MAchinE Learning for Scalable meTeoROlogy and climate

Report on improved data processing tools, and the weather data loading pipeline designed for large-scale deep learning training for the benchmark datasets from Deliverable D1.1

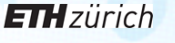
Kristian Ehlert, Fabian Emmerich, Oliver Kindler,

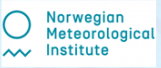
Saleh Ashkboos, Thomas Nipen, Matthew Chantry

www.maelstrom-eurohpc.eu

D2.4 Report on improved data processing tools, and the weather data loading pipeline designed for large-scale deep learning training for the benchmark datasets from Deliverable D1.1

Author(s):	Kristian Ehlert (4cast), Fabian Emmerich (4cast), Oliver Kindler (4cast), Thomas Nipen (MetNorway), Saleh Ashkboos(ETH), Matthew Chantry (ECMWF)
Dissemination Level:	Public
Date:	30/09/2023
Version:	1.0
Contractual Delivery Date:	30/09/2023
Work Package/ Task:	WP2/ T2.2 T2.3 T2.4 T2.5 T2.6
Document Owner:	4cast
Contributors:	4cast, ECMWF
Status:	Final





MAELSTROM

Machine Learning for Scalable Meteorology and Climate

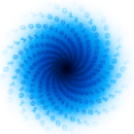
Research and Innovation Action (RIA)
H2020-JTI-EuroHPC-2019-1: Towards Extreme Scale Technologies and Applications

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

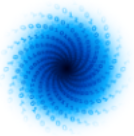
Published by the MAELSTROM Consortium

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

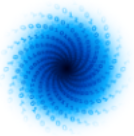


Contents



Figures

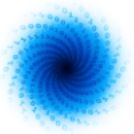
Tables



1 Executive Summary

This document provides an update on developments of the Mantik web-interface and presents work on data loaders.

One of the developments around data loading is focussing on the CliMetLab tool that simplifies the sharing and loading of datasets. Custom solutions can further improve the efficiency of data I/O for specific applications. The Deliverable also presents an update to the Mantik platform Graphical User Interface (GUI). Here, Machine Learning solutions developed in WP1 can be executed, shared, versioned and benchmarked without using a Command Line Interface but through the Mantik GUI. Users can form groups and recommend solutions via a labelling system. Finally, we will also touch on collaborations with our sister projects IO-SEA, RED-SEA and DEEP-SEA from EuroHPC JU in the 'Deployment and Infrastructure' section.



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 (ML) at scale, and large-scale ML 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 ML applications, toward needs of weather and climate applications.

The MAELSTROM software framework will enable scientists to apply and compare ML 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 ML solutions will be performed during the development phase. Tools will be published as open source.

The MAELSTROM ML 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 ML tool-developments at the scale of the fastest supercomputers in the world. MAELSTROM ML solutions will serve as a blueprint for a wide range of ML applications on supercomputers in the future.

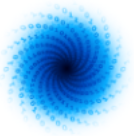
2.2 Scope of this Deliverable

2.2.1 Objectives of this Deliverable

As part of Task 2.5, this Deliverable provides an overview of the improvements on the data processing tools and the entire data loading pipeline. These tools need to be flexible when applied to a variety of large datasets while simultaneously providing adequate performance and user-friendly accessibility.

Large-scale deep learning training for the benchmark datasets from Deliverable 1.1 is possible through the workflow platform Mantik as part of the Co-design cycle. This boosts collaboration and easier reproduction of solutions (Task 2.2). An automated benchmarking routine and a recommendation system, integrated into the workflow platform, is implemented. This represents a major contribution to Task 2.3.

We report on the initial version of the GUI of the workflow platform which provides major improvements for the user experience (Task 2.4). An Update on the progress regarding deployment of models on different infrastructures is given as well (Task 2.6).



2.2.2 Work performed in this Deliverable

The CliMetLab python package provides access to the benchmark datasets from Deliverable 1.1 and enables researchers to migrate the datasets to a location of their choice (Task 2.5). In order to do so, every application implemented a plugin for their datasets. Since datasets can be very large in the domain of weather and climate sciences, application AP1 and AP4 developed customised data loaders to overcome bottlenecks due to memory limitations. This allows for more efficient processing of data by the neural networks.

Through the possibility to label applications and their assets, the workflow platform can recommend solutions to user specified ML problems. A new feature allows to schedule run submissions on HPC clusters. This also enables users to set up automated benchmarking cycles from the Mantik platform (Task 2.3).

The GUI of the workflow platform Mantik is now publicly accessible as part of this Deliverable (Task 2.4). The Mantik GUI allows users to execute training of ML models on the JSC clusters without the need for the user to interact with the cluster directly. Features of MLflow integrated into Mantik allow for monitoring of model training and analysis of results. The workflow suggested by Mantik fosters collaboration, shareability and reproducibility (Task 2.2).

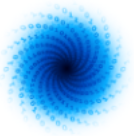
The software architecture of Mantik is designed such that additional clusters can be integrated. Furthermore, a feature that allows the deployment of models from the workflow platform is planned (Task 2.6).

2.2.3 Deviations and countermeasures

The project proposal envisions a workflow tool that enables users to conduct all stages of ML development on a single platform. During the first year of the project, it became apparent that the open-source platform MLflow already provides solutions for a couple of these stages, i.e. tracking and versioning features that are expected from Mantik. Instead of re-implementing these features from scratch, an integration plan of these MLflow functionalities was devised. Mantik now aims to enable the user to store, reproduce, share and benchmark ML solutions by integrating the MLflow Tracking UI while simultaneously providing its core feature, that is a GUI to HPC infrastructure.

Initially, the MAELSTROM project aimed to execute automated benchmarking cycles of the applications quarterly. This proved to be too much overhead for the scientists, as some of the training procedures take special hardware and significant compute time to reach a comparable checkpointing during training. To determine if a benchmark is meaningful is best judged by the scientists themselves. Therefore, a feature was implemented that allows automated run submission on HPC clusters by a user specified schedule. These run schedules can be instrumentalized to submit benchmarking runs but offer additional use cases and therefore provide a less resource intensive alternative to the initial idea.

For more detailed information on the deviations regarding the benchmarking infrastructure, see Deliverable D2.3 on performance benchmarking.



3 Progress by Work Package Tasks

In this section, we provide an update on recent progress in the MAELSTROM workflow and software benchmarking packages to put the Deliverable into context of the wider developments within Work Package 2.

3.1 Workflow Platform (Tasks 2.2) & Benchmarking (Task 2.3)

The MAELSTROM project develops the web-based platform Mantik considering requirements given by W&C researchers. Combining these with a typical ML workflow protocol and updated expectations on the platform during the project, various targets for the software were formulated to support the production cycle:

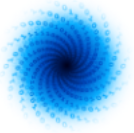
- The usage of HPC clusters usually requires in-depth knowledge of a specialised software landscape provided by the cluster administration, resulting in a large overhead of work for researchers. By abstracting away the specific conditions of the respective clusters, Mantik aims to provide a unified access to the hardware systems. First and foremost this regards the JSC cluster since many large data sets that are used by W&C researchers in the MAELSTROM project are hosted there. In addition, we plan to integrate additional HPC centres relevant to the project, e.g. HPC centre of E4 or further HPC EU project collaborators.

- In order to make our ML solutions reproducible, all steps of the workflow must be accessible through the platform including basic data management, application code and experiments (that hold variations of the same model with varying input parameters and their respective output metrics). Hosting these assets on the platform will enable users to interact with their application on HPC, e.g. training models, evaluating trained models and redo the same task at any later point in time, i.e. reproduce their results.

- We want to encourage collaboration and formation of interest groups within the research community. Platform users should be able to share their ML applications with researchers of their choice or the public to encourage knowledge exchange and improve the ML solutions.

- To assist the further development of solutions, the software framework should foster suggestions of well-performing ML applications between users. A large number of hosted applications, appropriately characterised, lay the foundation for an easier access to state-of-the-art ML solutions for newcomers and experienced researchers alike.

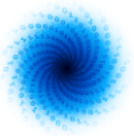
Features of the Mantik User Interface form a central contribution to this Deliverable. More details on the implementation of the Mantik User Interface are given in Section 3.2. To address the requirements listed above, the following features have been implemented:



- We chose to utilise the open-source ML framework MLflow[1] [2] for experiment tracking and model versioning. The software provides a large range of functionality to support the workflow of ML developers.
- For the usage of MLflow, we have developed a cloud architecture on the Amazon Web Services (AWS) 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. Personal credentials to HPC infrastructure can be added in user settings as well, so a run can be scheduled via the GUI.
- To provide users with a unified access to HPC resources, we have developed the Mantik Compute Backend. It exposes a REST API to submit ML 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 ML applications independent from any hardware or software environments, and thus, makes it possible to execute them on any system. The interface UNICORE was developed in 1997 during a research project funded by the German Ministry of Education and Research and has been under constant development ever since. Today it is used as the central gateway to the JSC clusters. The workflow platform is able to send requests and schedule runs via a GUI on JSC clusters through UNICORE.
- 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 ML applications on the desired HPC system from anywhere.

  The usage of the package is documented on GitHub.[4]
- Currently, Mantik is a fully functional web platform, which integrates all requirements listed above and it will be constantly updated with emerging new requirements.
    - For experiment tracking, the platform supports MLflow. It completely embeds the MLflow GUI encapsulating all MLflow features with additional security.
    - For projects that conform with the MLproject conventions, users are able to execute their ML 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



To benchmark the applications from the MAELSTROM project, the ETH Zürich developed the Deep500 python package that can be integrated into the application code and installed via `pip install deep500`. The code is open-source and can be found on GitHub[5]

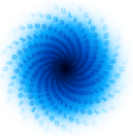
The Deep500 tool [7] is a modular benchmarking framework designed for evaluating the performance of high-performance deep learning systems. It breaks down the training of neural networks into four distinct levels: operators, network processing, training, and distributed training. This modular framework enables users to effortlessly create consistent guidelines for training machine learning models across various datasets and systems. Subsequently, these guidelines produce comprehensive reports, including logs and violin plots, that highlight specific performance metrics selected by the users during the training process.

