Comparison of Experiment Tracking Frameworks in Machine Learning Environments

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

***Abstract*—The machine learning market is growing and machine learning is increasingly being used productively. Because of this, more and more tools have been developed in the past with the aim of supporting machine learning in practice. One type of these tools is called *experiment tracking tools*. Their objective is to keep track of the information generated by different experiment runs so that the information can be used later, for example, to find the best experiment run. Within the context of a bachelor thesis, a pre-selection of 20 systems was made and then 4 of them were selected for a more in-depth analysis and their characteristics were examined in more detail. This paper summarizes the most important findings of this thesis.**

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

I. Introduction

The machine learning market is growing strongly. According to MarketsandMarkets [1], 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 [2] uses the term "machine-learning-reproducibility-crisis" to describe that the tools to meet these needs are often not applied 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 [3], in a large amount of scenarios these relatively new tools remain unused and tracking is either done manually or not done at all.

The rest of the paper is structured as follows: In Section II we explain the *machine learning lifecycle* and what artifacts needs to be tracked in the context of an experiment. Based on these findings we present the general architecture for experiment tracking tools and formulate the most important requirements in Section III. In Section IV a number of concrete tools are presented and compared. The papers is finished with a Conclusion 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.

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. [4] 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 model is not the important product of the first phase, but the pipeline, which can be reused to create a model from a data set. This pipeline can be used later in the second phase *training* (middle), to train and validate the model used for inference. The last phase (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. The prediction service provides information on the made predictions, which can be used for later trainings. The authors mention that the different stages are often managed by different teams.

B. Experiment Tracking

Langley [5] 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. [6] conducted

 ISBN: 978-1-61208-969-0

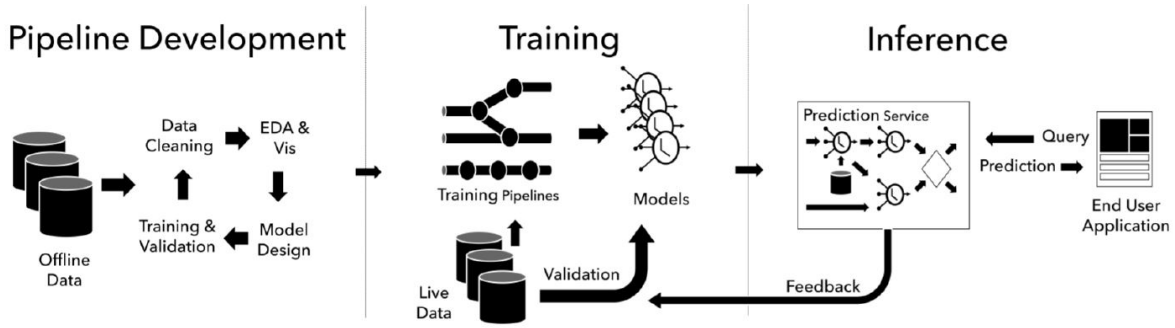

Fig. 1. Machine Learning Lifecycle (from [4])

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 [5]. Hence, an experiment consists of multiple runs. According to Vartak et al. [7], "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. [4], 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 a(n experiment) run 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 resulting 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 [8].

Various possibilities exist to assess the quality of a model. A common possibility is to calculate a metric for prediction quality (such as accuracy) on a data set 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. 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. 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* [9]. Pineau also published a machine learning reproducibility checklist [10], which is supposed to help increase the reproducibility of experiments. Tatman et al. [11] 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. [11] 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 [12] 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.

