Co-ordinated by ECMWF

MAchinE Learning for Scalable meTeoROlogy and climate

MAELSTROM

Report on software performance benchmarking for ML solutions from deliverable D1.3

Nikoli Dryden, Tal Ben-Nun, Saleh Ashkboos, Fabian Emmerich, Jannik Jauch

www.maelstrom-eurohpc.eu

MAELSTROM

D2.3 Report on software performance benchmarking for ML solutions from deliverable D1.3

Author(s):	Nikoli Dryden (ETH), Tal Ben-Nun (ETH), Saleh Ashkboos (ETH), Fabian Emmerich (4cast), Jannik Jauch (4cast)

Dissemination Level:	Public
Date:	31/12/2022
Version:	1.0
Contractual Delivery Date:	31/12/2022
Work Package/ Task:	WP2/ T2.2 T2.3 T2.4 T2.5 T2.6
Document Owner:	ETH
Contributors:	4cast, ECMWF
Status:	Final

MAELSTROM

Machine Learning for Scalable Meteorology and Climate

Research and Innovation Action (RIA)
H2020-JTI-EuroHPC-2019-1: Towards Extreme Scale Technologies and Applications

Project Coordinator:	Dr Peter Dueben (ECMWF)
Project Start Date:	01/04/2021
Project Duration:	36 months

Published by the MAELSTROM Consortium

Contact:
ECMWF, Shinfield Park, Reading, RG2 9AX, United Kingdom
Peter.Dueben@ecmwf.int

The MAELSTROM project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955513. The JU receives support from the European Union's Horizon 2020 research and innovation programme and United Kingdom, Germany, Italy, Luxembourg, Switzerland, Norway

Contents

1 Executive Summary **6**

2 Introduction **7**

2.1 About MAELSTROM 7

2.2 Scope of this deliverable 7

2.2.1 Objectives of this deliverable *7*

2.2.2 Work performed in this deliverable *7*

2.2.3 Deviations and counter measures *7*

3 Benchmarking Infrastructure by Work Package Tasks **8**

3.1 Workflow Platform (Task 2.2) 8

3.2 Benchmarking Tools (Task 2.3) 8

3.3 User Interface (Task 2.4) 8

3.4 Data Input/Output Acceleration (Task 2.5) 8

3.5 Deployment and Infrastructure (Task 2.6) 8

4 Software Benchmarking Results **9**

4.1 Application 1 9

4.2 Application 2 9

4.3 Application 3 9

4.4 Application 4 9

4.5 Application 5 9

4.6 Application 6 9

5 Conclusion **10**

6 References **11**

Figures

Figure 1: Usage of MLflow with the Mantik API 11
Figure 2: Example of an application with the Mantik Compute Backend using the Mantik CLI 11
Figure 3: Secured MLflow GUI 12
Figure 4: Landing page of the Mantik web platform 14
Figure 5: Overview of projects hosted on the platform 15
Figure 6: View of a project 15
Figure 7: Example PyTorch training loop with added timing 18

Tables

Table 1: Benchmarking results for Application 1 21
Table 2: Benchmarking results for Application 2 21
Table 3: Benchmarking results for Application 3 22
Table 4: Benchmarking results for Application 4 using PyTorch 22
Table 5: Benchmarking results for Application 4 using TensorFlow 22
Table 6: Benchmarking results for Application 5 23

1 Executive Summary

Contents of this delivery: report on software performance benchmarking for machine learning solutions.

- Overall status: Software benchmarking tools are on track and initial benchmarks have been completed.
- A comprehensive timing infrastructure for benchmarking deep learning applications has been developed.
- The machine learning solutions from Deliverable 1.3 have been benchmarked and initial results are documented in this report.
- Development of the Mantik workflow platform and user interface is progressing well.

2 Introduction

2.1 About MAELSTROM

To develop Europe's computer architecture of the future, MAELSTROM will co-design bespoke compute system designs for optimal application performance and energy efficiency, a software framework to optimise usability and training efficiency for machine learning at scale, and large-scale machine learning applications for the domain of weather and climate science.

The MAELSTROM compute system designs will benchmark the applications across a range of computing systems regarding energy consumption, time-to-solution, numerical precision and solution accuracy. Customised compute systems will be designed that are optimised for application needs to strengthen Europe's high-performance computing portfolio and to pull recent hardware developments, driven by general machine learning applications, toward needs of weather and climate applications.

The MAELSTROM software framework will enable scientists to apply and compare machine learning tools and libraries efficiently across a wide range of computer systems. A user interface will link application developers with compute system designers, and automated benchmarking and error detection of machine learning solutions will be Wperformed during the development phase. Tools will be published as open source.

The MAELSTROM machine learning applications will cover all important components of the workflow of weather and climate predictions including the processing of observations, the assimilation of observations to generate initial and reference conditions, model simulations, as well as post-processing of model data and the development of forecast products. For each application, benchmark datasets with up to 10 terabytes of data will be published online for training and machine learning tool-developments at the scale of the fastest supercomputers in the world. MAELSTROM machine learning solutions will serve as blueprint for a wide range of machine learning applications on supercomputers in the future.

2.2 Scope of this deliverable

2.2.1 Objectives of this deliverable

Deliverable 2.3 aims to utilise and refine the benchmarking infrastructure developed as part of Work Package 2 to evaluate the software performance of the machine learning applications delivered in Deliverable 1.3. The benchmarking should be low-overhead to integrate and have minimal impact on application performance while providing sufficient detail for users to begin to understand the bottlenecks in their applications or to identify performance regressions as a result of changes. This infrastructure should enable MAELSTROM applications to provide actionable insights from regular (i.e., quarterly) performance benchmarking.

2.2.2 Work performed in this deliverable

This Deliverable included the development of the following components:

- Fine-grained timing infrastructure for benchmarking
- Logging infrastructure for benchmarking
- Further development of the Mantik workflow platform and user interface
- Benchmarking of machine learning solutions

2.2.3 Deviations and counter measures

Following extensive discussion with the application teams, particularly at the MAELSTROM General Assembly and Bootcamp, it was decided to provide a more flexible and less intrusive benchmarking infrastructure than initially envisioned to simplify the initial software benchmarking. While applications may still be implemented as Deep500 recipes and fully utilise the benchmarking framework, we have additionally developed a lightweight timing infrastructure that can be easily added to existing applications without significant modifications. In either case, detailed software benchmarking results will still be produced.