Within the MAELSTROM project, we are collaborating with the applications in WP1 to expand the Deep500 framework, aiming to develop a straightforward solution that aligns seamlessly with weather and climate workloads. This endeavour involves crafting a unique recipe for each application. We reported our initial efforts in Deliverable 2.2.

As part of Deliverable 1.3, we expanded the framework with a specific emphasis on enhancing its capabilities for benchmarking the applications within the MAELSTROM project. This enhancement primarily concentrated on facilitating and annotating pertinent sections of an application for timing purposes (e.g., I/O, backpropagation), alongside tracking the overall runtime. We developed a simple script to benchmark all applications. All results have been shown in D 1.3.

As of now, Deep500 is a fully functional tool that is instrumentalized by all MAELSTROM applications for benchmarking.

[5] https://github.com/deep500/deep500



3.2. User Interface (Task 2.4)

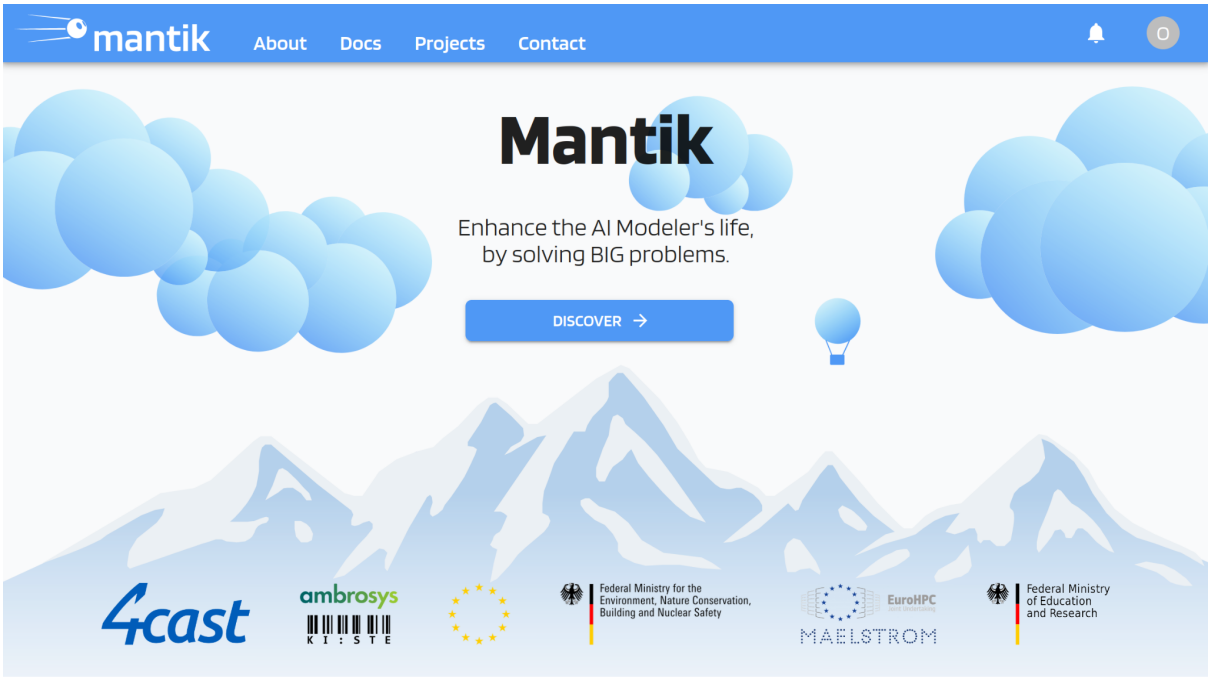
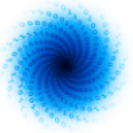


Figure 1: *Landing page of the Mantik web platform.*

3.2.1. Projects on the Web-Platform

Creating a functional ML application involves a variety of complex tasks. After creation of a dataset, data is usually stored on remote locations due to their large sizes in W&C sciences. For training on a remote system, data must then firstly be fetched. Then the dataset is pre-processed. After a promising model architecture is selected, our data is used to train the model. For large models, the application code must be executed on HPC machines with high security restrictions for this task. After evaluation of the results, application code including parameter choices are changed to improve performance. This means repeating the previously described cycle until the desired performance is reached. These steps are usually performed in labour-intensive manual steps. To reduce some of this redundant work overhead, that makes up the reality for W&C scientists, frames the expectations on a web-based workflow platform.

Data storage and management of datasets are left to the user's responsibility. As some datasets of MAELSTROM applications exceed sizes of multiple terabytes, they will always need to be stored at the source of the computing centre. The challenge to include flexible storage solutions and management for these datasets would therefore greatly exceed the scope of the project. However, we provide general access to our datasets to the community via the Climetlab tool (see Section 3.3.1). In addition, quick data transfer of for example smaller subsets of our datasets is provided by the Mantik CLI (see Section 3.2.1.1.). The main focus of Mantik is to provide users with a smooth workflow when working on and across multiple (HPC) machines. For this, Mantik aims to provide easy access to HPC to enable users to train, evaluate, reproduce and share their models. A well structured and user friendly GUI is implemented to serve this purpose.



We introduce different entities to organise every step of the development process. Projects form the core of the Mantik experience. They hold the main entities required for developing ML solutions with Mantik. These entities currently are *Code, Data, Experiments, and Runs* in a holistic approach. They will be explained in detail later in this section. To get started working with the platform, the user needs to register an account. After registration, the user can create a *Project* on the Mantik website[6]. A *Project* page is created and presented as the central hub for all upcoming work, providing access to the functions of Mantik that are accessible via the main menu (see Fig 2). Users can declare their project to be public, contributing to a wide range of accessible projects, or choose to restrict access only to certain users, e.g. their collaborators. An overview of all public projects can be found under the '*Projects*' repository of Mantik. Here the user can reach all public projects or private projects that the user may access.

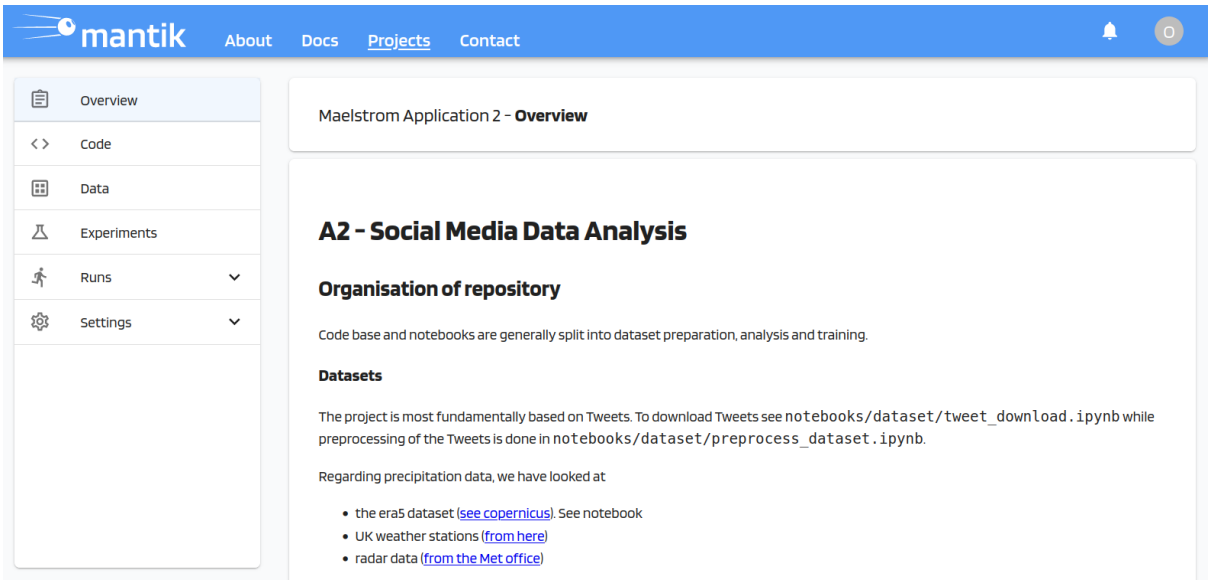
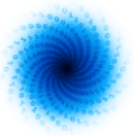


Figure 2: *View of a project. Here, we see the project description and all entities related to the project in the main menu on the left side, which provides access to all Mantik features.*

3.2.2. Model training and Data on HPC clusters

In W&C sciences, the size of the training data sets exceeds the capacity of commercially available personal computers. Due to their size, data sets need to be processed on the HPC cluster itself. In a typical workflow, the user triggers the training of the model manually by operating a Command Line Interface (CLI), either through containerizing their application code (for example as an Apptainer image) or through execution features provided by the cluster administrators (integrated *venv*[7] environments, Jupyter notebooks[8]). The coordination of working steps has to be carried out by the user and a unified access to HPC clusters through a GUI does not exist so far to the knowledge of the authors.

[6] https://cloud.mantik.ai/
[7] https://docs.python.org/3/library/venv.html
[8] https://docs.jupyter.org/en/latest/



The web-based workflow platform developed in the MAELSTROM project aims to offer its users a holistic view of their application development process independent of computing resources. As all related entities are managed on the platform reproducibility becomes an inherent feature of the platform.

To provide an introduction to the functionality of Mantik, this section provides an overview of how Mantik enables users to train their model on Mantik and simplifies aspects of data handling.
In order to train their ML solution on HPC from the Mantik platform, Mantik requires the user to have access to:

- A Git code repository (of the ML solution)
- Access credentials to an HPC cluster (so far JSC, E4/CSCS planned)
- Two configuration files governing the execution of a run (MLproject file, Compute Backend config)

More details and format of the Code repository and the configuration files are explained next.