 ISBN: 978-1-61208-969-0

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. [11] 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. [13], 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" [8]. 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 [8]. Metrics can be used to compare different runs of an experiment and can be considered as one of the dependent variables of the experiment. Which type of metric is used, is not important regarding 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 2. 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. [14], 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 the 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 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 [15]
- Amazon SageMaker Experiments [16]
- Azure Machine Learning [17]
- ClearML [18]
- Comet [19]
- DAGsHub [20]
- DominoDataLab [21]
- DVC Studio [22]
- Guild AI [23]
- H2O MLOps [24]

 ISBN: 978-1-61208-969-0

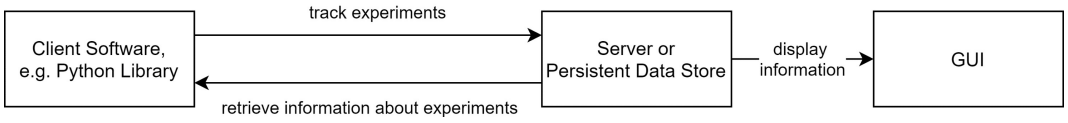

Fig. 2. General Architecture of an Experiment-Tracking-Tool

- MLflow [25]
- Neptune [26]
- Paperspace Gradient [27]
- Polyaxon [28]
- Sacred [28] in combination with Omniboard, Incense or Sacredboard (GUIs)
- TensorBoard [29]
- Valohai [30]
- Verta [31]
- Vertex AI [32]
- Weights & Biases [33]

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. 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) [34] 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.

```
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)
```

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, 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. MLflow provides functions to log one value (line 9), or to log multiple values (here, a dictionary is passed, as the only parameter and the name and values of the dictionary will be used (line 6). Grouping multiple runs together allows easy viewing and comparison in the GUI. This can be achieved by setting up an experiment (line 3). Metadata (params, metrics, etc.) are by default stored in a local text file. However, other possibilities exist; such as saving them in a SQL Database, which can be achieved by specifying a tracking URI (line 2). By default, models logged with MLflow are stored in the local file system. However, it is possible to change the location, e.g., to an S3 bucket.

The MLflow GUI in Figure 3 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 support automated tracking of the environment either. It 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. While the Client Software (Python package) is open-source, the server code is not publicly available. Free as well as paid plans exist. 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. After those setup steps, Neptune is ready for use.

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

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 dict (line 9). To track series such as the loss, the log function has to be used (line 6). This automatically generates a plot in the GUI. To upload a trained model, it first has to

