Course-ID:	MA-INF 4306
Course:	Frameworks for Reproducible Machine Learning
Term:	Winter Semester 2020/2021
Supervisor(s):	Dr. Daniel Trabold

UNIVERSITÄT BONN ML AI lab

Frameworks for Reproducible Machine Learning

Hanna Kondratiuk Anna Rasgauski

April 20, 2021

Abstract

Producing reproducible results is paramount in scientific research, but this can be a challenge in the field of machine learning. In this work we compare three frameworks which assist with the reproduction of results in machine learning: Weights and Biases, Guild AI and MLflow. We empirically compare their performance on a number of experiments and derive a decision tree which takes the required features for a machine learning problem as an input for the query and returns the framework which would be most suited to the task. We also look at their logging capabilities and their collaboration support. Moreover, we compare the definitions of reproducibility and replicability in the context of neural networks and consider the possibility of *exactly* reproducing the results of our experiments with neural networks using the Docker framework.

1 Introduction

Reproducibility in machine learning is said to be in crisis (Hutson, 2018). Producing reproducible results is paramount in scientific research, but this can be difficult to achieve in the field of machine learning. To take a closer look at this, Papers with Code (*Papers with Code* n.d.) has been launching Reproducibility Challenges since 2018. During these challenges participants are to select a paper submitted to a conference (in 2018 it was the ICLR conference) and try to replicate the experiments described in the paper. The goal of the Reproducibility Challenge is two folded: on the one hand, it will help the authors improve the quality of their work and papers. On the other hand, it will allow the community to assess the issues of reproducibility in machine learning with the help of some of the most well-known machine learning conferences.

In this section we start by discussing the definitions of reproducibility and replicability. We then have a first glance at the frameworks we compare in this report.

1.1 Reproducibility vs. Replicability

In a review paper on the use of the terms reproducibility and replicability (Barba, 2018), three categories of usage were outlined, which were characterised as **A**, **B1** and **B2**. **A** was referring to the case when the terms are used with no distinction between them, **B1** to *reproducibility* as instances in which the original researcher's data and computer codes are used to regenerate the results, while *replicability* refers to instances in which a researcher

collects new data to arrive at the same scientific findings as a previous study. In **B2** *reproducibility* refers to independent researchers arriving at the same results using their own data and methods, while *replicability* refers to a different team arriving at the same results using the original author's artifacts.

We observe that the definitions contradict each other. A study conducted by the National Academies of Sciences, Engineering, and Medicine (National Academies of Sciences and Medicine, 2019) to investigate reproducibility and replicability in science adopted the following definitions, after extensive review: *Reproducibility* is obtaining consistent results using the same input data, computational steps, methods and code; and conditions of analysis. *Replicability* is obtaining consistent results across studies aimed at answering the same scientific question, each of which has obtained its own data. This final definition of reproducibility is similar to definition **B1** above and this definition **B1** is the definition of reproducibility, which we will be referring to and striving to in this report.

However, most deep learning models are likely to fail when striving for exact reproduction, most likely due to randomness in the model initialisation and optimisation, unlike many other machine learning models (Li and Talwalkar, 2020; Liu et al., 2020). Thus, for deep learning studies, we need to consider reproducibility under a broader definition: whether a reported experimental result can be approximately reproduced to a high probability with the same model and the same data (Liu et al., 2020). For a wide variety of other machine learning models the probability of reproduction is equal to 1. In Section 3 we delve even deeper into reproducibility and attempt exact reproduction of models using randomness.

1.2 Overview of the Frameworks

Reproducibility was also discussed at the ECML PKDD 2020 conference; reproducibility assisting frameworks which came to the forefront at this conference were Weights and Biases (W&B) (Biewald, 2020), Guild AI (Guild) (*Guild AI* 2020) and MLflow (Zaharia et al., 2018). These frameworks aim to track and monitor the machine learning cycle and support the user during optimisation and tuning of these models. Mlflow supports a very large number of libraries and frameworks, and can be used both locally and in collaboration with others. W&B also offers a wide range of supported libraries and frameworks, and focuses on visualisation. The default setting of W&B is to work using a remote server and, unlike the other two frameworks, is not open source. Guild aims to monitor the machine learning cycle without needing to change any code at all. With its integration of Tensorboard it also allows for easy visualisation.

In this report we take a deeper look at these frameworks and conduct an empirical comparison on them in the context of reproducibility. To do this we attempt to reproduce the same train and test metrics and artifacts during the test phase for both deterministic and randomised algorithms. The experiments and findings of this empirical comparison are discussed in Section 2. Moreover, we address the issue of reproducibility of results of convolutional neural networks and provide a Docker image to reproduce exactly the same inference results for one of our experiments across different machines. This we discuss in Section 3.