We have further added logging support for services that did not exist or were not widely adopted when the proposal was written, in response to changes in both MAELSTROM and broader community usage. In particular, we integrated support for logging timing results to MLFlow (with integration provided by Mantik) and to Weights & Biases (widely used by the machine learning community and by several MAELSTROM applications).

3 Benchmarking Infrastructure by Work Package Tasks

We detail recent progress in the MAELSTROM workflow and software benchmarking packages.

3.1 Workflow Platform and User Interface (Tasks 2.2 and 2.4)

3.1.1 Workflow Tools

In MAELSTROM, we aim to develop a web platform that simplifies the workflow of W&C researchers. The platform has various targets that support the development cycle of machine learning applications:

- It should especially be designed for development of W&C applications on HPC infrastructures by providing a unified access to the hardware systems. Here, unified access means that we want to abstract away the infrastructure and specific conditions of the respective computation sites to avoid the requirement of in-depth knowledge for interaction with the specialised software of each site, which typically results in a large overhead of work for researchers.
- A main target is to achieve maximum reproducibility of machine learning solutions. The platform should hence govern all parts of a machine learning workflow: data, application code, experiments (input parameters and output metrics), and models. This allows researchers to review earlier solutions and reproduce them at any time.
- Hosting a large number of machine learning applications and gathering all kinds of information about them enables a detailed evaluation of the machine learning solutions. This allows the software frameworks to provide suggestions for well-performing machine learning solutions to other platform users when approaching new problems:
    - What data have people used for this problem?
    - Which machine learning algorithms, architectures, and input parameters have they applied?
    - How well have these approaches performed, and which have performed best?

  This gives newcomers the opportunity to directly jump into the state-of-the-art solutions of a problem they are interested in and try improving it.
- We want to encourage collaboration within the community to improve the quality of research and enable scientists to benefit from each other’s domain knowledge on specific applications. Platform users should be able to share their machine learning applications with researchers of their choice or the public to encourage knowledge exchange and improve the machine learning solutions.

To address these requirements, the following was done:

- We chose to utilise the Open Source machine learning framework MLflow[1] for experiment tracking and model versioning. The software provides a large range of functionality to support the workflow of machine learning developers.
- For the usage of MLflow, we have developed a cloud architecture on the Amazon Web Services platform that encapsulates a MLflow Tracking Server and the MLflow GUI[2]. Natively, MLflow does not provide any security measures to host these software components publicly with restricted access. Thus, on top of MLflow, we implemented services that allow for user management and restrict the access to only allow authorised users to access the cloud instance.
- To provide users with a unified access to HPC resources, we have developed the Mantik Compute Backend. It exposes a REST API to submit machine learning applications directly to a computation site. To allow this, the applications have to conform with the structure of so-called MLprojects. This format is a feature of MLflow and makes machine learning applications independent from any hardware or software environments, and thus, makes it possible to execute them on any system.
- We have developed the Mantik Python package[3] – that also comes with a command line interface (CLI) – to provide users with an API that exhibits different functionalities.
    - To give MLflow users access to the cloud instance, the package is able to authenticate users at the platform and give them access to the secured MLflow services from Python applications.
    - The package provides Pythonic access to the Compute Backend API, and hence, enables users to directly execute and supervise their machine learning applications on the desired HPC system from anywhere.

  The usage of the package is documented on GitHub.[4]

- Currently, the Mantik web platform, which integrates all requirements listed above, is under development.
    - For experiment tracking, the platform will support MLflow. The long-term plan is to completely embed the MLflow GUI in Mantik and encapsulate all MLflow features in Mantik.
    - For projects that conform with the MLproject conventions, users will be able to execute their machine learning applications on the HPC system of their choice directly from the browser. The results of their runs will be logged in real-time and directly displayed on the platform.
- The newly developed Deep500 benchmarking tools additionally support logging measured performance data to Mantik's MLflow instance.

[1] https://mlflow.org
[2] https://cloud.mantik.ai
[3] https://pypi.org/project/mantik
[4] https://github.com/mantik-ai/tutorials

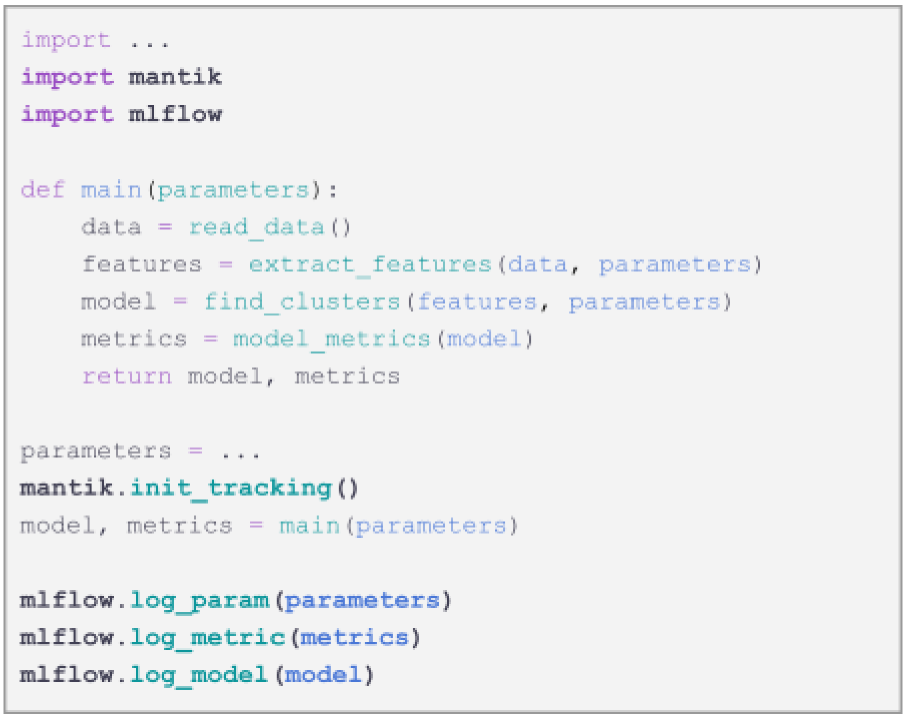
3.1.2 User Interface

3.1.2.1 Python and Command Line Interface

We built user interfaces that allow machine learning developers to:

- Develop all steps of their machine learning applications on a single platform.
- Benchmark the usage of different state-of-the-art machine learning libraries for their specific use case.
- Get unified access to any HPC hardware and infrastructure.