3.2.1.1. Code repositories, configuration files and data management

In order to make a code repository accessible through Mantik UI, the user can add one or multiple code repositories in the *Code* tab of the *Project* page. This should be the first step of every new project created in Mantik. The application code repository can then be instrumentalized to submit runs to HPC systems.

In addition, we highly encourage users to track their results with Mantik. Our solution is built on MLflow, which allows the user to view their tracked results in a GUI with comprehensive ways to compare and analyse results. Insights can be gained and easily shared via the GUI as no access to user created log files or plots is required. As all results are in one place, the tedious task of keeping track of various figures or log files scattered all over the developer's local machine or HPC node becomes a thing of the past. In accordance with MLflow, we group tracked results and parameters of (training, evaluation,...) runs in *Experiments*. Users are encouraged to create multiple *Experiments* per *Project* where individual *Experiments* are related to a specific model architecture or solution related to the *Project*. The user can create *Experiments* in the respective tab. They are required for creating new *Runs*.

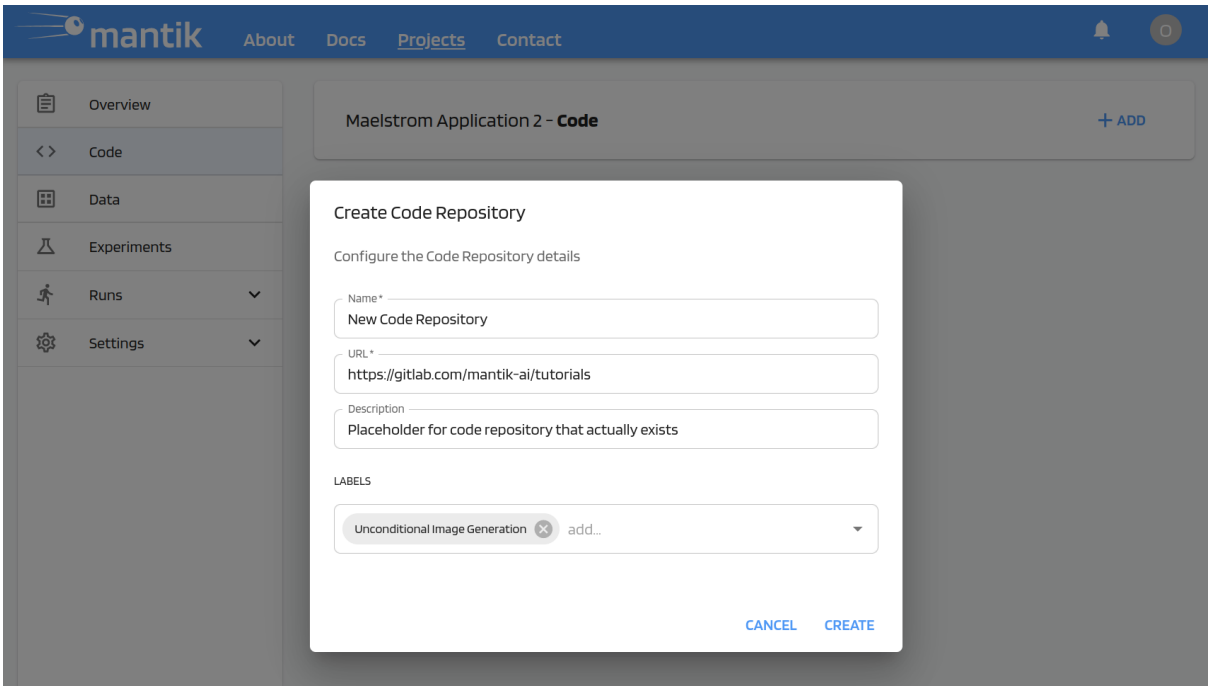
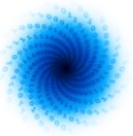


Figure 3: *Creating a Code repository for a Mantik project. Name, URL and short description are required, labels are optional but recommended*

The tracking of metrics and parameters for the purpose of evaluating performance of ML solutions requires usage of the MLflow package in the user's application code. The Mantik python package is hosted on *pypi*[9] and can be installed via the widely used Python package manager `pip`.

When executing the ML solution on HPC Mantik takes care of linking the execution to the user specified Experiment. Input parameters specified in the execution abstraction used by Mantik (MLproject file, see below) are automatically tracked. Additional parameters or metrics only need to be passed to the respective logging functions (see Figure 4 for an example). Details on the relationship between Mantik and MLflow are discussed in detail in the next section.

[9] https://pypi.org/project/mantik/



```python
import ...
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 = ...
model, metrics = main(parameters)

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

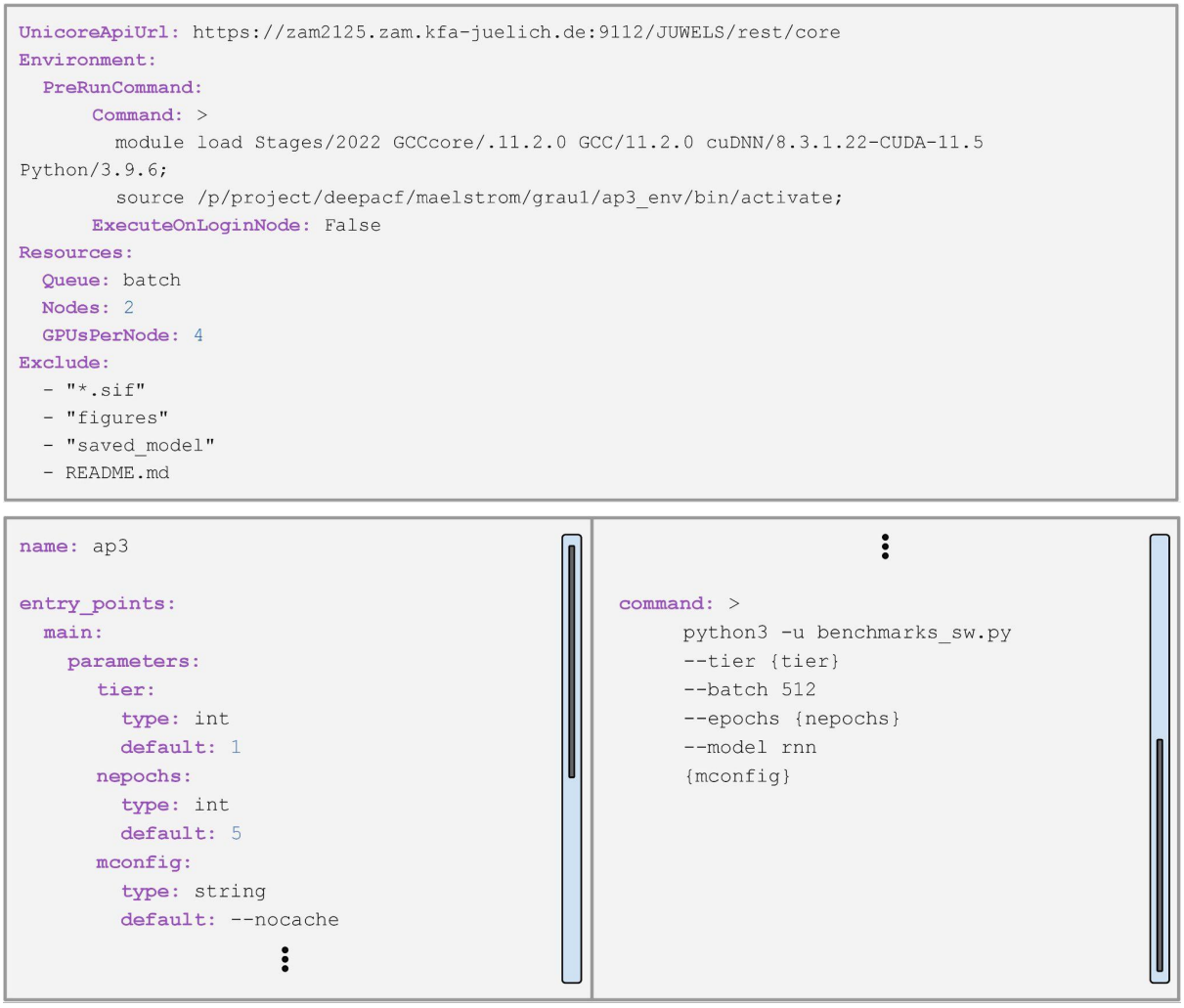
Figure 4: *Usage of MLflow with the Mantik python package. The package allows logging of e.g., parameters, models, metrics to a secure MLflow cloud instance.*

Two configuration files need to be present in the project directory of the repository for executing a run through Mantik:

- The *MLproject* file is required to configure the ML project in the MLflow instance. We use the same format as introduced by MLflow[10], which is already well established in the community. The MLproject file allows the user to list various commands (pre-processing, training, etc.) related to their application and related input parameters. When submitting a new run, Mantik allows the user to then select the desired command and vary the specified input parameters. Having a single file to configure the application requires a certain amount of planning beforehand but ultimately results in a format that can easily be reused for similar tasks like changing input parameters. For better organisation, we encourage users to have one MLprojet file per MLflow *Experiment* such that the user will have multiple MLproject files per Project. An example of this *YAML* file can be found in Figure 5. For further information on the MLproject format see the Mantik documentation[11].

- Mantik requires an additional Backend configuration file to operate properly. Here details for the HPC cluster execution are specified. This includes the name of the addressed cluster queue, the number of nodes to include in the training or the path to the container that contains the execution environment for the application. Additional optional parameters can be used to e.g.specify environment variables or execute code on the HPC cluster before run submission. [12]

[10] see MLflow Documentation: https://www.mlflow.org/docs/latest/projects.html
[11] https://mantik-ai.gitlab.io/mantik/mlflow/mlproject.html
[12] Please note, that credentials to HPC are not specified in the Backend config but in the account settings of Mantik