2 Experimental Comparison of the Frameworks

In this section we talk about the experiments we conducted with the frameworks to investigate what the frameworks can do and to summarise this is in a formal way. Our

experiments are structured as follows:

1. Scope of Auto-logging
2. Investigating Components of a Run (Initialisation, Auto-Logging and Estimator Parameter Logging)
    - computational time
    - memory consumption of computation
    - number of lines
3. Deep Dive on Reproducibility Using Sklearn
4. Deep Dive on Reproducibility Using Keras

In the final subsection of the experiments we turn our attention to the collaboration aspect of the frameworks. We then summarise the findings of our experimental comparison of the frameworks.

2.1 Scope of Auto-Logging

Each of the frameworks come equipped with varying degrees of auto-logging. This default implementation of the framework allows the framework to be used with minimum change to the code. Auto-logging is not the same across the three frameworks we are comparing, but will save some information about the run. This information can be system information, parameters regarding the model, metrics etc. The purpose of this experiment is to investigate to what extent the frameworks can be used with auto-logging to support reproducibility and which parameters are logged by default by each framework.

Setup: We decided to implement our first experiment using Python, sklearn and Jupyter notebooks, as this is an often seen combination for machine learning experiments. This also enabled a fair comparison as all of the frameworks support these.

We implement a ridge regression classifier and a support vector machine (SVM) using the Boston Housing Dataset (*StatLib archive* 2020) and the Iris Dataset (*UCI repository of machine learning databases* 2020) respectively. We also implement a random forest, k-means clustering and a naive Bayes classifier model, but we do not analyse their performance in the experiment and decide to focus instead on the ridge regression and the SVM.

Steps: Using the classifiers we have implemented, we start by looking at which run-initialisation parameters get saved. This is information such as run id, start and stop time, status of run and so on. We look at whether all estimator parameters are saved, even when these are not explicitly defined and only use default values. We also investigate which training and test metrics get saved and whether or not the sklearn version is recorded.

Results: The results of this experiment can be viewed in Table 1. We can see that MLflow performs well across all of the categories. We find only test metrics to not be recorded by default by MLflow. If we use default values for our model we are only able to record these for W&B and Guild if we use a slight workaround and use a dictionary structure. As often the case with Guild, for training and test metrics Guild requires the information to be printed in a particular form. W&B does not save training and test metrics by default.

2.2 Investigating Components of a Run: Initialisation, Estimator Parameter Logging and Auto-Logging

The purpose of this experiment is to get an idea of the computation time and memory needed for each framework and whether there are any major differences amongst the frameworks. In order to assess the ease of implementation of these frameworks, we also take a look at the number of lines each framework requires to implement each component.

Setup: The setup of this experiment is the same as the previous experiment. See 2.1.

Steps: In order to measure computation time for each of the frameworks, we start by dividing a run into three components: *initialisation*, *estimator parameter logging* and *auto-logging*. *Initialisation* comprises of starting and stopping a run using the framework. For *initialisation* we do not create or run any machine learning models and we note that during *initialisation* we include the framework's import and initialisation of the experiment. We define *estimator parameter logging* as the process of logging any custom parameter of the model desired by the user. *Auto-logging* is the default setting of the frameworks for recording a run and was discussed in Section 2.1.

Once we had decided upon our components we set about implementing them. We then measure the required computation time of each framework using Ipython's magic *timeit* function. Here we run each of the components 100 times for each framework. We use the function *memit* from the Python module (*Memory Profiler* 2020) to take a look at the memory required for computation and run each component 10 times for each framework.

Results: The results of the computation time of this experiment can be seen in Table 2. The table contains the three aforementioned components: initialisation, estimator parameters and auto-logging. The experiment was conducted on two machines with the names MacBook and Dell respectively, the results are however following the same pattern across both machines. It can be seen that Guild performs best here and requires substantially less time for initialisation than the other two frameworks. W&B requires a large amount of time for initialisation, this is most probably due to it connecting to the remote server. W&B does also offer auto-logging like the other two frameworks, however, in a lower capacity in the scope of the sklearn framework, as can be observed in Table 1. We may also observe in Table 1, that MLflow logs a wide range of parameters automatically. This

	W&B	**Guild**	**MLflow**
Run-Initialisation Parameters	+	+	+
Estimator Default Parameters	when passed as dictionary to wandb.config	when parameters are global or passed into run function	recorded when fit function of estimator is called
Training Metrics	when printed	when printed in specific form	automatically
Test Metrics	-	when printed in specific form	-
sklearn Version	+	-	+
Score*	**3**	**2.5**	**4**