Currently, the Mantik Python API and CLI provide programmatic access to the Mantik API. As described in Section 3.1.1, the API provides features for experiment tracking and model versioning (see Figure 1), and remote execution and supervision of applications on HPC hardware (see Figure 2). In the near future, the GUI, i.e., the Mantik web platform, will provide a more abstract and visual access to the Mantik API.



```python
import ...
import mantik
import mlflow

def main(parameters):
    data = read_data()
    features = extract_features(data, parameters)
    model = find_clusters(features, parameters)
    metrics = model_metrics(model)
    return model, metrics

parameters = ...
mantik.init_tracking()
model, metrics = main(parameters)

mlflow.log_param(parameters)
mlflow.log_metric(metrics)
mlflow.log_model(model)
```

Figure 1: *Usage of MLflow with the Mantik Python API. The Mantik package allows authentication at the secured MLflow cloud instance and logging any desired information.*

```
mantik runs submit /path/to/project \
  --experiment-id 1 \
  --backend-config config.yaml
  -P <parameter 1>=<value>
  -P <parameter 2>=<value>
```

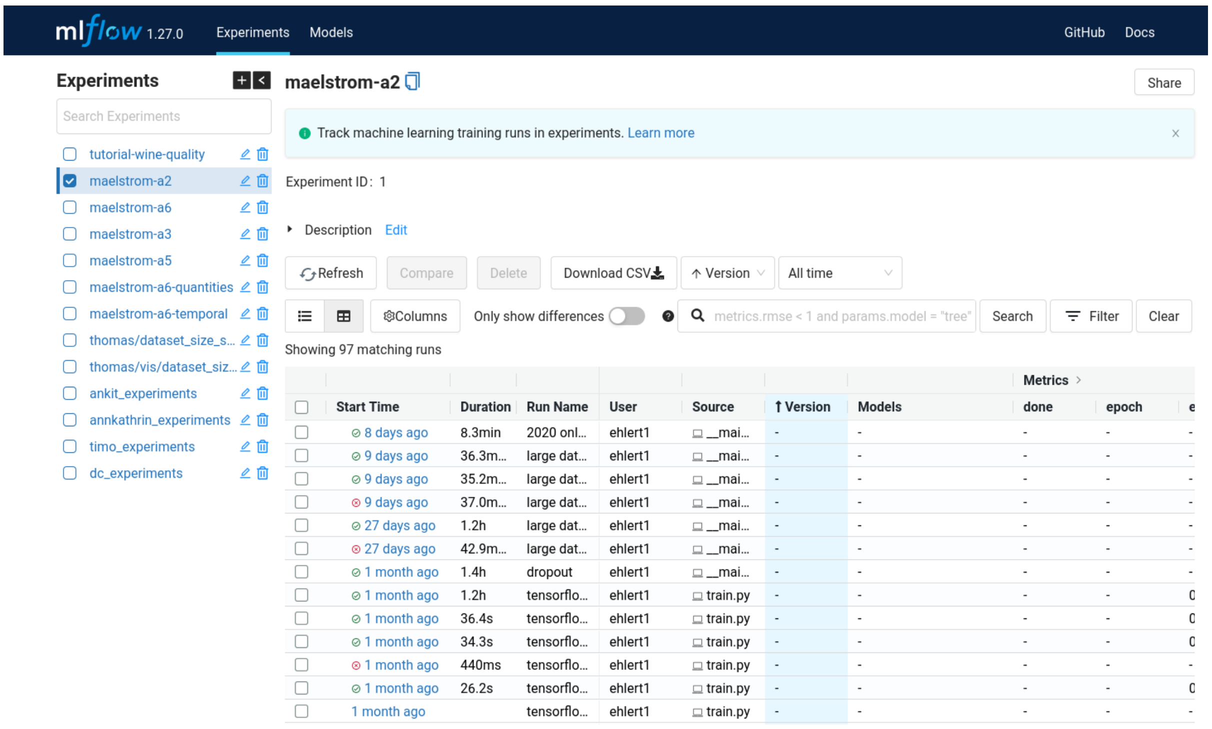
Figure 2: *Execution of an application with the Mantik Compute Backend using the Mantik CLI. The required YAML config allows configuring the resources that will be requested for the run (e.g. type and number of compute nodes, CPUs and RAM per node, etc.), as well as defining any required software modules that should be loaded in the run's environment.*

Figure 2 shows how a user can, e.g., start the training of a machine learning model on an HPC resource from their local machine. Mantik offers additional commands that allows the user to monitor and interact with a submitted job (or *run*):

- `list`: Shows a detailed list of all submitted runs.
- `cancel`: Cancel a submitted run.
- `status`: Shows a run's current status (e.g. if the job is queued, running, failed, or successful).
- `info`: Shows detailed information about an individual run.
- `logs`: Prints the application logs (i.e. stdout/stderr).
- `download`: Allows a single file or an entire folder from the run's working directory to be downloaded.

3.1.2.2 Graphical User Interface

A preliminary GUI is the MLflow cloud instance (see Section 3.1.1) that gives access to stored machine learning experiments and models (see Figure 3). This GUI also offers more advantageous features that allow for a visualisation and comparison of a set of experiments such that researchers can quickly gain insight into the results of their recently executed HPC applications.



mlflow 1.27.0 Experiments Models GitHub Docs

Experiments

Search Experiments

- tutorial-wine-quality
- [✓ maelstrom-a2]
- maelstrom-a6
- maelstrom-a3
- maelstrom-a5
- maelstrom-a6-quantities
- maelstrom-a6-temporal
- thomas/dataset_size_s...
- thomas/vis/dataset_siz...
- ankit_experiments
- annkathrin_experiments
- timo_experiments
- dc_experiments

maelstrom-a2

Share

Track machine learning training runs in experiments. Learn more

Experiment ID: 1

Description Edit

Refresh | Compare | Delete | Download CSV | Version | All time

Columns | Only show differences | metrics.rmse < 1 and params.model = "tree" | Search | Filter | Clear