```
UnicoreApiUrl: https://zam2125.zam.kfa-juelich.de:9112/JUWELS/rest/core
Environment:
  PreRunCommand:
      Command: >
        module load Stages/2022 GCCcore/.11.2.0 GCC/11.2.0 cuDNN/8.3.1.22-CUDA-11.5
Python/3.9.6;
        source /p/project/deepacf/maelstrom/grau1/ap3_env/bin/activate;
      ExecuteOnLoginNode: False
Resources:
  Queue: batch
  Nodes: 2
  GPUsPerNode: 4
Exclude:
  - "*.sif"
  - "figures"
  - "saved_model"
  - README.md
```

```
name: ap3

entry_points:
  main:
    parameters:
      tier:
        type: int
        default: 1
      nepochs:
        type: int
        default: 5
      mconfig:
        type: string
        default: --nocache
                 ⋮
```

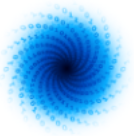
```
                 ⋮

command: >
      python3 -u benchmarks_sw.py
      --tier {tier}
      --batch 512
      --epochs {nepochs}
      --model rnn
      {mconfig}
```

Figure 5: *MLproject configuration file[13] (bottom) and a Mantik Backend config[14] (top) of AP3.*

Naturally, data needs to be accessible to train a model. Since datasets can take up several Gigabytes or even Terabytes of disk space, uploading the data directly to the workflow platform and referencing it from here would not be practical as it would significantly degrade performance or slow down the workflow due to then required downloading or streaming of the data.

Instead, Mantik offers a versatile CLI command to manage data. Additionally, the Data tab on the Mantik platform offers the user an overview of the manipulation tasks performed. An example snippet for a command that copies a directory from the user's local machine to HPC can be found in Figure 6. Mantik's *unicore-file-service* allows the user to connect to a cluster through a Python Client

[13] https://github.com/4castRenewables/maelstrom-radiation/blob/main/mlproject/MLproject
[14] https://github.com/4castRenewables/maelstrom-radiation/blob/main/mlproject/unicore-config-venv.yaml



(called Mantik Remote File Service) via an UFTP[15] connection. This allows the user to move any file or directory from his laptop to the HPC cluster and also perform basic file management operations on the remote file system like deleting or editing a file without the need to establish a SSH connection to the HPC cluster.