Table 1: **Auto-Logging** results of 2.1. +: full integrated support, -: no support (**Score: 1 point for "+", 0.5 points if workaround is possible, 0 points for "-"*)

	Initialisation		Estimator Parameters		Auto-Logging		Score*
sklearn Algorithm	*		Ridge Regression		SVM		
Dataset	*		Boston Housing		Iris		
Hardware	MacBook	Dell	MacBook	Dell	MacBook	Dell	
W&B	3.74 s ± 0.96s	4.54 s ± 368 ms	3.45 ms ± 283 µs	1.86 ms ± 109 µs	4.71 s ± 0.98 s	4.10 s ± 612.83 ms	**1**
Guild	3.76 ms ± 261 µs	1.15 ms ± 39.7 µs	77.8 ms ± 1 ms	40.8 ms ± 697 µs	730 ms ± 7.92 ms	40.5 ms ± 568 µs	**2.5**
MLflow	87.7 ms ± 4.64 ms	65.8 ms ± 1.1 ms	403 ms ± 72 ms	168 ms ± 2.22 ms	755 ms ± 12.7 ms	197 ms ± 3.67 ms	**1**

Table 2: **Runtime** results for 2.2. *Score: 1 point for best, 0.5 points for second best, 0 points otherwise (for each run component and for both systems)*

	Initialisation		Estimator Parameters		Auto-Logging		Score*
sklearn Algorithm	*		Ridge Regression		SVM		
Dataset	*		Boston Housing		Iris		
Hardware	MacBook	Dell	MacBook	Dell	MacBook	Dell	
W&B	98.67MiB, +12.62MiB	83.63MiB, +3.28MiB	149.34MiB, -0.09MiB	146.64MiB, +1.84MiB	130.2MiB, +12.71MiB	128.66MiB, +2.91MiB	**1**
Guild	88.03MiB, +0.14MiB	90.81MiB, +0.41MiB	119.59MiB, -0.09MiB	131.57MiB, +2.10MiB	119.88MiB, +0.29MiB	128.87MiB, +0.50MiB	**1.5**
MLflow	104.41MiB +2.62MiB	107.24MiB +2.85 MiB	151.0MiB +2.25MiB	164.7MiB +3.96 MiB	174.0MiB +25.5MiB	162.3MiB, +3.33MiB	**0.5**

Table 3: **Memory** results for 2.2, in the form of peak memory and increment. **Score: ranking of the frameworks across all run components with 0.5 point increments*

corresponds to a worse runtime shown in Table 2.

We then turn our attention to the computational memory required for each of our components. The findings of this can be viewed in Table 3. MLflow is prone to require more memory. This can however be expected as it also saves more parameters than Guild and W&B. The best memory results are being shown by Guild, this may be a result of the framework's simplicity.

For each of the components we also look at the number of lines required. A framework requiring a large number of lines to implement requires larger changes to the code and may not be as straightforward to integrate into the run. The results of this can be viewed in Table 4. We can see that Guild requires the most number of lines for initialisation. For Guild in a Jupyter notebook we have to rewrite the code into a function and then call it using Guild. Each of the frameworks enable single line auto-logging. For W&B the auto-logging of the parameters is performed automatically with initialisation, however the integration of auto-logging with sklearn directly is not presented in any way. It can be seen in the table that MLflow requires the least number of extra lines across all components.

2.3 Deep Dive on Reproducibility Using Sklearn

Inspired by the information logged by default in W&B, Guild and MLflow we take the union of these and add extra parameters which allow for better reproducibility. Using this information we decide upon a list of criteria which we find important for reproducibility. We then also look at the log size required to save this criteria for each framework.

Component of Experiment	W&B	Guild	MLflow
Initialisation	2	3	2
Estimator Parameters	2	0	0
Auto-Logging	(0)	1	1
Score*	**2**	**2**	**3**

Table 4: Additional **Number of lines** required for each component of a sklearn experiment for 2.2. (**Score: 1 point for best score, 0 points otherwise*)

Setup: As in the previous two experiments we use Python, sklearn and a Jupyter notebook. We are however not restricted to our three components and used the full scope of the frameworks to implement our criteria.

Steps: Based upon the information we had gained from the previous experiments we start by considering our criteria for reproducibility and decide upon the following: System and Hardware, Saves Code, Saves Classifier Parameters, Saves Non-Estimator Parameters, Saves Metrics, Saves Train-Test Data, saves Path to Data from Root, saves Environment and Environments Variables, saves the Matplotlib Plot and saves the Trained Model (.pkl-file).

For our deep dive we implement the frameworks to save each of the items in our criteria, when possible, for a ridge regression classifier.

