. [Online]. Available: https://guild.ai (Accessed 2022-12-13).

[36] H2o mlops. [Online]. Available: https://www.h2o.ai/products/h2o-mlops/ (Accessed 2022-12-13).

[37] Dvc studio. [Online]. Available: https://studio.iterative.ai (Accessed 2022-12-13).

[38] Mlflow. [Online]. Available: https://mlflow.org (Accessed 2022-12-13).

[39] Neptune. [Online]. Available: https://neptune.ai/product (Accessed 2022-13-10).

[40] Paperspace gradient. [Online]. Available: https://gradient.paperspace.com (Accessed 2022-12-13).

[41] Polyaxon. [Online]. Available: https://polyaxon.com (Accessed 2021-07-31).

[42] Tensorboard. [Online]. Available: https://www.tensorflow.org/tensorboard/ (Accessed 2022-12-13).

[43] Valohai. [Online]. Available: https://valohai.com (Accessed 2022-12-13).

[44] Verta. [Online]. Available: https://www.verta.ai (Accessed 2022-12-13).

[45] Vertex ai. [Online]. Available: https://cloud.google.com/vertex-ai (Accessed 2022-12-13).

[46] Weights & biases. [Online]. Available: https://wandb.ai/site (Accessed 2022-12-13).

[47] Data version control - documentation. [Online]. Available: https://dvc.org/doc (Accessed 2022-12-13).

[48] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe *et al.*, "Accelerating the machine learning lifecycle with mlflow," *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 39–45, 2018.

[49] Pypi mlflow history. [Online]. Available: https://pypi.org/project/mlflow/#history (Accessed 2022-12-13).

[50] Managed mlflow. [Online]. Available: https://www.databricks.com/product/managed-mlflow (Accessed 2022-12-13).

[51] Mlflow. [Online]. Available: https://www.mlflow.org/docs/1.29.0/pipelines.html (Accessed 2022-12-13).

[52] Mlflow documentation. [Online]. Available: https://www.mlflow.org/docs/latest/index.html (Accessed 2022-12-13).

[53] Pycaret logging with mlflow. [Online]. Available: https://pycaret.gitbook.io/docs/get-started/functions/initialize#experiment-logging (Accessed 2022-12-13).

[54] Mlflow projects. [Online]. Available: https://mlflow.org/docs/latest/projects.html (Accessed 2022-12-13).

[55] Pypi neptune client history. [Online]. Available: https://pypi.org/project/neptune-client/#history (Accessed 2022-12-13).

[56] Neptune r client package. [Online]. Available: https://docs.neptune.ai/integrations/r/ (Accessed 2022-12-13).

[57] Neptune - deploying neptune on your server. [Online]. Available: https://docs.neptune.ai/about/on-prem_intro/ (Accessed 2022-12-15).

[58] Neptune-mlflow integration. [Online]. Available: https://docs-legacy.neptune.ai/integrations/mlflow.html (Accessed 2022-12-13).

[59] T.-W. Huang. tensorboardx. [Online]. Available: https://github.com/lanpa/tensorboardX (Accessed 2022-12-13).

[60] Clearml. [Online]. Available: https://clear.ml/docs/latest/docs/webapp/webapp_exp_table/ (Accessed 2022-12-13).

[61] Fast data science. [Online]. Available: https://github.com/DAGsHub/fds (Accessed 2021-07-20).

[62] M. Tkachenko, M. Malyuk, A. Holmanyuk, and N. Liubimov, "Label Studio: Data labeling software," 2020-2022, open source software available from https://github.com/heartexlabs/label-studio. [Online]. Available: https://github.com/heartexlabs/label-studio (Accessed 2022-12-13).

[63] E. Commission. Europe fit for the Digital Age: Commission proposes new rules and actions for excellence and trust in Artificial Intelligence. [Online]. Available: https://ec.europa.eu/commission/presscorner/detail/en/ip_21_1682 (Accessed 2022-12-13).

[64] T. Budras, "Evaluation of machine learning lifecycle tools in the context of a specific nlp project," Bachelor's Thesis, Department of Computer Science and Business Information Systems, University of Applied Sciences Karlsruhe, Germany, 2021. [Online]. Available: https://www.smiffy.de/thesis/thesis-buti1021.pdf (Accessed 2022-12-13).