```
mantik unicore-file-service copy-directory \
  --connection-id 637e72cf-da59-4a42-beaf-fc08ffad75d9 \
  /local/path/to/my/data/ \
  hpc://p/project/maelstrom/data
```

Figure 6: *Example of a Mantik data CLI command*

3.2.1.2 Executing runs on HPC clusters

Before a user can interact with an HPC cluster (submit runs, upload data, etc.), a new *Connection* with the access credentials to the cluster has to be added in the user settings as seen in Figure 7. Mantik offers a high standard of security through usage of the state of the art encryption management system HashiCorp Vault[16]. This way all sensitive information is double-encrypted while still easily accessible by the user.

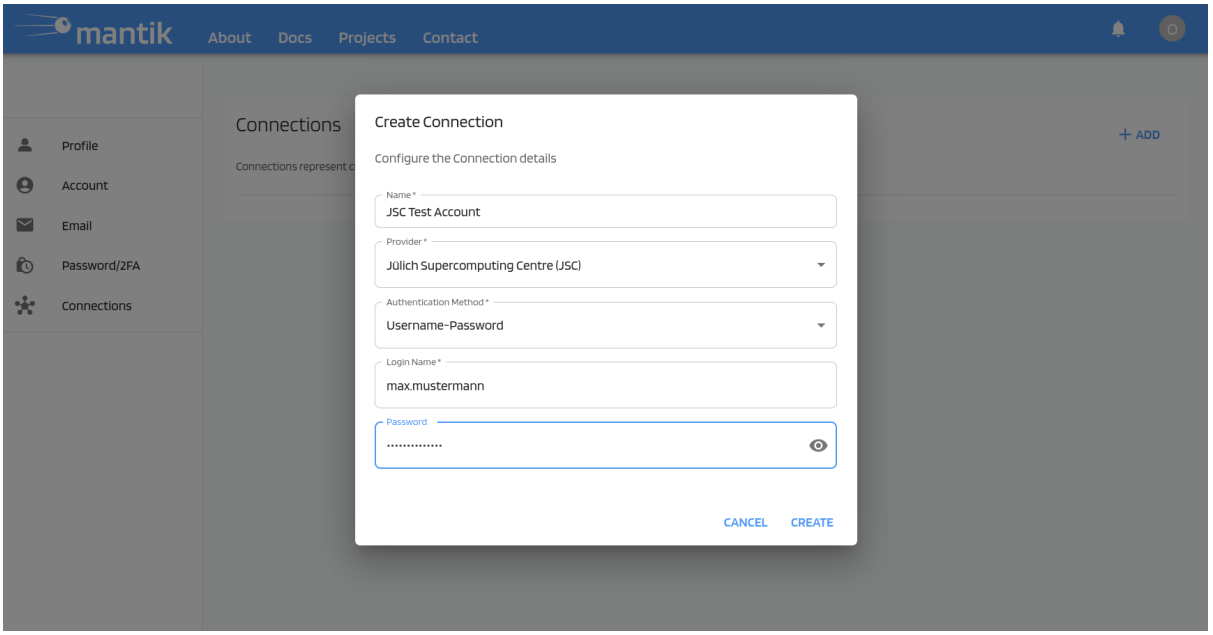
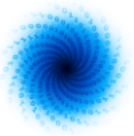


Figure 7: *Adding a new Connection in the user settings, which holds user credentials required for external services like HPC access.*

[15] https://en.wikipedia.org/wiki/UFTP
[16] https://developer.hashicorp.com/vault/docs?product_intent=vault



Since Mantik incorporates the versioning & tracking capabilities of MLflow, every *Run* must be assigned to an MLflow Experiment, which holds tracked information of all related runs. Therefore, an experiment has to be created in the *Experiments* tab before a *Run* can be submitted. For this, only a name for the experiment is needed. During run submission, the user can then simply choose the respective *Experiment* from a drop down menu.

After adding a code repository, which contains configuration files that control the executable and the HPC run environment , an experiment for tracking and making sure pre-processed data is available on the HPC cluster, everything is set to submit a *Run*. To then add a new *Run Submission*, we use the add a new *Run Submission* form in the *Run* tab (see Figure 7).

In the Run form, the user picks a *Code* repository. Mantik allows the user to specify a branch of the git repository. Relative paths to the two previously introduced configuration files (MLproject file, compute backend config) should then be provided. Mantik automatically detects all user-defined entry points and their input parameters given in the MLproject file and allows the user large flexibility when setting parameter values that will be used for running and configuring the application once the Run is submitted to HPC. Since changing of crucial parameters for the model are very common during the development process of a ML solution, not forcing the user to rewrite or introduce new configuration files for this was a core priority for the developers. In addition, Mantik allows the user to modify the backend configuration file on-the-fly. For example, this makes it possible to spontaneously pick less busy queues.

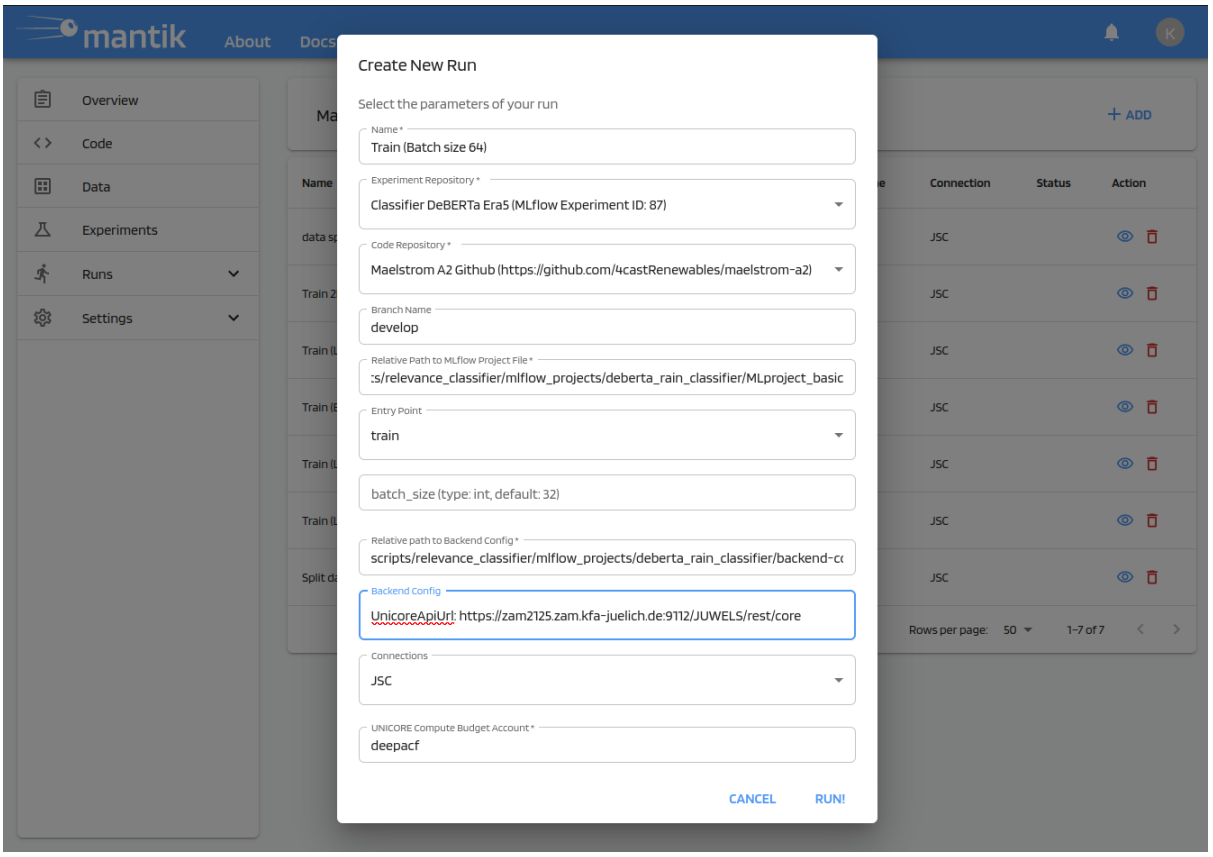


Figure 8: *Executing a run on HPC clusters is handled via this form on the Mantik platform. Configuration files that handle execution are specified and can be customised in this form before submission*

Finally, a *Connection* to a HPC cluster and the respective compute budget account is selected , which handle credentials and bookkeeping on the cluster side. e. Since most data for the MAELSTROM project is located on the HPC clusters of the Jülich Science Center (JSC), Mantik at the moment primarily focuses on accessing these machines, especially the JUWELS cluster[3].

JSC provides a secure interface to its computational resources named UNICORE. It provides a RestAPI that is used by the Mantik Compute Backend to submit applications to HPC from the platform. In the future, more HPC clusters will be integrated, including specialised machines by E4 and the CSCS cluster at Zürich.

As of now, debugging, logs and info on status of a *Run* are only accessible via CLI commands. We are currently in the process of introducing them to the Mantik GUI, such that users get *Run* related information directly under the *Run details* menu and can also download any log files directly to their machine.

3.2.2. Reproduction of solutions, MLflow Integration & Run scheduling

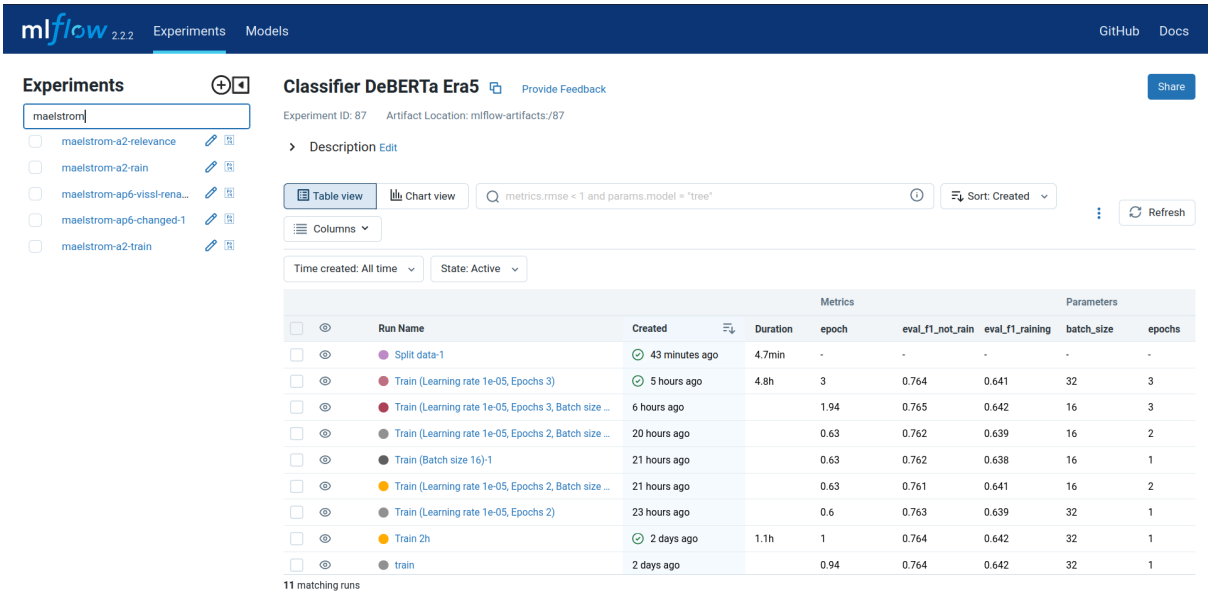


Figure 9: *Secured MLflow instance. It shows all experiments and experiment runs of the users. A run represents e.g. a model training with specific input parameters and output metrics and a trained model.*

After a *Run* was successfully submitted to the HPC cluster, input settings shaping a run like the sizes of batches, number of training epochs or other specifications denoted in the MLproject file are tracked automatically without any necessary intervention by the user. Purely single MLflow function calls are required to track any additional parameter, metric or artefacts (e.g., figures). Then, results will be tracked automatically by Mantik and are accessible in real-time in the provided MLflow UI. Pooled in MLFlow experiments, runs can be compared and their results investigated. All tracked parameters can be plotted or analysed regarding their interdependencies. Additionally, the evolution of metrics is readily available. Typical questions emerging from the performance analysis of a model like the dependency between solution accuracy and size of the training dataset can be answered with minimal effort. Interactive plots can be created directly in MLflow as well. Should researchers decide to track their own plots as crucial evaluation characteristics in the training script itself, they will be displayable without requiring any further adjustments.

As all project assets are saved to Mantik for creating *Runs*, the same *Run* can be executed by any authorised user in the future. A convenient solution for handling these "*Re-Runs*" is currently in the testing phase and will soon be available to all users of Mantik. This represents a user-friendly implementation to ensure reproducibility of ML solutions.

In addition to submitting singular runs to HPC, Mantik offers the possibility to create *Run Schedules*. As an additional sub-tab in the *Runs* tab of the *Project* page, the user can set up a specific execution schedule for any completed *Run* with fixed frequencies or simply run a script on a HPC at a certain point in time.

The most prominent use case of *Run Schedules* in the MAELSTROM project are the proposed benchmarking cycles. While performing regular runs every 3 months proved to be not efficient in regards to the computation time as a resource, the *Run Schedules* function grants the researcher the ability to automatize benchmarking runs under the conditions and in the time intervals that are deemed appropriate by the individual user. This allows for more efficient use of HPC resources. The Mantik GUI displays characteristics of a *Run Schedule* like owner, end date, Cron expression, etc. Furthermore they can be deleted or edited.

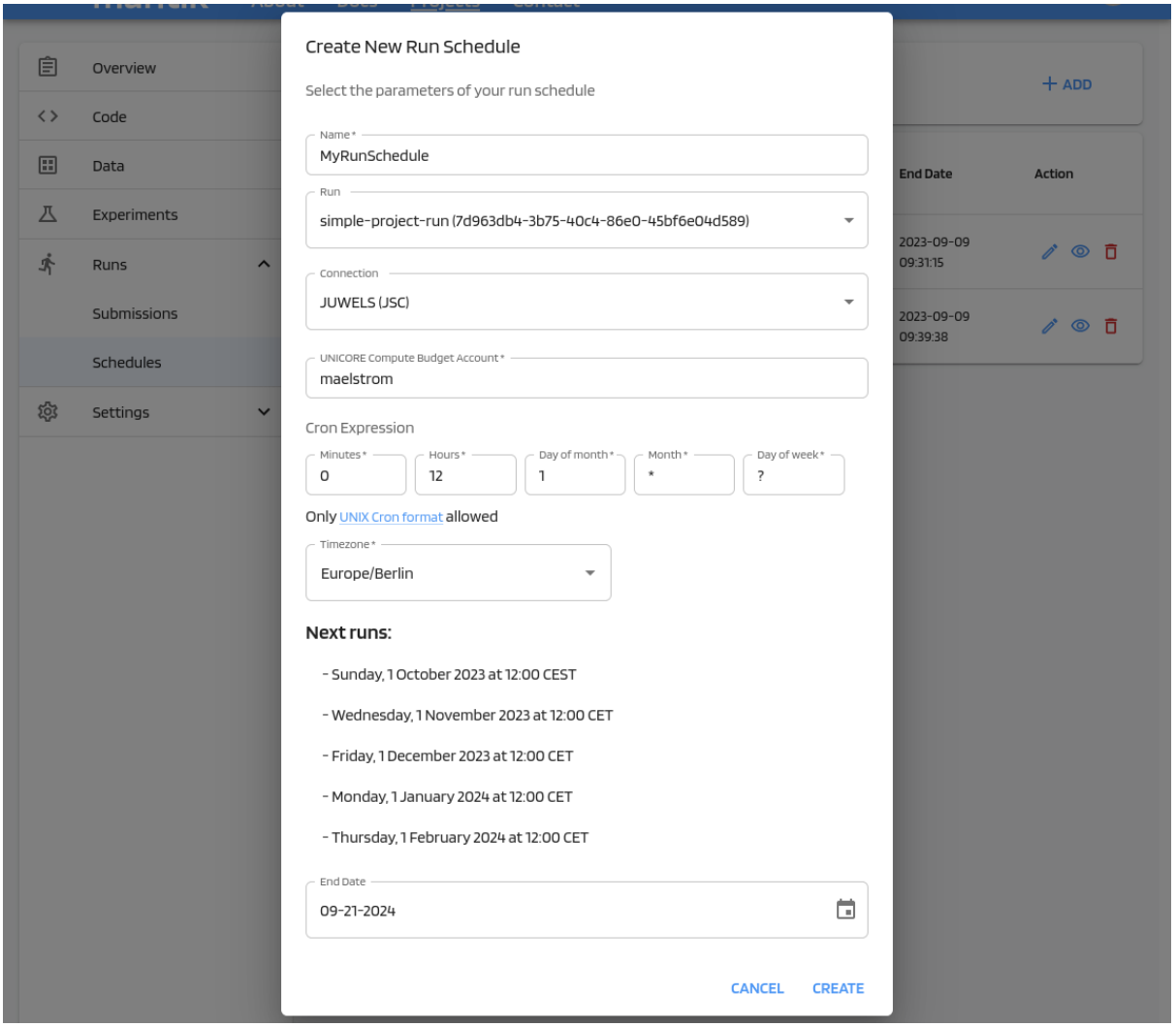
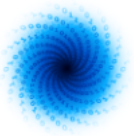


Figure 10: *The form to set up a run schedule. Every Run Schedule is run until its "End Date" at a user-defined frequency.*



3.2.3. Sharing, collaborating and recommending ML solutions

Cornerstones of research are collaboration and sharing of discovered insights. As a workflow platform intended to streamline the entire scientific process for researchers, Mantik offers features that foster communication and organisation of the workflow and results between collaborators including external interested parties.

Mantik simplifies the often tedious task of finding relevant projects for specific use cases or research questions by introducing labels. When entering the project settings through the *Settings* tab, the project can be labelled in the *General* section. A wide range of labels is available to choose from. In addition to a project itself, codes, data and experiment repositories can be assigned labels as well [17] via their respective edit button. Labels are sorted in categories like data size, data classes, programming frameworks, licences and more. In case a desired label is missing, new labels can be proposed via our Service Desk.

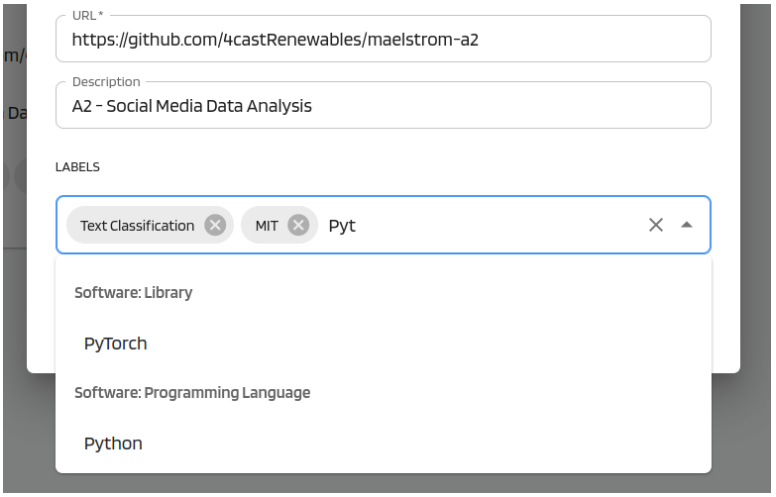
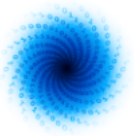


Figure 11: *Labels can be assigned to most entities in Mantik, which simplifies searching for the user and enables recommendation of relevant assets to the user.*

The labels act as the basis for recommending and finding publicly available or accessible private projects of interest. From the project page, users are able to find the appropriate projects by specifying relevant labels. Furthermore available projects can be filtered by their name, author or keywords in their project description.

To aid the management of projects, project based roles will be available in Mantik soon. They will be used to restrict user access to specific actions, i.e. creating, editing, deleting specific entities. There will be two main structures to organise users, *Groups* and *Organizations.* While *Groups* will only consist of users, *Organizations* will be able to add entire *Groups* and users alike. Five different roles are planned that offer different levels of access control.
Restricting user access has a lower priority for the MAELSTROM project, since all applications are

[17] Please note that labels have scopes.
Different sets of labels are available for code, data, experiments and projects.



meant to be publicly available. Nonetheless, allocating fixed roles can lead to improvements in organisation and overview for the users and increase the scope of the platform. A fully functional implementation of role based access control is planned as a feature for the final version of the workflow platform and will be explained in detail in the next Deliverable 2.5.

3.2.4. Additional features

In addition to core features explained above, Mantik offers a range of additional services like a CLI, that provides an alternative access point to most features discussed above, detailed documentation and a convenient way for users to submit bug reports.

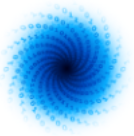
For debugging purposes or user reference, we offer CLI commands that allow the management of *Runs*. The user can submit and cancel *Runs* from the CLI similar to the platform. Our current implementation requires the user to access entities defined on the platform so runs remain reproducible. In the future, we plan to introduce a sandbox version of *Run* submission, which is optimised for quick and easy debugging but created runs will not be reproducible. Additionally, status information and logs of runs can be retrieved via the CLI. We are currently introducing this feature to the Mantik platform as it will greatly enhance user-friendliness of the platform. Should additional information be required when a *Run* is submitted to the HPC cluster, the user can interact with the Mantik via a collection of CLI commands[18]. A user can monitor and interact with a submitted *Run* via:

- `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.

For easy reporting of bugs found by users, we set up a Service Desk. A description of a bug can be sent to the Service Desk mail address. This will result in the automated creation of a Bug Ticket in the Mantik GitLab repository[19] and immediately available to our developers for investigation. Frequent updates on the state of processing are provided via mail as well.

All features implemented in Mantik are carefully documented. The documentation is separated into meaningful chapters allowing the user to easily find the desired information. *Tutorials* and *Quickstart* guides can be found as well as highly detailed documentation of the more sophisticated *CLI*. An entire chapter is dedicated to setting up *Mantik Projects, User Settings, Experiment Tracking* and *Running on HPC on the Mantik platform*. The entire documentation can be found here: https://mantik-ai.gitlab.io/mantik/

[18] https://mantik-ai.gitlab.io/mantik/cli.html
[19] https://gitlab.com/mantik-ai/mantik/-/issues?label_name%5B%5D=type%3A%3Abug



In the future Mantik plans to offer a wider range of available HPC clusters, allowing researchers to switch more flexibly to an alternative computing resource, should computation time be limited on their commonly used cluster.