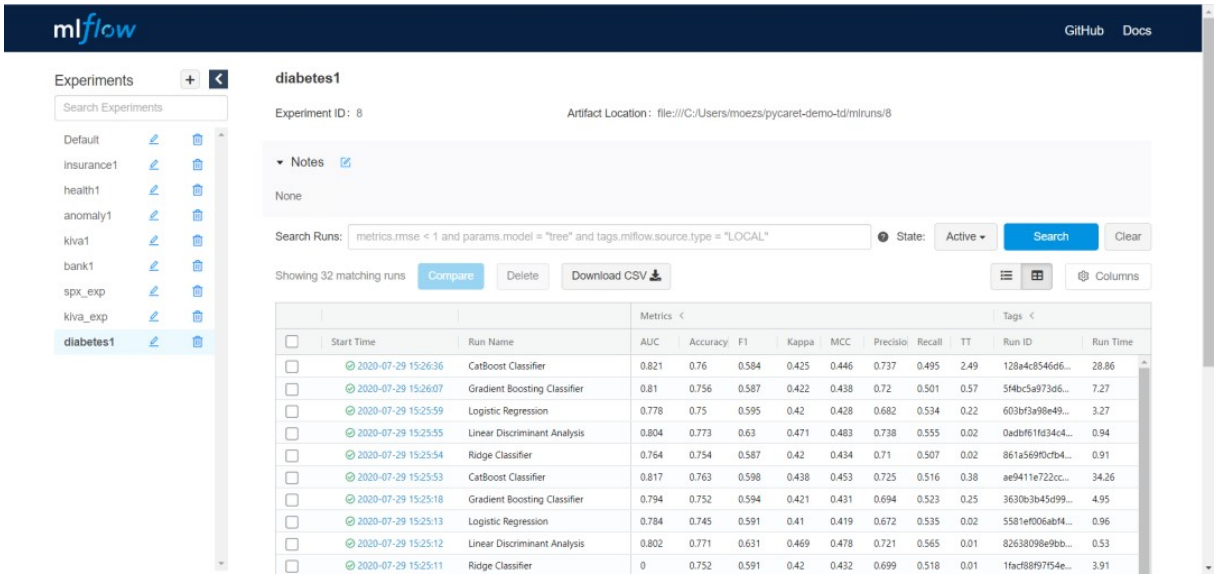

Fig. 3. MLflow GUI (from [35])

be saved locally and can then be uploaded to Neptune using the upload (line 7).

The GUI of Neptune (Figure 4) 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. The data can also be retrieved through the Python API. Similar to MLflow, Neptune's focus is tracking metrics and hyperparameters. The setup is easier as with MLflow, however, using Neptune raises Data Governance questions, because data is stored on Neptune servers, outside your own company. For single users Neptune can be used for free. When working in teams the prize is calculated based on the usage.

C. ClearML

ClearML is an open-source tool developed by Allegro AI, it was formerly known as Allegro Trains. Multiple options to operate ClearML exist, it can be self-managed for free, used with a free as-a-Service plan for up to three team members or used with a paid plan.

```
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", task_name
     ="Task1", reuse_last_task_id=False, output_uri="
     gs://MyProject",)
4 hyperparams = {"lr": 0.01,}
5 task.connect(hyperparams)
6 #Training placeholder, model stored in var model
7 torch.save(model, "log_r.mdl")
8 task.get_logger().report_scalar("model", "accuracy",
      0.99, 0)
```

Listing 3. ClearML example code

Besides its hyperparameter and metric tracking capabilities, ClearML provides a possibility to efficiently store and manage large datasets. It works similar to DVC [34]. 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. In ClearML, a task is similar to a run in MLflow and describes something that is executed and should be tracked. In line 3, a task is initialized and assigned to a project. Setting `reuse_last_task_id` to False ensures that this task will not override an old task. The `output_uri` specifies the location for the artifacts (e.g., the model) and is in this example set to a Google Cloud Storage. By initializing a task, the tracking is automatically started. ClearML allows logging hyperparameters by connecting an object to a task (line 5). When a model is saved locally, ClearML automatically uploads it to the artifact store and connects it to the task (line 7). Metrics can be reported to a logger, where the first argument is the title of the plot, the second is the name of the series, the third is the value and the last is the iteration (x-coordinate). It should be noted that executing `Task.init` automatically tracks the used python packages and their versions, providing an additional amount of information.

Figure 5 shows a screenshot of the GUI. 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. Besides the Python API data collected with ClearML can also be retrieved with a REST API.