Showing 97 matching runs

							Metrics		
Start Time	Duration	Run Name	User	Source	Version	Models	done	epoch	e
8 days ago	8.3min	2020 onl...	ehlert1	__mai...	-	-	-	-	-
9 days ago	36.3m...	large dat...	ehlert1	__mai...	-	-	-	-	-
9 days ago	35.2m...	large dat...	ehlert1	__mai...	-	-	-	-	-
9 days ago	37.0m...	large dat...	ehlert1	__mai...	-	-	-	-	-
27 days ago	1.2h	large dat...	ehlert1	__mai...	-	-	-	-	-
27 days ago	42.9m...	large dat...	ehlert1	__mai...	-	-	-	-	-
1 month ago	1.4h	dropout	ehlert1	__mai...	-	-	-	-	-
1 month ago	1.2h	tensorflo...	ehlert1	train.py	-	-	-	-	0
1 month ago	36.4s	tensorflo...	ehlert1	train.py	-	-	-	-	0
1 month ago	34.3s	tensorflo...	ehlert1	train.py	-	-	-	-	0
1 month ago	440ms	tensorflo...	ehlert1	train.py	-	-	-	-	-
1 month ago	26.2s	tensorflo...	ehlert1	train.py	-	-	-	-	0
1 month ago		tensorflo...	ehlert1	train.py	-	-	-	-	-

Figure 3: *Secured MLflow GUI. It shows all experiments and experiment runs of the users. A run represents e.g. a model training with specific input parameters and output metrics and a trained model.*

The GUI of the Mantik web platform is currently under development.[5] A first version of the platform will enable users to create machine learning projects that manage their data, code, experiments, stored (versioned) models, and allow model training and inference (*prediction*) on HPC systems. These projects can be shared with other users of the platform (including specific groups of users) to promote knowledge exchange that improves the machine learning solutions.

Figure 4 shows the current layout of the landing page of the project[6]. Here, users can log in or register (central right button or top right corner buttons), but also just try out the basic features without requiring an account (central left button).

The user's projects as well as public and shared projects hosted on the platform can be viewed and filtered by different criteria (see Figure 5) such as Name, Author, and Problem Type (e.g. Classification, Regression, Recurrent Neural Network, Natural Language Processing).

Once a project has been selected, the project overview and all other details can be viewed (see Figure 6). This view provides all required information about the project:

- Overview (e.g. description, motivation, methodology, latest results, etc.)

[5] https://github.com/mantik-ai/mantik-gui

[6] The layout of the web platform and any pages displayed here (see Figure 3.1.2.4–3.1.2.6) may encounter changes in the future since these were taken from an alpha version of the website that is still under development.

- Code repositories (GitHub, Gitlab, or Atlassian Bitbucket) containing the machine learning application code
- Data (description and access)
- Models – Training (remote training on HPC), Prediction (remote prediction on HPC), Stored (versioned) models.
- Runs (individual runs of training or prediction with models on HPC)
- Experiments (input parameters, metrics, trained models, files, plots, etc.)

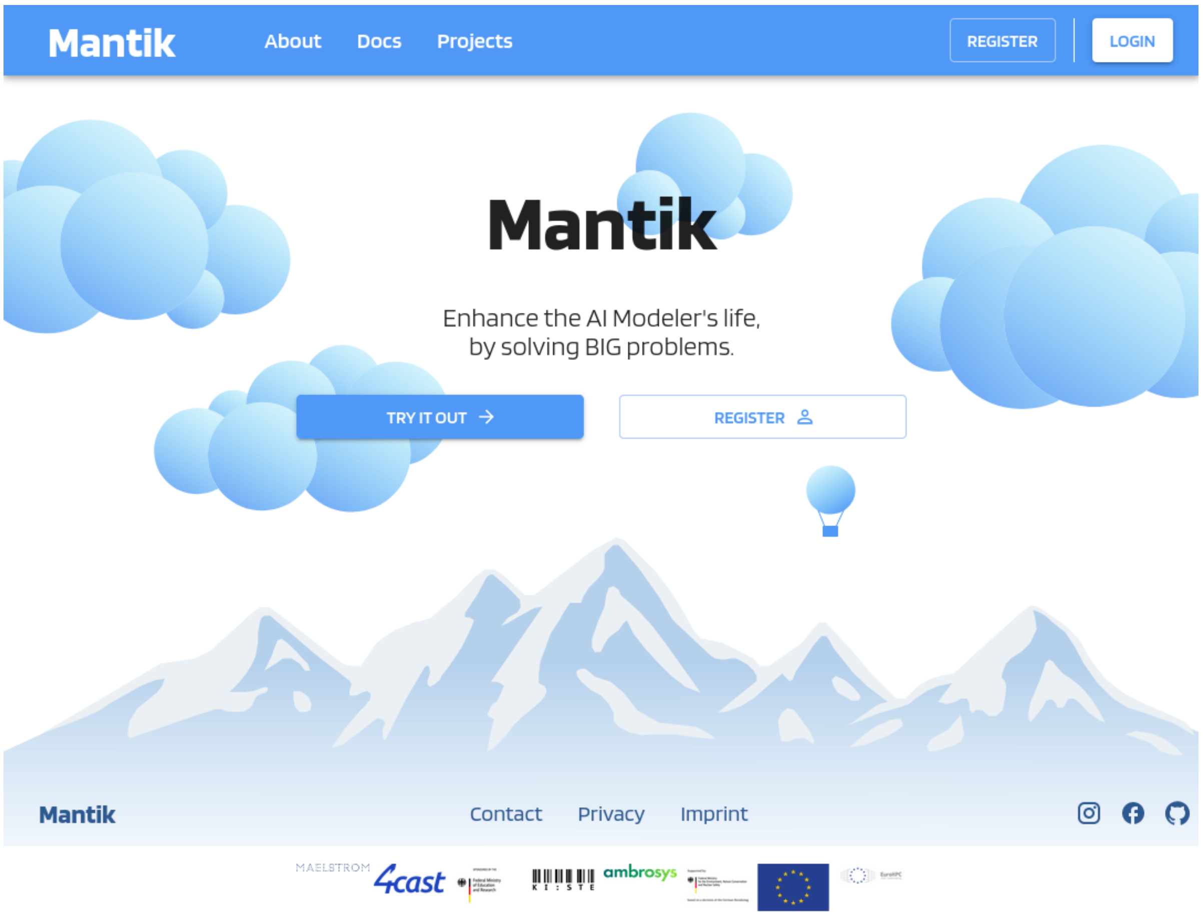
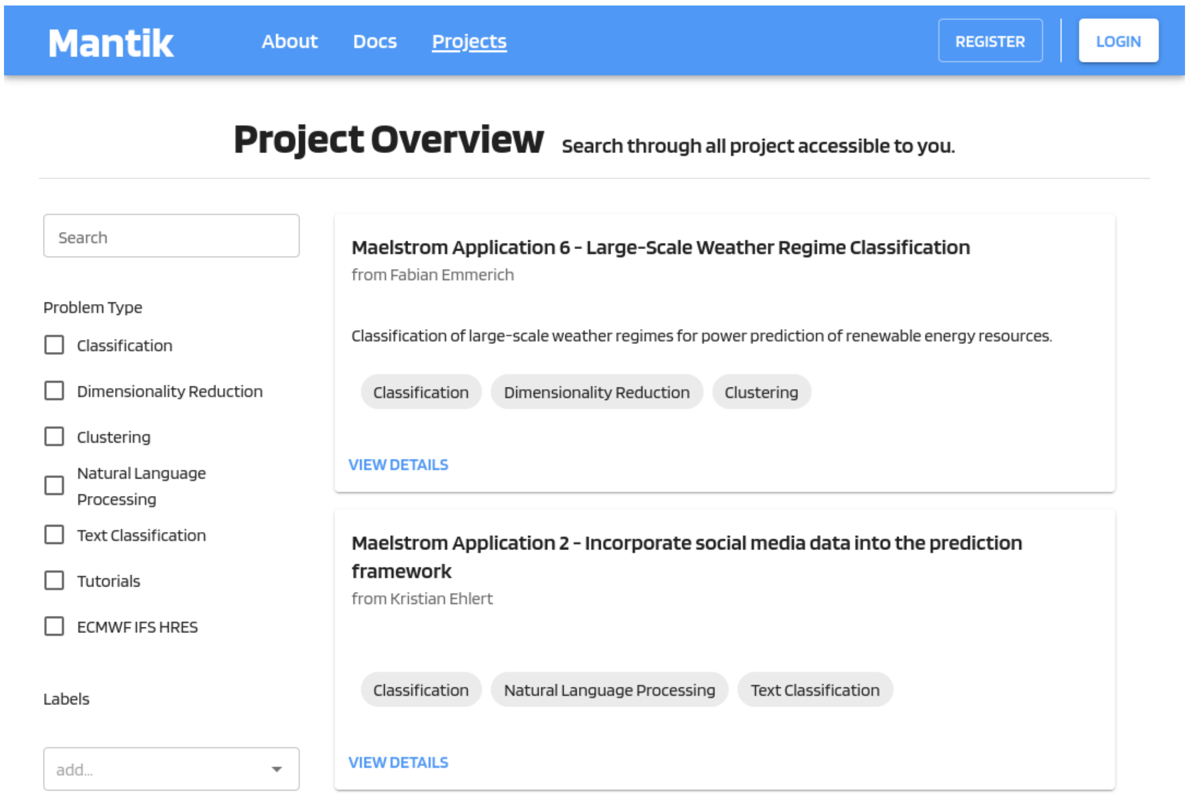