Results: The results of this deep dive can be viewed in Table 5. In the table we "grade" the framework "+" if the information is logged automatically or easy to add, "(+)" if there is a workaround to achieve the criteria and "-" if it is not possible. We notice that each of the frameworks saves the training and testing metrics, the Python version and the envi-

	W&B	Guild	MLflow
System and Hardware**	+	+	(+)
Code Saving**	+	+	(+)
Estimator Parameter Saving	(+)	(+)	+
Non-Estimator Parameter Saving**	+	(+)	+
Metrics (Train, Test)	+	+	+
Train, Test data	(+)	+	(+)
Path to Data From Root**	+	(+)	+
Environment and Environment Variables**	+	+	+
Matplotlib Plot	+	+	+
Trained Model	(+)	(+)	+
Score*	**8.5**	**8**	**8.5**

Table 5: **Deep Dive with Sklearn** accompanying Experiment 2.3. ** is relevant for Table 2.4. +: full integrated support, (+): can be integrated with workaround. (**Score: 1 point for "+", 0.5 points for "(+)"*)

Framework	W&B		Guild		MLflow	
Hardware	MacBook	Dell	MacBook	Dell	MacBook	Dell
Log size	71.6 kB	67.3 kB	66.5 kB	65.9 kB	88.8 kB	82.8 kB
Score*	**1**		**1**		**1**	

Table 6: Results from Experiment 2.3 featuring **Log size**. We observe only slight differences in the log size, thus **Score: 1 point for each framework*

Layer	Type	Output Shape	Parameters
conv2d	Conv2D	(None, 26, 26, 32)	320
max pooling2d	MaxPooling2D	(None, 13, 13, 32)	0
dropout	Dropout	(None, 13, 13, 32)	0
$conv2d_1$	Conv2D	(None, 11, 11, 64)	18496
$max\ pooling2d_1$	MaxPooling2D	(None, 5, 5, 64)	0
$dropout_1$	Dropout	(None, 5, 5, 64)	0
flatten	Flatten	(None, 1600)	0
dense	Dense	(None, 256)	409856
$dense_1$	Dense	(None, 10)	2570

Table 7: MNIST CNN Structure

ronment dependencies. Although the frameworks have different strengths, across all of the criteria all frameworks achieve a similar score.

When running a machine learning experiment thousands of times the size of the log will make a difference. To this end we looked at the log size for each framework. The results of this can be found in Table 6. Here Guild has the smallest log size, but as we are looking at a difference in log size across the frameworks of a few kilobytes we score all the frameworks equally.

2.4 Deep Dive on Reproducibility Using Keras

Our goal of this experiment is to look at Keras and introduce a convolutional neural network (CNN) in terms of frameworks comparison. We investigate how well the frameworks perform with the Keras library. We extend our previously discussed deep dive and show to what extent the frameworks can record information from a CNN. Using this same setup we then investigate the memory load depending on the size of the data set.

Setup: As with the previous experiments we also use Python and a Jupyter notebook. We implement a simple CNN in Keras on the MNIST data with 60000 images as the training data, 10000 images as test data and a validation split of 0.2 to classify the 10 digits. The network uses the adam optimiser and the relu activation function. Moreover, we set the seed to the same value, so the training on one machine would perform exactly the same way. The structure of the implemented CNN can be viewed in Table 7.

Steps: We adjust the code so that the training, learning and testing process would be

	W&B	Guild	MLflow
Model Parameter Saving (i.e. Epoch Number)	-	-	+
Metrics: Train, Validation Test Accuracy and Loss	+	+	+
Model Summary	+	(+)	+
Trained Model	+	(+)	+
Score*	**8**	**6**	**8**

Table 8: **Deep Dive with Keras** accompanying Experiment 2.4. The results for criteria with ** in Table 5 were also found to be identical for the current experiment, as they do not depend on the library in question (Keras vs. sklearn). +: full integrated support, (+): can be integrated with a workaround, -: no support (**Score: 1 point for "+", 0.5 points for "(+)", 0 points for "-"*)

Data size	10000		20000		30000		60000		Score*
Hardware	MacBook	Dell	MacBook	Dell	MacBook	Dell	MacBook	Dell	
W&B	941MiB, +684MiB	1193MiB, +873MiB	961MiB, +676MiB	1224MiB, +904MiB	993MiB, +708MiB	1263MiB, +943MiB	1127MiB, +842MiB	1420MiB, +1099MiB	**1**
Guild	1019MiB, +753MiB	1176MiB, +868MiB	1055MiB, +790MiB	1216MiB, +908MiB	1195MiB, +929MiB	1285MiB, +979MiB	1376MiB, +1110MiB	1416MiB, +1111MiB	**1**
MLflow	1076MiB, +804MiB	1204MiB, +887MiB	1120MiB, +848MiB	1270MiB, +950MiB	1092MiB, +819MiB	1296MiB, +976MiB	1363MiB, +1090MiB	1440MiB, +1121MiB	**1**

Table 9: **Memory from Data** accompanying Experiment 2.4 featuring memory, in the form of peak memory and increment. As the memory values do not differ much: **Score: 1 point for each framework*

recorded by the frameworks and so each of the frameworks implements the criteria discussed in the previous deep dive (if possible). The comparison of the memory results of the different size data sets is once again achieved using the *memit* cell function.