3.3 Data Input/Output Acceleration (Task 2.5)

Well-designed data loaders are critical for deep-learning applications. Not only must data loaders read and process data efficiently, they must also support the workflow of the data scientist.

The key characteristics of effective data loaders for W&C applications that we have explored are:

- Performance: The average throughput of the pipeline, ideally delivering data faster than GPUs can consume them when training.
- Accessibility: The ability of a data loader to access data efficiently from external stores.
- Flexibility: The ability of the user to change aspects of the loading process on the fly (e.g. normalisation, feature engineering), without having to reformat the input data.

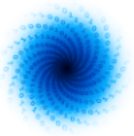
The relative importance of these characteristics is application and workflow dependent, as is the case for the 6 MAELSTROM applications. For long training runs, performance and scalability is most important. For extensive hyper-tuning or debugging, flexibility is important since constant reformatting of data stored on disk is tedious and time-consuming. For collaborative work, or for testing on multiple systems, having automated ways to retrieve data from external stores is critical for working efficiently.

In this section, we summarise the work we have done to improve each of the key characteristics.

3.3.1 CliMetLab

Data sharing, downloading and manipulation can be a time-consuming task limiting the value and use of benchmark datasets. CliMetLab seeks to overcome these challenges, by providing a tool that can fetch data from various sources, a caching mechanism for storing data locally for repeated use and interfaces to a range of data formats (catering to both meteorological and machine learning formats). Datasets can be described by plugins for CliMetLab, which describe where to find the data and how to interact with the data. The end result of this is that datasets can be used by new users by simply installing Python packages and then calling CliMetLab with the appropriate commands.

Downloading data such as those used in MAELSTROM can be challenging due to its long duration, sometimes spanning several hours. During this process, it's not uncommon to encounter connectivity issues, interruptions, and the need to restart the download. While running multiple downloads in parallel can be advantageous, often accelerating download speeds, it requires careful management. Researchers frequently resort to writing error-prone scripts to handle this issue, consuming a significant amount of their time. Climetlab offers a solution by automating this process with robust and sensible Python code, handling parallel transfers by default and error management.



Moreover, the common practice of keeping a local (often partial) copy of the downloaded data leads to additional data management complexities. Many users of this data neglect the development of robust solutions and predominantly rely on ad-hoc methods. In contrast, Climetlab employs a robust SQL database to effortlessly implement caching on behalf of the user. As a result, when a user accesses data for the first time, using `cml.load_dataset(...)`, the data is downloaded and cached. Subsequent access then seamlessly draws from this cache. Additionally, Climetlab handles cache maintenance.

Once in cache, the data becomes easily accessible to Python users through familiar objects in well-established packages such as Pandas, Xarray, and TensorFlow. This integration is implemented through Climetlab plugins developed within the MAELSTROM project, which rely on code from the CliMetLab core. The open-source nature of Climetlab's code stems from the repetitive nature of these tasks, which ultimately benefits the wider community.

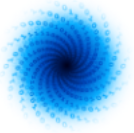
Plugins have been created for five MAELSTROM applications, developed by each application owner, leveraging the tooling provided by CliMetLab. These can all be installed using the Python package manager pip, as `pip install climetlab-plugin-package`, with the package names for each application shown in the table below.

Application #	CliMetLab pip package
1	`climetlab-maelstrom-yr`
2	`climetlab-maelstrom-social-media`
3	`climetlab-maelstrom-radiation`
4	`climetlab-maelstrom-ens10`
5	`climetlab-maelstrom-downscaling`
6	`climetlab-maelstrom-power-production`

***Table 1**: CliMetLab plugins available via pip.*

Note, that data the text of the Tweets used for application 2 are proprietary and can therefore not be shared. We share the Tweet Ids that allow users with access to Twitter's API to retrieve used Tweets.

Data for each application has been stored in public S3 buckets on the European Weather Cloud, making these datasets accessible to all. Each of these plugins is configured to know about the data storage location, but if in the future data is moved to a new location this can be easily updated by changing the address in the plugin. Each plugin supports the `to_xarray` functionality, giving a uniform way to explore the data in Python. For application 3, a special version of the dataset has been made available in the TensorFlowRecords format. This, loaded via the to_tfdataset method, provides a dataset which can be efficiently streamed from disk and incur no data loading overheads



which cause inefficient use of accelerator hardware. This is performant, scalable and accessible for this application, but is less flexible as it requires rewriting a copy of the dataset.

Below we explore two complementary approaches to improving data streaming capabilities as we establish optimal methodologies in weather and climate machine learning.

3.3.2 Improvements of the AP1 data loader

The AP1 data loader is designed to be performant and flexible. The pipeline allows decisions about normalisation, feature engineering, and spatial patching to be deferred to run time. As the whole dataset (6TB) does not fit into main memory, the AP1 data loader must stream data from disk. The input files are stored in uncompressed NetCDF format.

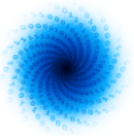
The initial implementation was based on reading data from NetCDF, then converting to numpy arrays. The many processing steps then transformed these arrays using standard numpy functions. The numpy arrays were finally converted to Tensorflow tensors. This achieved a processing performance of 0.10 GB/s (D3.4), measured on the JUWELS Booster system. Analysis of hardware utilisation quickly identified that CPU utilisation was low. Also, GPU utilisation was very low, suggesting that it was idle most of the time waiting for data from the data loader.

We then worked on optimising the code for each processing step. This resulted in a processing performance of 0.42 GB/s (D1.3). The data loader was still found to be the main bottleneck of the training.

We then rewrote the entire data loader to use Tensorflow's data processing API (tf.data). This allowed us to run each processing stage in parallel. It also allowed us to speed up each individual stage by splitting tensors into chunks that could be processed in parallel. This resulted in a performance of 1.45 GB/s when feeding a single GPU. We also implemented parallel data loading using Horovod to feed 4 GPUs, which yielded a performance of 2.11 GB/s. This was documented in D3.6.

A challenge with this loading process was that it required a lot of memory as many files were loaded in parallel. With 4 data loaders running in parallel we quickly ran out of memory. The problem was that the first stage of the pipeline (reading from file) was faster than the other stages and therefore often got far ahead of the other stages. We therefore had to force each loader to only fetch one file (10GB) from disk at a time. Fortunately, as reading from disk was not the bottleneck of the pipeline, limiting it to one file did not impair the overall performance. This setup with 4 GPUs uses a main memory footprint of around 350GB, which is less than the JUWELS Booster node's 512 GB of available memory.

Further analysis of JUWELS Booster job reports revealed that simultaneous multithreading (SMT) was not exploited as only 48 threads were specified in the job submission scripts. The poor scaling to 4 GPUs was a result of this, as the processing steps were limited by CPU threads. When SMT was used (by running with 96 threads instead of 48), the processing performance improved to 3.12 GB/s.



State of the data loader	Performance on JUWELS Booster
Initial implementation (D3.4)	0.10 GB/s
Code optimization (D1.3)	0.42 GB/s
Reimplementation using tf.data API (D3.6)	1.45 GB/s
Using 4 data loaders (D3.6)	2.11 GB/s
Enabling simultaneous multithreading	3.12 GB/s

***Table 2**: Progression of the processing performance of the AP1 data loader throughout the project period.*

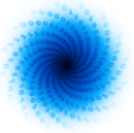
3.3.3 Improvements of the AP4 data loader

The primary objective of AP4 is to enhance weather forecasting accuracy through ensemble post-processing. In pursuit of this goal, we introduced the ENS-10 dataset, comprising ten ensemble members covering a two-decade period from 1998 to 2017. Subsequently, we conducted training sessions using diverse neural network models, such as UNet, MLP, and Transformer-based models, on this dataset to assess the performance of various models in our specific task [1].

All of our models underwent training utilising the PyTorch framework. In order to facilitate this process, our initial step involved saving the dataset in the NetCDF format. Subsequently, we implemented a PyTorch dataloader, specifically tailored to cater to the requirements of our models. It's noteworthy that during this endeavour, we observed a significant portion, approximately 90%, of the training time being consumed by the data loading and normalisation operations, primarily due to the substantial size of our dataset, which amounted to 3 terabytes (3TB). The comprehensive outcomes and findings derived from our training efforts are meticulously documented and can be found in Deliverable 1.3.

In order to address the substantial input/output (IO) bottleneck that was impeding the efficiency of our application, we implemented a pre-processing step. Our approach involves iterating our dataset, and for every temporal step, we extract the data from the NetCDF format and apply the normalisation step using pre-computed mean and standard deviations. Finally, the transformed data, along with its corresponding target, are saved in the Numpy format.

Our extensive experiments indicate that this preprocessing procedure can accelerate the overall training time by up to five times when compared to the use of the NetCDF format. We have compiled a comprehensive record of these detailed results, which can be perused in the dedicated Deliverable 1.4.



3.4 Deployment and Infrastructure (Task 2.6)

Deploying and reusing efficient Machine Learning pipelines on different infrastructures is the goal of many ML projects. The inference of a successfully trained model usually requires significantly less computational effort. To develop state-of-the-art ML models, researchers profit from being able to switch easily between local development and development on HPC clusters.

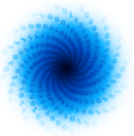
Requirements concerning security restrictions need to be approached from the side of the platform as well as from the side of the HPC infrastructure. Since users of Mantik need to have access to a HPC cluster beforehand, a third party credentials handler was implemented in the platform (see HashiCorp in section 3.2.1.2.). Administrators of HPC clusters are aware of the growing need to implement an API-based interface to their cluster to meet the growing demand for processing time more efficiently.