Figure 4: Landing page of the Mantik web platform.



Project Overview Search through all project accessible to you.

Search

Problem Type

- Classification
- Dimensionality Reduction
- Clustering
- Natural Language Processing
- Text Classification
- Tutorials
- ECMWF IFS HRES

Labels

add...

Maelstrom Application 6 - Large-Scale Weather Regime Classification
from Fabian Emmerich

Classification of large-scale weather regimes for power prediction of renewable energy resources.

Classification | Dimensionality Reduction | Clustering

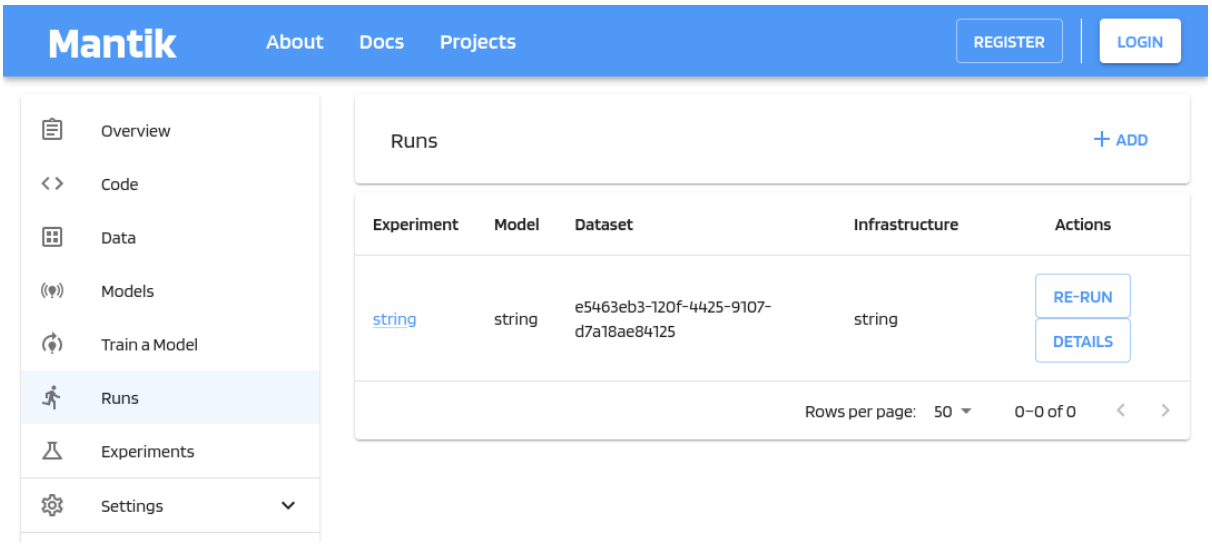
VIEW DETAILS

Maelstrom Application 2 - Incorporate social media data into the prediction framework
from Kristian Ehlert

Classification | Natural Language Processing | Text Classification

VIEW DETAILS

Figure 5: *Overview of projects hosted on the platform. Projects can be filtered by applying different criteria.*



Runs

Experiment	Model	Dataset	Infrastructure	Actions
string	string	e5463eb3-120f-4425-9107-d7a18ae84125	string	RE-RUN DETAILS

Rows per page: 50 0–0 of 0

Figure 6: *View of a project. Here, we see all runs of the application executed on external infrastructure. Other project details that can be viewed are listed in the menu bar on the left hand side.*

3.2 Benchmarking Tools, Deployment, and Infrastructure (Tasks 2.3 and 2.6)

Deep500 [1] is a modular benchmarking infrastructure for high-performance deep learning. As part of Deliverable 2.2, it was extended with better support for deep learning applications in weather and climate. As part of this deliverable, we further extended it specifically to better support benchmarking and analysing the MAELSTROM applications produced in Deliverable 1.3. This effort primarily focused on supporting and annotating relevant regions of an application for timing (e.g., I/O, backpropagation), in addition to overall runtime; supporting applications written in any framework or style; and integrating with external logging applications.