Results: The results to this experiment can be seen in Table 8. Results in Table 5 marked with ** also apply to this experiment. As before, we assign a framework "+" for a criteria if the information is logged automatically or easy to add, "(+)" if an easy workaround exists and "-" if it is not possible to implement. Unlike the previous results with sklearn, not all of the parameters are possible to save in W&B and Guild, namely, the parameters of the model, such as learning rate, epoch number and so on. Unless explicitly specified in the initialisation, these cannot be saved or easily retrieved from the model.

In Table 9 we look at the memory load for each framework depending on the size of the data. We have data splits of 10000, 20000, 30000 and 60000 of 60000 images from the MNIST data for training, with a validation split of 0.2 respectively. We can see that MLflow shows the same trend as in Table 3, that is, it requires a greater amount of memory. We ran the experiment only once, this needs to be considered when deriving the results further. However, we can clearly see the trend that W&B is performing generally better, with the second best being Guild. We find the differences in memory size to not be significant enough to assign different scores.

2.5 Collaboration

When looking at reproducibility one of the concepts that may not be be overlooked is collaboration. Our comparison of collaboration is summarized in Table 10. We observe that Guild does not prioritise collaboration, while W&B and MLflow allow for various options and DataBricks integration. Moreover, the valuable difference is that W&B needs the enterprise version or subscription plan for collaboration team members, while the

Collaboration Criteria	W&B	Guild	MLflow
Local File Storage	+	+	+
Scalable Cloud Env.	(+)	-	+
http Tracking Server (local)	+	+	+
Hosting Remotely on User Server	+	-	+
Databricks	+	-	+
Integrated API on Remote Framework Server	(+)	-	-
Score*	**5**	**2**	**5**

Table 10: **Collaboration**. +: full integrated support, (+): requires payment, -: no support (**Score: 1 point for "+", 0.5 points for "(+)", 0 points for "-"*)

Criteria	W&B	Guild	MLflow
Auto-Logging	3	2.5	4
Runtime	1	2.5	1
Memory	1	1.5	0.5
No. Lines	2	2	3
Deep Dive Sklearn	8.5	8	8.5
Log Size	1	1	1
Deep Dive Keras	8	6	8
Memory from Data	1	1	1
Collaboration	5	2	5
Score*	**30.5**	**26.5**	**32**

Table 11: Summary of Findings of Framework Comparison. (*Score* taken from tables 1-10 excluding Table 7*)

clear advantage of MLflow is it being free and open source.

2.6 Summary of Findings of Framework Comparison

From our conducted experiments and the respective scores assigned to each experiment, we summarise our findings of the framework comparison in Table 11. This is the concise summary of the high-level concepts based upon our experiments. In this table we can see that all frameworks perform well with MLflow performing marginally better than W&B. If reproducibility is the goal then MLflow and W&B would be more suited to the task. Guild does not offer the same level of collaboration and automatic saving of model parameters as the other two frameworks, but if runtime and memory is a priority for the user, then Guild would be a good choice. The supporting decision trees to visualise our findings are available in the appendix.

3 Reproducibility With Models Using Randomness

We have conducted additional computations using the basis of the experiment discussed in Section 2.4. We take a look at the inference results of the CNN presented in Table 7 and investigate, whether it is possible to *exactly* reproduce the inference metrics. When setting the 'PYTHONHASHSEED' environment variable to a fixed value, the Python, numpy and Tensorflow built-in pseudo-random generators, as well as configuring a new global Tensorflow session, we are able to reproduce the results of all steps of the experiment, training, validation and testing on one machine. This can be viewed at (*Replication with fixed seed* 2020).