 ISBN: 978-1-61208-969-0

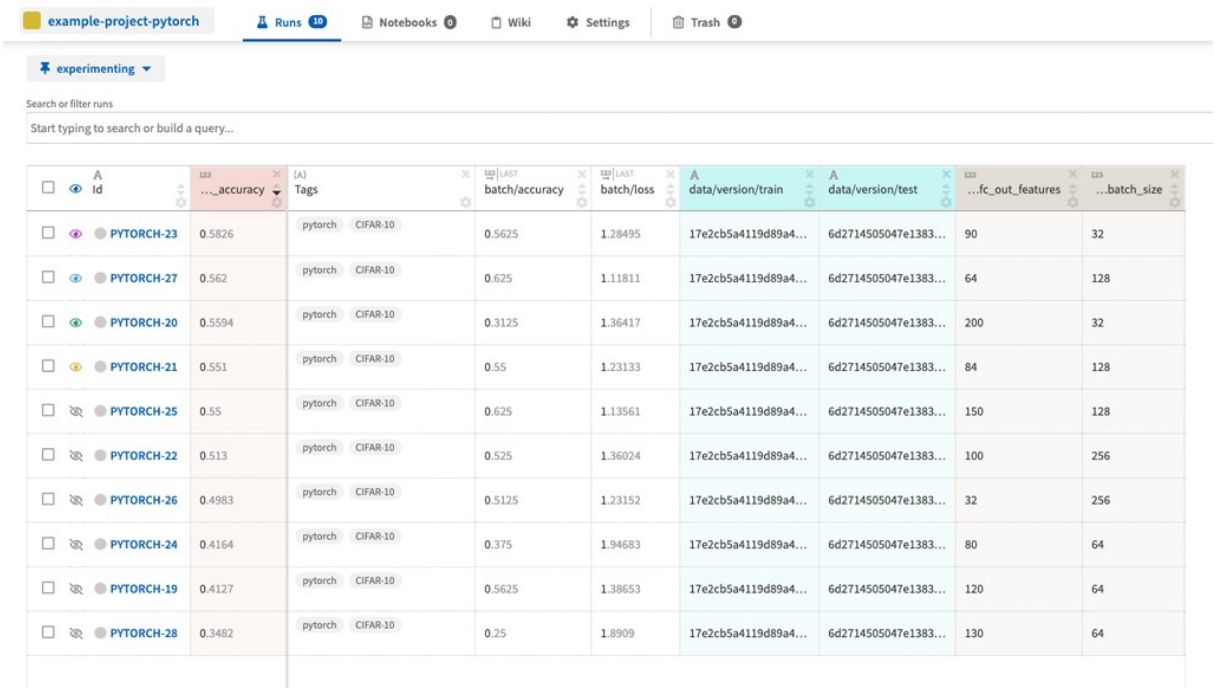

Fig. 4. Neptune GUI (from [26])

D. DAGsHub

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

- DVC [34] 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 6.

Beside DAGSHub's own client, MLFlow can be used to track hyperparameters and metrics. In this case the integration into code looks like in Listing 1. The most important advantages of DAGsHub are the unified storage and the efficient handling of variants of datasets using DVC.

The GUI of DAGsHub in Figure 7 is familiar to GitHub users, but additionally includes a data section, as well as an overview of the experiment runs as known from MLflow. 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.

E. Comparison

Table IV-E 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. Neptune, on the other hand, offers a simple setup and highly functional GUI but requires a paid plan when used as a team. 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. 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

This paper showed the benefits of tracking machine learning experiments. After presenting 20 tools with functionalities to track experiments which have been identified in a market research. Requirements for machine learning experiment tools were defined based on the needs of an industrial data science project and 4 tools have been evaluated in detail. This evaluation has shown that the right choice of an experiment tracking tool depends on the specific requirements, and identified ClearML as an open source tool that meets most of the requirements.

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.