3.2.1 Timing for Machine Learning Applications

We extended Deep500 with a generic `Timer` interface, which supports marking the beginning (`start()`) and end (`end()`) of a region to be timed and measuring the time elapsed between the beginning and end of the region. This allows a developer maximum flexibility to deploy both fine- and coarse-grained timing. Each region is annotated with a *key* that describes the region being timed, to facilitate later analysis, and regions with different keys may be nested or otherwise overlap to break down timing further.

We define the following default keys that an application may use:

- Epoch — One complete pass over a dataset during training
- Batch — One mini-batch during training
- Forward — Forward propagation during one batch
- Backward — Backpropagation during one batch
- Comm — Communication time during one batch (currently intended primarily for measuring the allreduce operation used in distributed data-parallel training)
- IO — Time to load one mini-batch during training

Additionally, a generic “Other” key may be used to represent application-specific regions. Extending this with new keys as needed is also simple.

CPU versus GPU Timing

Many deep learning applications utilise GPUs to accelerate training. However, this poses a challenge for timing, as the GPU component of the application typically executes *asynchronously* with respect to the CPU host; indeed, for best performance, synchronisation points should be minimised. Therefore, timing a region on the CPU (e.g., forward propagation) may give misleading results, as only the time spent on the CPU to launch the asynchronous kernels is measured, rather than the complete time spent performing computations.

To address this, the `Timer` class is generic, and supports pluggable timing implementations. The default implementation utilises standard high-resolution CPU performance timers. We also implemented a region timer that utilises CUDA events, a lightweight timing infrastructure provided as a standard part of the CUDA runtime for measuring elapsed time on GPUs. (A functionally identical API is available for the HIP runtime on AMD GPUs.) These events only synchronise with the CPU when explicitly requested, allowing this overhead to be amortised over many timed regions, minimising any overhead to the application.

In the current implementation, due to technical reasons, GPU-side timing may have additional overhead in frameworks besides PyTorch. This is because other frameworks (e.g., TensorFlow) do not expose the underlying CUDA streams they launch GPU kernels on, making timing difficult. We plan to address this in an updated version of the benchmarking tools.

We also note that the generic timing implementation makes it simple to add support for timing on other accelerators which may be explored by Work Package 3.

Timing Distributed Training

The current timing infrastructure is agnostic as to whether an application is being trained in a distributed manner. At their discretion, the user may time either only a single process among multiple, or time all processes independently and separately log the results (e.g., to detect load imbalance). For typical deep learning applications utilising distributed data-parallel training, the frequent global synchronisation means that timing on a single process is representative. The timing infrastructure can additionally be used to specifically measure the time spent in communication operations, so an application can determine whether it is communication-bound.

In an updated version of the benchmarking tools, we plan to include additional support for timing distributed training, particularly in the context of model-parallel training.

3.2.2 Framework-Independent Timing

The existing Deep500 implementation provided a basic CPU wallclock metric that can be integrated into a Deep500 recipe, and its events interface can allow one to time specific regions. Our above `Timer` interface adds additional support for GPU timing and logging (discussed below). However, many applications in D1.3 are not currently implemented as Deep500 recipes; indeed, one is using scikit-learn, which is not supported by Deep500. To ease the transition to Deep500 and allow applications to get immediate performance feedback, the entire timing infrastructure is entirely framework-independent. A user can directly add timing to their existing implementation without any reimplementation or additional dependencies.

An important limitation we discovered during the benchmarking process is that many applications are using “prebuilt” training loops (e.g., the `model.fit()` paradigm in TensorFlow/Keras). These approaches do *not* provide a way to obtain fine-grained benchmarking results (e.g., of forward, backward, or I/O time). While these libraries typically provide a “callback” mechanism for their training loops, the granularity is limited to the batch level. While this can be circumvented by rewriting the training loop, this is a large burden, and we plan to investigate methods that will enable finer-grained timing despite these limitations.

3.2.3 Logging

The MAELSTROM workflow envisions a unified experience for monitoring and interpreting deep learning applications. To this end, the timing infrastructure supports logging recorded times to the MLFlow metrics logging interface provided by Mantik. Further, for flexibility and to support adoption within the broader machine learning community, it also supports logging to Weights & Biases.

3.2.4 Example Usage

Below we provide an example of using the timing infrastructure with a generic PyTorch deep learning training loop.

```python
import torch
from deep500.utils import timer


def train_epoch(loader, net, criterion, optimizer, tmr):
    tmr.start(timer.TimeType.BATCH)
    for idx, (inputs, targets) in enumerate(loader):
        tmr.start(timer.TimeType.FORWARD)
        output = net(inputs)
        loss = criterion(output, targets)
        tmr.end(timer.TimeType.FORWARD)
        tmr.start(timer.TimeType.BACKWARD)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        tmr.end(timer.TimeType.BACKWARD)

        tmr.end(timer.TimeType.BATCH)
        if idx != len(loader) - 1:
            tmr.start(timer.TimeType.BATCH)

def train_model(args, loader, net, criterion, optimizer):
    net.train()
    tmr = timer.Timer()
    for epoch in range(0, args.num_epochs):
        tmr.start(timer.TimeType.EPOCH)
        train_epoch(loader, net, criterion, optimizer, tmr)
        tmr.end(timer.TimeType.EPOCH)
    tmr.print_all_time_stats()
    tmr.log_mlflow_all('train')
```

***Figure 7**: Example PyTorch training loop with added timing.*

3.3 Data Input/Output Acceleration (Task 2.5)

As I/O can be a major bottleneck for deep learning applications for weather and climate, we ensured that the timing infrastructure developed as part of Task 2.3 included an “I/O” region annotation from the very beginning. This will allow developers to more easily pinpoint whether I/O performance needs to be accelerated for their application.