However, it is only possible to reproduce the inference across one machine. Thus we investigate, how different the metrics (train, validation and inference) would be depending on the seed value. And, on the other hand, while running the CNN with the same seeds on two different computers, how different the statistics would be. We ran our experiments on different seeds ranging from 0 to 99. For comparison, we define the range statistics as

$$\text{range} = \max_{seed \in \{0,\ldots,99\}} (\text{metric}) - \min_{seed \in \{0,\ldots,99\}} (\text{metric})$$

The statistical summary for metrics of two different laptops can be seen in Table 12. We compute the mean, standard deviation and the above defined range statistic for both computers as well as the absolute difference between those statistics in row "$|a-b|$". From

Statistic	Measure	Train loss	Train accuracy	Validation loss	Validation accuracy	Test loss	Test accuracy
Mean	Dell (a)	0.356426	0.892669	0.097229	0.971655	0.088083	0.973322
	Mac (b)	0.356423	0.892678	0.097264	0.971662	0.088128	0.973309
	$\lvert a-b \rvert$	3.08e-06	9.17e-06	3.46e-05	6.67e-06	4.52e-05	1.30e-05
Std	Dell (a)	0.009887	0.003258	0.005040	0.001831	0.004966	0.001796
	Mac (b)	0.009879	0.003256	0.005076	0.001846	0.004987	0.001804
	$\lvert a-b \rvert$	8.07e-06	1.92e-06	3.57e-05	1.48e-05	2.07e-05	8.27e-06
Range	Dell (a)	0.044789	0.016646	0.029591	0.010167	0.029687	0.010200
	Mac (b)	0.044769	0.016583	0.029515	0.010167	0.029711	0.010400
	$\lvert a-b \rvert$	1.95e-05	6.25e-05	7.60e-05	0.00e+00	2.31e-05	2.00e-04

Table 12: Results from the comparison of the metrics (train, validation, inference) with $seed \in \{0, \ldots, 99\}$ across two computers and their statistics. The computation can be viewed at (*Metric statistics* 2020).

these values it is possible to conclude that the difference between the statistics of two machines does not exceed 10^{-3}, thus we say that reproduction was successful.

We decide to go one step further and investigate, whether it is possible to *exactly* reproduce the experiments i.e. at the end of training the values of the weights and metrics (train loss, train accuracy, validation loss, validation accuracy, test loss and test accuracy) should be identical to the previous run. We assume that it would be possible to reproduce metrics using a virtual machine or a containerised system, such as Docker, given the wide-spread use and importance of Docker for reproducibility highlighted in (Boettiger, 2015).

One finding is, that it is possible to reproduce the results exactly, given that the set of above mentioned environment and seed parameters are fixed and that we have created a global Tensorflow session. Using the instructions provided (*Docker Instructions* 2020), it is possible to load the Docker image and reproduce exactly the same results as in the notebook in the same repository. Even though the exact reproduction is not a practically applicable feature, as it requires us to not use the GPU during training, thus drastically decreasing the speed of calculation, it is a step towards understanding that the results can be exactly reproduced across different systems.

4 Conclusion

In this report we discussed our comprehensive analysis of the reproducibility-aiding frameworks W&B, Guild and MLflow. We investigated the frameworks in the context of runtime, information saved during auto-logging, number of lines, memory, parameters supporting reproducibility and log size using the sklearn library. Moreover, we compared aspects of memory consumption depending on the data size and parameters supporting reproducibility within the Keras library. We also investigated collaboration support of the three frameworks.

We have shown that it is possible to reproduce the exact values of the metrics of a convolutional neural network using Docker, under a certain set of conditions. We expect that this can be applied to every model which uses randomness. Moreover, we quantified reproducibility for random models by measuring statistics over the metrics obtained from runs, depending on different seeds of random number generators across two machines. We conclude, that even the difference of a strict statistic such as the range is bounded by 10^{-3} across two machines.

References

Barba, Lorena A (2018). "Terminologies for reproducible research". In: *arXiv:1802.03311*.

Biewald, Lukas (2020). *Experiment Tracking with Weights and Biases*. Software available from wandb.com. URL: https://www.wandb.com/.

Boettiger, Carl (Jan. 2015). "An Introduction to Docker for Reproducible Research". In: *SIGOPS Oper. Syst. Rev.* 49.1, pp. 71–79. ISSN: 0163-5980. DOI: 10.1145/2723872.2723882. URL: https://doi.org/10.1145/2723872.2723882.

Docker Instructions (2020). URL: https://github.com/pandorica-opens/Reproducibility-Tools-in-Machine-Learning/blob/master/docker%20replication/Docker.ipynb (visited on 02/17/2020).

Full decision tree (2020). URL: https://github.com/pandorica-opens/Reproducibility-Tools-in-Machine-Learning/blob/master/Decision%20tree/frameworks-all-leaves-tree.svg (visited on 02/17/2020).

Guild AI (2020). URL: https://guild.ai/ (visited on 02/17/2020).