While the workflow platform is currently equipped to handle the UNICORE interface of JSC, integration of additional interfaces like FirecREST from the CSCS cluster are planned[20]. Since Mantik is designed with an abstract interface, it also allows for a quick adaptation of any other HPC facility. The only requirement is that it hosts a service that exposes a REST API such as UNICORE or FirecREST. In the case of UNICORE, no additional effort is required to enable users to execute their research applications on clusters that provide a UNICORE API. In case of other interface technologies that may be hosted by other research facilities, as soon as that interface has been adopted by Mantik, their users will be able to run their applications from the platform.
Hence, model training on specialised machines of E4, developed in WP3, is feasible as soon as UNICORE is deployed. The development of UNICORE dates back to a research project funded by the German Ministry of Education and Research in 1997 and has been progressively updated and maintained ever since[21]. Furthermore, the UNICORE interface has proven to be a promising candidate to be implemented on many different HPC infrastructures, like the clusters of the BSC, which reinforced the usage of UNICORE as the first implemented HPC interface for Mantik.

HPC infrastructure is a common asset of many fields of work requiring intensive calculations. A resource-saving and efficient usage is not just desirable but indispensable for the benefit of all. The EuroHPC sister project IO-SEA aims to improve the energy consumption of data migration on cluster nodes through a software package allowing to tag datasets, indicating future use cases and adjusting the computation accordingly. They recently released an open-access database[22] where IO-traces from applications can be uploaded. These traces are automatically analysed on the website and made available to the community. We have currently contributed IO-traces from AP1 to the public repository. Since MAELSTROM deals with especially large datasets, possibilities for collaboration will be investigated until the end of the project.

In the RED-SEA EuroHPC project, a VEF-Traces framework was developed for profiling the communication traffic in parallel applications. We used their tools to profile the AP1 application and

[20] https://user.cscs.ch/tools/firecrest/
[21] https://www.unicore.eu/about-unicore/history/
[22] https://hpcioanalysis.zdv.uni-mainz.de/



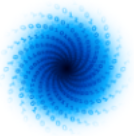
the traces were shared with the RED-SEA project. We will analyse these profiles in the remainder of the project to find insights that can improve our applications.

To optimise working with large datasets on exascale systems and allow for efficient partitioning, researchers from the MAELSTROM project worked together with the DEEP-SEA25 EuroHPC project. This collaboration resulted in the Data-Centric Machine Learning framework (DaCeML; Rausch 2022). By leveraging the Data-Centric IR (Ben-Nun 2019), DaCeML can represent the data movement within a neural network. Opportunities for further collaborations are being investigated.

The Mantik platform currently allows users to train, evaluate and analyse their results. Next, platform development will focus on enabling deployment of models from the platform. Currently, we plan to allow users to evaluate models by easily retrieving results from the platform. In addition, new users can quickly test models they are interested in by retrieving test predictions from deployed models.

The researchers of the MAELSTROM project use Deep500 recipes[23] for benchmarking and found their recipe produces the same results on three different infrastructures for the application A4 as seen in Deliverable 2.2. Additional software for performance optimization and portability of solutions is being developed.

[23] see Deliverable 2.3



4 MAELSTROM applications on the Mantik web-platform

The first and most important use case of Mantik in MAELSTROM is hosting all ML Applications developed during the project, allowing transparency of the research process and fostering collaboration between the consortium partners. Machine Learning algorithms offer a plethora of possible approaches to the problems of W&C forecasts. Different data sources like large reanalysis data sets, tweets from social media platform Twitter or data from mobile weather stations are utilised to create reliable and powerful yet energy efficient models. Communication is essential to progress in research and usage of HPC clusters with strict security measures becomes inevitable. Monitoring the entire process, enabling interested users to get a deeper understanding of Maelstrom's Applications is one of the main goals of Mantik.

On the following pages, features presented in chapter 3.2 are applied to the MAELSTROM applications. A1-A6 will be explained briefly in the context of Mantik and A2 will be examined in greater detail as an example.

4.1 General information on the applications A1-A6 in Mantik

All applications from the MAELSTROM project are now accessible through the Mantik platform with a basic model available. For now, available runs correspond to training runs used for benchmarking the Applications in the context of Deliverable D3.6. For this, configuration files in the respective `mlproject` folders were added to the repositories of all applications. In order to be integrated into the web-platform, the original repositories were forked or copied to GitHub.com on the company GitHub account of 4cast[24]. This step was necessary as Mantik currently only supports public Git repositories on GitHub and GitLab. We plan to allow access for additional Git hosts in the future to simplify migration to Mantik. All datasets necessary to train ML solutions of the applications A1-A6 are located on the JSC clusters[25]. *Experiments* were created for every Application to allow for tracking of parameters and metrics. After adding their credentials through the *Connection* form, users are able to execute their benchmarking runs on the JSC clusters through Mantik. Crucial training parameters and performance metrics that are essential for evaluating the performance of a model are tracked via the Mantik integration of MLflow. The essential tracking parameters and metrics including loss curve evolution can be viewed in the respective *Experiment* on Mantik's MLflow instance. All applications of MAELSTROM can be found as publicly available projects on Mantik (see Figure 12) . Assignment of appropriate labels for the different entities of the projects makes them easily accessible even to new users.

[24] https://github.com/4castRenewables

[25] An exception are a few large data sets for A4 located on CSCS cluster. An integration of this cluster is planned.

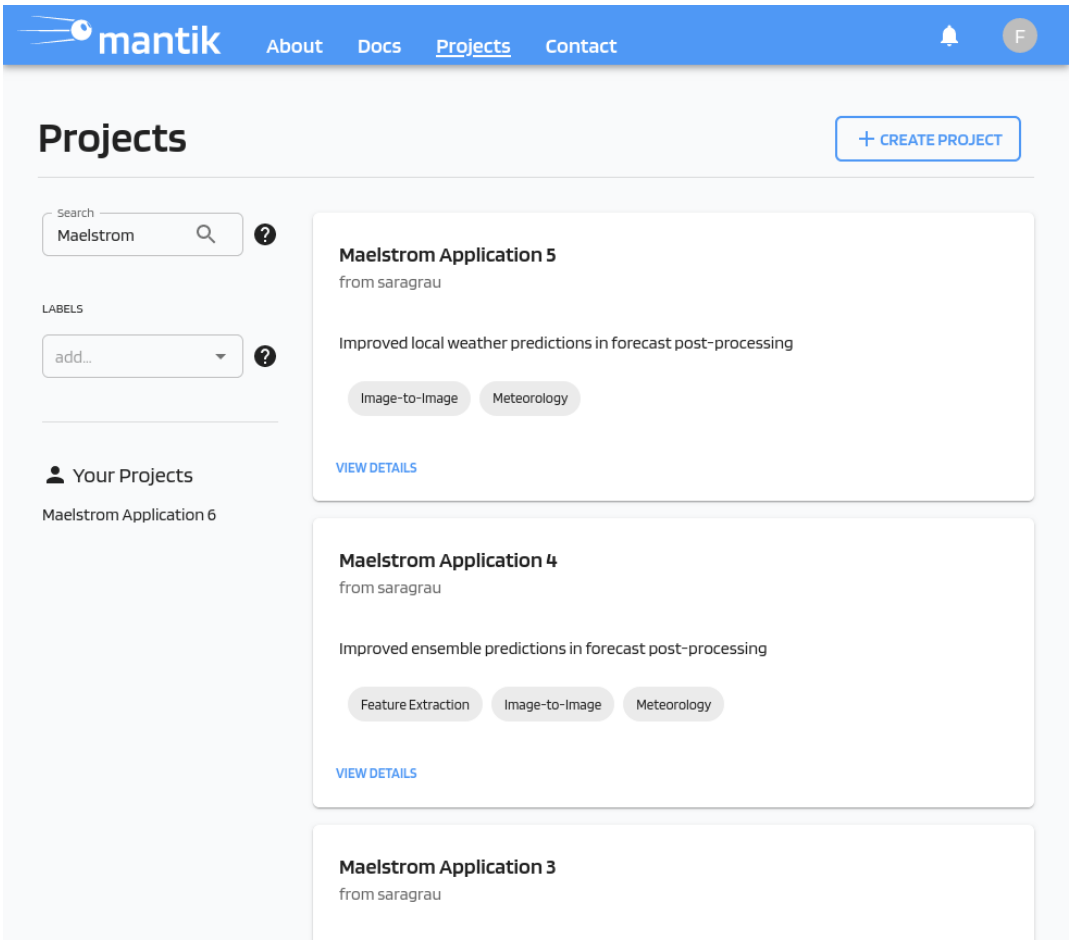
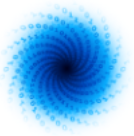


Figure 12: *MAELSTROM applications in Mantik.*

4.2 Example Application A2 on Mantik - Predict presence of rain from Tweets

The MAELSTROM application A2 aims to instrumentalise tweets as weather sensors. As an initial application, we aim to predict the presence of rain from Tweets. Generally, this problem can be phrased as a text classification task, where deep learning NLP models based on the transformer architecture (Vaswani et al., 2017 [4]) achieve state-of-the-art results (e.g. Yang et al., 2019 [5]). For this experiment, we rely on the most recent version of the model DeBERTa (DeBERTaV3), which is a popular transformer-based model(He et al., 2021b [6]).

Historical tweets from the years 2017-2020 are used that contain keywords related to the presence of weather phenomena. All tweets stem from the UK, so most popular NLP models, pretrained in English, can be applied. We only consider Tweets with location information, which is required for labelling. Tweets are labelled as “raining” or “not raining” based on precipitation values from ERA5-land (Muñoz Sabater 2019 [8]), a popular reanalysis set of weather forecasting data. For more details, see the latest update on the application as presented in Deliverable D1.4.



Next, we will demonstrate how Mantik enables the user to quickly train a baseline model on HPC and then compare it to a selection of model variants of the baseline version on the Mantik platform. Model comparison is an inherent feature of the MLflow GUI, so the user merely needs to log parameters and metrics in their script without further setup required. In addition, the results are quickly accessible to collaborators and everyone else as the *Project* is set to be publicly available. We start from the perspective of a new Mantik user and create a *Project* from scratch. We will demonstrate the following steps:

- Create Application A2 as a public *Project*.
- Create a *Code* repository that makes the application's github repository available to Mantik.
- Create a MLflow project file to handle the execution command and a Compute Backend config to specify the execution environment and parameters on HPC.
- Create an *Experiment* that holds logged information and enables analysis via the MLflow GUI.
- Create a *Connection* that holds our credentials to provide secure access to HPC.
- Create a *Run* that trains the baseline model and additional *Runs* where we vary hyper parameters.
- Analyse the results on the MLflow GUI.
- Create a Run schedule that can be used for automated benchmarking.

4.2.1 Project setup

We login to the Mantik platform and create our *Project* by specifying a name "Maelstrom Application 2", giving a short description that will be visible on the Mantik's overview page listing all available projects and a long description that aims to provide an overview of the project (see Figure 2 for a sample project page).

In our git repository on our development branch *develop*, we create a folder that contains the script used for splitting and training our model *build_dataset_rain_classifier.py* and *finetune_deberta_classifier.py,* respectively. In addition, we add our MLproject file *MLproject* that specified parameters and execution command and the UNICORE backend file *backend-config-venv-deberta.yaml*. The content of the latter two configuration files is shown in Figure 13 and Figure 14, respectively.

The MLproject file in our example has two entry points "split" and "train". These are used to separate steps of the ML developer workflow. Usually a different step calls a different script using a different set of input parameters such that command and parameters vary widely between entry points. While "split" is used to split our dataset into training, validation and test datasets based on script *build_dataset_rain_classifier.py*, "train" is used to train the model via script *finetune_deberta_classifier.py.* Note, that the parameters specified in the MLproject file as well as the whole compute backend file can easily be adopted on the Mantik platform as they become editable fields when creating a new *Run*.

On the other hand, the compute backend file specifies the modules, i.e. libraries that are required for our application to execute. Mantik will load them before job execution. Global variables are set like "GIT_PYTHON_REFRESH" in our example. Additionally, we specify the hardware specifications required for job execution (i.e., "Resources"). These include the duration required to perform the

job, e.g., amount of compute nodes, number of GPUs per node. Finally, as the MLflow project folder is uploaded as a whole to the cluster from the "Code" repository, we allow users to specify files that should not be included in the upload (i.e., "Exclude").

We create an *Experiment* "Classifier DeBERTa Era5", which will hold our tracked parameters and tracking. We will return to it during our analysis. To access the HPC cluster JUWELS *Booster* on the Juelich Supercomputing Center, we create a *Connection* "JSC", which requires us to specify our credentials that are provided to all users with access to the cluster.

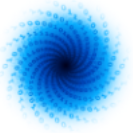
```
entry_points:
  split:
    parameters:
      …
      validation_size:
        type: float
        default: 0.2
      threshold_rain:
        type: float
        default: 7e-6
      …
    Command: >
      python build_dataset_rain_classifier.py
            …
            --validation_size {validation_size}
            --threshold_rain {threshold_rain}
            …
```