 ISBN: 978-1-61208-969-0

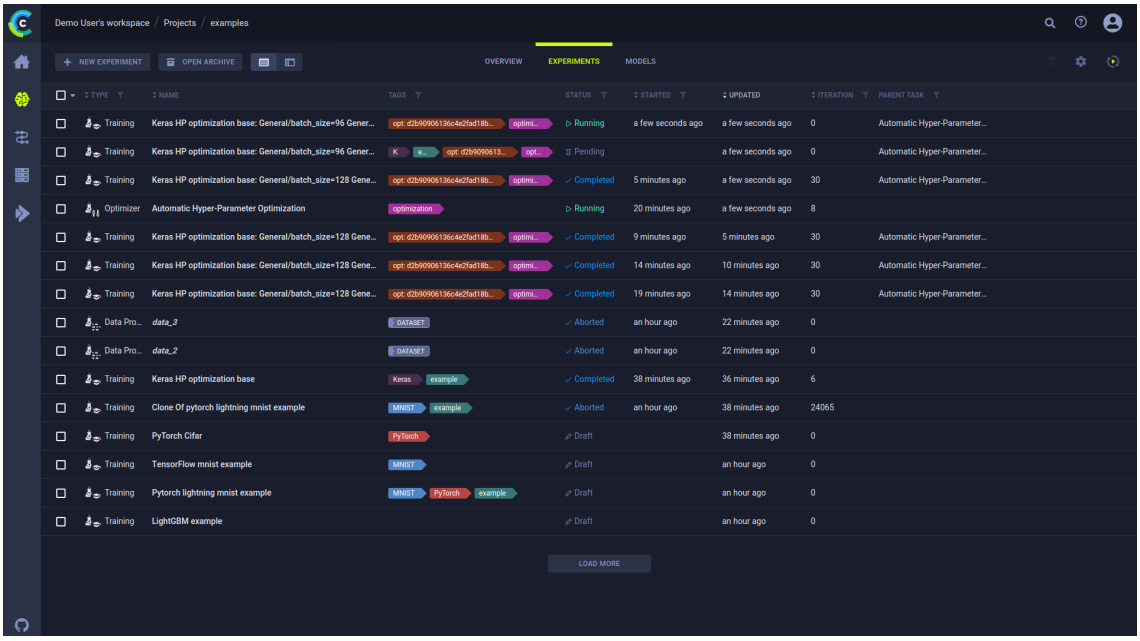

Fig. 5. ClearML GUI (from [36])

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

	MLflow	**Neptune**	**ClearML**	**DAGsHub**
Evaluated version	1.17	0.9.18	1.0	as of June 2021
Data	no dedicated functionality provided	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
Accessibility of tracked information	basic GUI as well as Python, R, Java and REST APIs	highly customizable & advanced GUI as well as a Python API	advanced GUI as well as a Python API	unified GUI for data, code, and experiments, no Python API for retrieving experiment data
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	easy setup, as the user does not have to take care of the infrastructure	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

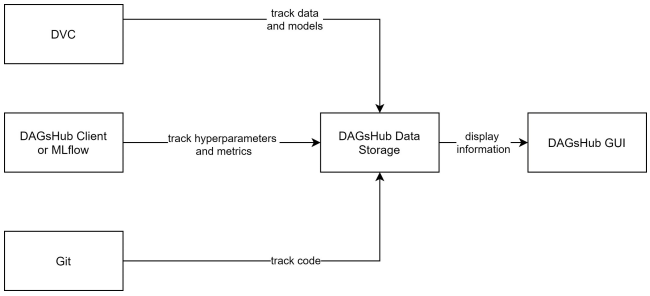

Fig. 6. DAGsHub Architecture

ACKNOWLEDGMENT

The work was carried out in the course of the bachelor thesis [37] of the first author at the company inovex GmbH.