Hutson, Matthew (2018). "Artificial intelligence faces reproducibility crisis". In: *Science* 359.6377, pp. 725–726. ISSN: 0036-8075. DOI: 10.1126/science.359.6377.725. eprint: https://science.sciencemag.org/content/359/6377/725.full.pdf. URL: https://science.sciencemag.org/content/359/6377/725.

Li, Liam and Ameet Talwalkar (2020). "Random search and reproducibility for neural architecture search". In: *Uncertainty in Artificial Intelligence*. PMLR, pp. 367–377.

Liu, Chao et al. (2020). "On the Replicability and Reproducibility of Deep Learning in Software Engineering". In: *CoRR* abs/2006.14244. arXiv: 2006.14244. URL: https://arxiv.org/abs/2006.14244.

Memory Profiler (2020). URL: https://github.com/pythonprofilers/memory_profiler (visited on 03/07/2020).

Metric statistics (2020). URL: https://gitlab.com/anrasg/lab/-/blob/master/Measuring%20accuracy%20deviation%20for%20CNN/Conv_MNIST-accuracy-calculation.ipynb (visited on 02/17/2020).

National Academies of Sciences, Engineering and Medicine (2019). *Reproducibility and replicability in science*. National Academies Press.

Papers with Code (n.d.). https://paperswithcode.com/. Accessed: 2021-03-13.

Replication with fixed seed (2020). URL: https://gitlab.com/anrasg/lab/-/blob/master/Memory%20consumption%20from%20dataset%20size:%20MNIST%20/CNN-Memory-from-Dataset-guildai.ipynb (visited on 02/17/2020).

StatLib archive (2020). URL: http://lib.stat.cmu.edu/datasets/boston (visited on 02/24/2020).

UCI repository of machine learning databases (2020). Department of Information and Computer Science, University of California, Irvine, CA, 1998. URL: http://www.ics.uci.edu/mlearn/MLRepository.htm (visited on 02/24/2020).

Zaharia, Matei et al. (2018). "Accelerating the Machine Learning Lifecycle with MLflow." In: *IEEE Data Eng. Bull.* 41.4, pp. 39–45.