However, measuring I/O remains a task that we plan to improve upon in future iterations of the software, in collaboration with both MAELSTROM I/O acceleration frameworks (e.g., CliMetLab) and other collaborations (e.g., the IO-SEA project). In particular, when training deep neural networks, I/O is heavily asynchronous and is typically performed by many background threads, which also perform tasks such as data preprocessing and augmentation; in some cases, parts of the process may be offloaded directly onto GPUs. Our initial timing infrastructure primarily measures I/O time as it is observed by the primary training loop (i.e., the time that is not hidden), which is the direct bottleneck. Future improvements will incorporate a more detailed timing breakdown of the entire I/O pipeline.

4 Software Benchmarking Results

In collaboration with the Work Package 1 application teams, the Deep500 timing infrastructure was utilised to conduct initial software benchmarking of the machine learning solutions from Deliverable 1.3. With assistance from Work Package 2, each application integrated the timers and ran benchmarks to produce these results, with the exception of Application 6. As Application 6 is an experimental application not using deep learning, we did not benchmark it, as we expect it to continue to evolve significantly over the remaining course of the MAELSTROM project.

Moreover, Work Package 2 has organised a workshop aimed to introduce Mantik to each of the research applications at the end of November. Each researcher learned how they can utilise Mantik to log their machine learning experiments, including input parameters and output metrics. In conjunction with Deep500, which can be used with Mantik, the frameworks allow detailed benchmarking of the machine learning solutions. Application 2 and 3 have already adopted both of these frameworks for the results shown, with others following in upcoming deliverables.

Because of the different frameworks, structure, and needs of the applications, each application conducted benchmarks at a different granularity, although always at a minimum reporting the time per epoch (i.e., one complete pass over the training dataset). Below we report the results for each application, along with brief details of their benchmarking setup.

4.1 Application 1: Blend Citizen Observations and Numerical Weather Forecasts

Application 1 aims to produce high-resolution (1x1 km) hourly temperature forecasts for the Nordic countries, 58 hours into the future.

The benchmark consists of training a U-Net on a subset of the A1 dataset. We chose the U-Net configuration that scored the best in the tests in Deliverable 1.3. We also chose a small subset (9 out of 362 available training files) of the dataset and ran it for 10 epochs. The 122GB training data was not cached in memory (even though it fits) in order to simulate the expected performance when running on the full dataset.

We ran the benchmark on a single 40GB A100 GPU on the Juwels Booster system using TensorFlow 2.6 and CUDA 11.5. The data processing was done on the CPU and the training on the GPU.

The variability of the processing times are relatively low, as the minimum times are close to the median times. The exception is the first batch, which is significantly slower than the others. This is due to the fact that it includes the data loading time for the first file and framework initialization overhead.

Metric	Min [s]	Mean [s]	Median [s]	Max [s]	Stdev [s]
Epoch	226.96571	228.82088	227.65208	234.10900	2.97469
Batch	0.09784	0.40341	0.11701	41.52411	2.31171

***Table 1**: Benchmarking results for Application 1.*

4.2 Application 2: Incorporate Social Media Data into the Prediction Framework

Application 2 aims to use data provided by social media and weather sensors. As a first step, we use the text of Tweets to predict the occurrence of rain at the Tweet's location and time of creation.

We initialise the *small* variant of the DeBERTa model [3] with pre-trained weights from the *Hugging Face* repository[7]. We finetune the model during one epoch on our large dataset (ca. 1 Mio. Tweets). We used the transformer package (Version 4.25.1) with the PyTorch backend (Version 1.13.0+cu117) to train our model on a NVIDIA Tesla V100 SXM2 16 GB from the Jülich Supercomputing Centre (JSC) using CUDA 11.7. Batch, forward, and backward time was measured using GPU-side timing.

Metric	Min [s]	Mean [s]	Median [s]	Max [s]	Stdev [s]
Epoch	1559.11014	1559.11014	1559.11014	1559.11014	0.00000
Batch	0.13183	0.18065	0.15963	0.54078	0.04867
Forward	0.07812	0.09223	0.08042	0.40440	0.02321
Backward	0.03128	0.06710	0.05871	0.12593	0.02822

***Table 2**: Benchmarking results for Application 2.*

4.3 Application 3: Neural Network Emulators to Speed Up Weather Forecast Models and Data Assimilation

Application 3 seeks to emulate the radiative transfer process for both short and long wavelengths, a columnar problem found within all weather and climate models.

Here we use the TensorFlow framework, version 2.9.1. Training is carried out on one 40GB A100GPU from ECMWF computing resources using CUDA 11.6. Batch time was measured using GPU-side timing.

We evaluate the timing of the final model benchmark from the WP1 reporting, which features convolutional and self-attention layers. Training was carried out as in that reporting, but for only 10 epochs, with an epoch containing 42,268 batches.

We observe very little variability in batch and epoch time. The maximum batch time is considerably larger than the mean and median, but the standard deviation indicates that this maximum is a large outlier, likely due to initialization overheads in the first batch.

[7] https://huggingface.co/microsoft/deberta-v3-small/tree/main

Metric	Min [s]	Mean [s]	Median [s]	Max [s]	Stdev [s]
Epoch	1801.48753	1806.99452	1807.31890	1812.72032	3.91004
Batch	0.02241	0.04203	0.04195	5.69129	0.01004

***Table 3**: Benchmarking results for Application 3.*

4.4 Application 4: Improve Ensemble Predictions in Forecast Post-Processing

Application 4 aims to utilise deep neural networks to post-process the ensemble outputs of ensemble numerical weather prediction systems to improve the quality and skill of forecasts.

The U-Net model from the ENS-10 [2] baselines was selected for benchmarking, and trained for 5 epochs with batch size 1 and the Adam optimizer, following prior methodology. To accelerate the training process, we saved the dataset in NumPy format and used it during the training. Training was performed using PyTorch 1.12.1 on one 40 GB A100 GPU from local ETH computing resources using CUDA 11.6.0. Batch, forward, and backward time was measured using GPU-side timing.

Benchmarking results are shown in Table 4, including a breakdown of time spent on forward and backpropagation and unoverlapped I/O. These results indicate training this model is heavily I/O-bound, and that future performance improvements should focus first on optimising this stage of the application.