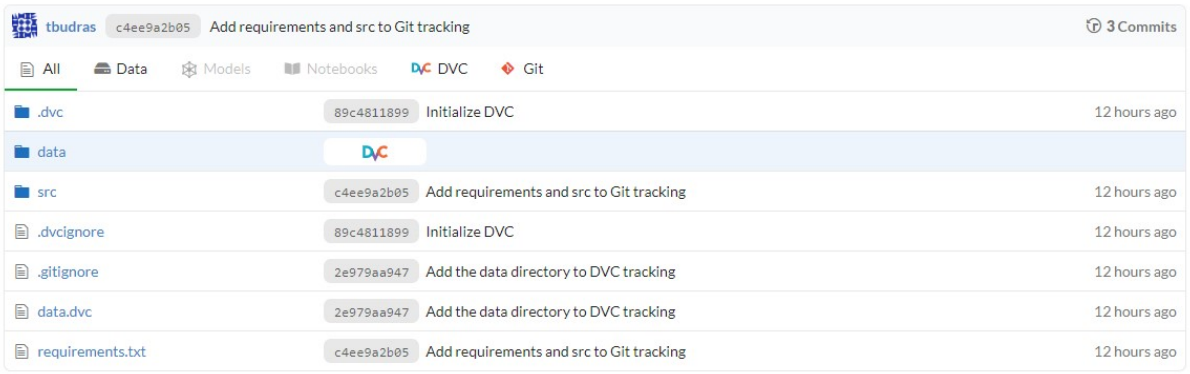

Fig. 7. DAGsHub GUI

REFERENCES

[1] Machine learning market. [Online]. Available: https://www.marketsandmarkets.com/Market-Reports/machine-learning-market-263397704.html (Accessed 2021-07-19).

[2] P. Warden. The machine learning reproducibility crisis. [Online]. Available: https://petewarden.com/2018/03/19/the-machine-learning-reproducibility-crisis/ (Accessed 2021-03-11).

[3] State of data science and machine learning 2020. [Online]. Available: https://www.kaggle.com/kaggle-survey-2020 (Accessed 2021-03-17).

[4] 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.

[5] 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

[6] 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.

[7] 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 2021-03-10).

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

[9] 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

[10] J. Pineau, "The machine learning reproducibility checklist," 2020. [Online]. Available: https://www.cs.mcgill.ca/~jpineau/ReproducibilityChecklist.pdf

[11] 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.

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

[13] 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).

[14] 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 2021-05-10).

[15] Aim. [Online]. Available: https://aimstack.io (Accessed 2021-07-31).

[16] Amazon sagemaker. [Online]. Available: https://aws.amazon.com/sagemaker/features/ (Accessed 2021-07-31).

[17] Azure machine learning. [Online]. Available: https://docs.microsoft.com/de-de/azure/machine-learning/how-to-track-monitor-analyze-runs?tabs=python (Accessed 2021-07-31).

[18] Clearml. [Online]. Available: https://clear.ml (Accessed 2021-05-11).

[19] Comet. [Online]. Available: https://www.comet.ml/site/ (Accessed 2021-07-31).

[20] Dagshub. [Online]. Available: https://dagshub.com (Accessed 2021-07-31).

[21] Dominodatalab. [Online]. Available: https://www.dominodatalab.com (Accessed 2021-07-31).

[22] Dvc studio. [Online]. Available: https://studio.iterative.ai (Accessed 2021-07-31).

[23] Guild ai. [Online]. Available: https://guild.ai (Accessed 2021-07-31).

[24] H2o mlops. [Online]. Available: https://www.h2o.ai/products/h2o-mlops/ (Accessed 2021-07-31).

[25] Mlflow. [Online]. Available: https://mlflow.org (Accessed 2021-07-31).

[26] Neptune. [Online]. Available: https://neptune.ai/product (Accessed 2022-04-07).

[27] Paperspace gradient. [Online]. Available: https://gradient.paperspace.com (Accessed 2021-07-31).

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

[29] Tensorboard. [Online]. Available: https://www.tensorflow.org/tensorboard/ (Accessed 2021-07-31).

[30] Valohai. [Online]. Available: https://valohai.com (Accessed 2021-07-31).

[31] Verta. [Online]. Available: https://www.verta.ai (Accessed 2021-07-31).

[32] Vertex ai. [Online]. Available: https://cloud.google.com/vertex-ai (Accessed 2021-07-31).

[33] Weights & biases. [Online]. Available: https://wandb.ai/site (Accessed 2021-07-31).

[34] Data version control - documentation. [Online]. Available: https://dvc.org/doc (Accessed 2022-03-12).

[35] Pycaret logging with mlflow. [Online]. Available: https://pycaret.gitbook.io/docs/get-started/functions/initialize#experiment-logging (Accessed 2022-04-06).

[36] Clearml. [Online]. Available: https://clear.ml/docs/latest/docs/webapp/webapp_exp_table/ (Accessed 2022-04-06).

[37] 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 2022-04-08).

 ISBN: 978-1-61208-969-0