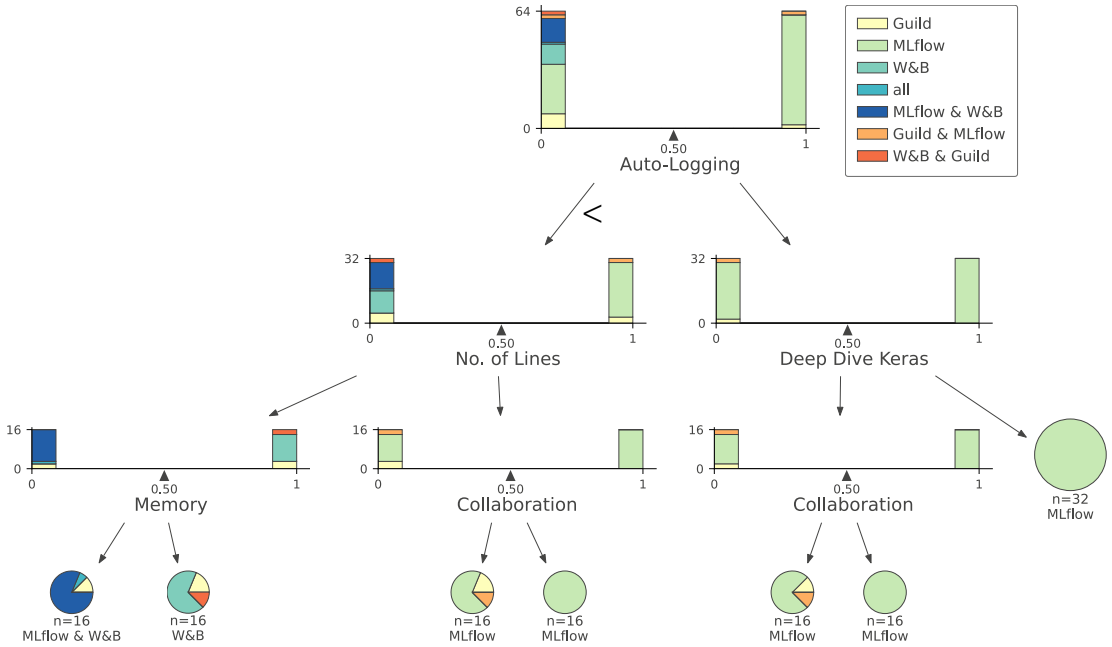

Figure 1: Based on our findings, we construct a decision tree to assist the user according to their preferences of certain criteria using our given scores. It enables a better visualisation of the probabilities as to which framework would be most suitable according to the required criteria. Shown is the decision tree built on each possibility of the criteria choice shown in Table 11, namely: *Auto-Logging*, *Runtime*, *No. of Lines*, *Collaboration*, *Memory*, *Deep Dive Keras* and *Deep Dive Sklearn* (scores for the other criteria are identical). The current tree has a restriction of 3 on the tree depth. The decision tree fits Table 11 completely. Each feature has a value of 1 if it is important to the user, and 0 if this is not the case. The decision space ranges from (0,0,0,0,0,0,0) for the case that none of the criteria are important, to (1,1,1,1,1,1,1) where all of the criteria are important. For instance, if the values for *Deep Dive Keras* and *Auto-Logging* are 1 (they are important to the user), then MLflow would be selected. The full tree is available at (*Full decision tree* 2020). We can observe that MLflow has a clear advantage, W&B is chosen in a specific case, when, amongst other chosen parameters, *Memory* is important and *No. of Lines* is not important.

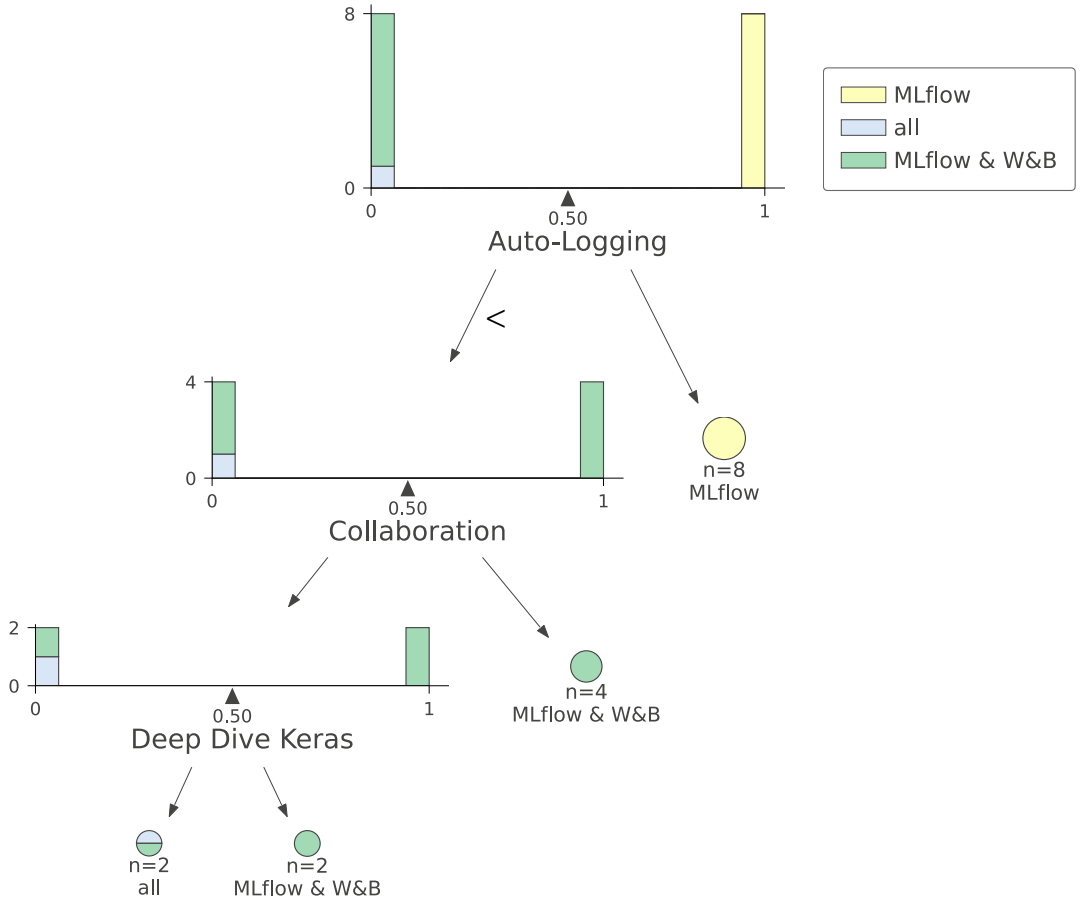

Figure 2: This decision tree concentrates on the criteria that we find particularly useful in terms of reproducibility, namely *Auto-Logging*, *Collaboration*, *Deep Dive Keras* and *Deep Dive Sklearn*. The decision tree is built using the same algorithm as described in the annotation of Fig. 1. The *all* case stands for the edge case of all parameters being not important or $(0,0,0,0)$. We can see that MLflow has the highest chance to be chosen, with wandb being available as an alternative when auto-logging is not important to the user.