We additionally benchmarked the same configuration using TensorFlow 2.11.0. In this case, we save the dataset in TFRecord format. Results are shown in Table 5. By using TFRecords, I/O ceases to be a bottleneck. However, overall epoch time is worse than with PyTorch, due to extremely large initialization overhead for epochs. If this is neglected, then TensorFlow delivers improved performance, with the median batch time being half that of PyTorch. Were the I/O pipeline in PyTorch better optimised, we would expect PyTorch to deliver superior performance (up to 3x faster batch times), based on the timing breakdown for forward and backpropagation.

Metric	Min [s]	Mean [s]	Median [s]	Max [s]	Stdev [s]
Epoch	178.54078	189.92810	184.69046	214.2706	14.18365
Batch	0.03891	0.12581	0.12044	6.96415	0.27348
Forward	0.00472	0.00714	0.00525	4.71542	0.06220

Backward	0.01346	0.01616	0.01587	0.58133	0.00931
IO	0.02158	0.10881	0.10577	6.02123	0.22815

***Table 4**: Benchmarking results for Application 4 PyTorch.*

Metric	Min [s]	Mean [s]	Median [s]	Max [s]	Stdev [s]
Epoch	196.47212	255.80167	263.16660	282.21101	34.15999
Batch	0.05325	0.16876	0.06064	46.32998	1.02192
IO	0.00043	0.03479	0.00089	28.64989	0.69822

***Table 5**: Benchmarking results for Application 4 TensorFlow.*

4.5 Application 5: Improve Local Weather Predictions in Forecast Post-Processing

Application 5 explores the application of deep neural networks to statistical downscaling of meteorological fields. The Tier-2 dataset, which is tested here, is designed for downscaling the 2m temperature from ERA5 reanalysis data with $\Delta x_{ERA5} \simeq 30$ km to the spatial resolution of the COSMO REA6 dataset ($\Delta x_{CREA6} \simeq 6$ km). For training, 11 years of data (from 2006 to 2016) are used which amounts to a number of 94052 training samples. The data from 2017 (2018) are reserved for validation (testing).

As in Deliverable 1.3, a WGAN model is trained in this report. The WGAN model constitutes a composite model with a convolutional U-Net similar to the study of Sha et al. [4], as generator, and a conventional convolutional network as critic model. The model is implemented with Keras from TensorFlow 2.6.0 and trained on a single 40GB A100 GPU node of the JUWELS Booster cluster using CUDA 11.5.

The two sub-networks of the WGAN, the generator and the critic, are alternately optimised during training. Here, the critic gets optimised five times with 32 samples each, before the generator is trained on another set of 32 samples. Thus, the effective mini-batch size where both sub-networks have been at least updated once, consists of 192 training samples. In total, the WGAN model is trained for 60 epochs, meaning that the generator (critic) has been updated on the full training dataset 60 (300) times.

In total, training time shows little variation. Even though the maximum training time for one mini-batch is nearly 200 times larger than its average, the standard deviation is small (less than 50% of the mean training time per mini-batch). Thus, this maximum is a strong outlier with only minor effects on the overall training time. With a standard deviation of about 18.3 s, fluctuations in the training time per epoch are also fairly small. The maximum time per epoch thereby corresponds to the first epoch where the computation graph of Keras is built up and the training data is cached.

Metric	Min [s]	Mean [s]	Median [s]	Max [s]	Stdev [s]
Epoch	553.93	586.63	595.58	661.61	18.27
Batch	0.17573	0.19720	0.20007	39.25604	0.09329

***Table 6**: Benchmarking results for Application 5.*

4.6 Application 6: Provide Bespoke Weather Forecasts to Support Energy Production in Europe

Application 6 investigates whether weather models informed by large-scale weather regimes over Europe have the ability to reduce the uncertainty of power forecasting for wind and solar resources. This application is currently at a highly experimental stage and implemented using the Scikit-learn package to apply dimensionality reduction and clustering, rather than deep learning solutions. As we expect it to continue to evolve heavily over the remaining course of the MAELSTROM project, we did not conduct software benchmarking at this time.

5 Conclusion

This Deliverable describes the current status of the MAELSTROM software benchmarking infrastructure and the results of the initial benchmarking of the machine learning solutions from Deliverable 1.3. The initial timing and benchmarking infrastructure is sufficient, and we see a clear path forward for refining it toward providing further, more detailed information as well as standardising the infrastructure used by all the applications for benchmarking.

6 References

[1] Tal Ben-Nun, Maciej Besta, Simon Huber, Alexandros Nikolaos Ziogas, Daniel Peter, and Torsten Hoefler. “A modular benchmarking infrastructure for high-performance and reproducible deep learning.” In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2019.

[2] Saleh Ashkboos, Langwen Huang, Nikoli Dryden, Tal Ben-Nun, Peter Dueben, Lukas Gianinazzi, Luca Kummer, and Torsten Hoefler. “ENS-10: A dataset for post-processing ensemble weather forecasts.” In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[3] Pengcheng He, Jianfeng Gao, and Weizhu Chen. “DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing.” *arXiv preprint:2111.09543,* 2021.

[4] Yingkai Sha, David John Gagne II, Gregory West, and Roland Stull. “Deep-Learning-Based Gridded Downscaling of Surface Meteorological Variables in Complex Terrain. Part I: Daily Maximum and Minimum 2-m Temperature.” *Journal of Applied Meteorology and Climatology* 59.12, 2020.

Document History

Version	Author(s)	Date	Changes
0.1	Nikoli Dryden (ETH), Tal Ben-Nun (ETH)	07/12/2022	Initial draft
0.2	Nikoli Dryden (ETH), Fabian Emmerich (4cast)	16/12/2022	Section 3.1 and other edits
0.3	Nikoli Dryden (ETH)	23/12/2022	Minor edits
1.0	Nikoli Dryden (ETH)	23/12/2022	Final version

Internal Review History

Internal Reviewers	Date	Comments
Peter Dueben (ECMWF)	21/12/2022	Minor comments and suggestions provided
Thomas Nipen (MetNor)	23/12/2022	Minor comments and suggestions provided

Estimated Effort Contribution per Partner

Partner	Effort
ETH	1 PM
4cast	1 PM
Total	**2 PM**

This publication reflects the views only of the author, and the European High-Performance Computing Joint Undertaking or Commission cannot be held responsible for any use which may be made of the information contained therein.