```
train:
  parameters:
    …
    batch_size:
      type: int
      default: 32
    weight_decay:
      type: float
      default: 0.01
    …
  Command: >
    python finetune_deberta_classifier.py
          …
          --batch_size {batch_size}
          --weight_decay {weight_decay}
          …
```

Figure 13: *Excerpt from the MLflow project file of Application 2 that specifies tasks (i.e, "entry_points") that can be executed and configured with Mantik. (Note, the file is split into two images for clarity)*

```
UnicoreApiUrl: https://zam2125.zam.kfa-juelich.de:9112/JUWELS/rest/core
Environment:
  PreRunCommand:
      Command: >
            module load Stages/2023 GCCcore/.11.3.0 Python/3.10.4 CUDA/11.7;
            source
/p/scratch/deepacf/maelstrom/maelstrom_data/ap2/venvs/ap2_deberta/bin/activate;
      ExecuteOnLoginNode: false
  Variables:
      GIT_PYTHON_REFRESH: quiet
Resources:
  Queue: develbooster
  Nodes: 1
  Runtime: 2h
Exclude:
  - "*.sif"
  - …
```

Figure 14: *Excerpt from the Compute backend config of Application 2 that specifies the execution environment on HPC (i.e, "Environment") and the allocated computational resources (i.e, "Resources").*



4.2.2 Train and analyse model

Now, we are ready to execute a training run of our model on HPC from the Mantik platform. In the "Create new run form" (Projects > Any project > Runs > Submissions > +Add), we configure our new run with the name "Training".

The run form (see Figure 15) automatically adjusts its size to display all parameters specified in the MLproject file and allows the user to change them from their default value in the form (see Figure 15, left panel). In addition, the backend config file is loaded into an editor, which enables the user to freely edit the file before submitting the edited result to HPC (see Figure 15, right panel). This for example greatly simplifies the process of changing queues on the fly without requiring the user to adjust the backend config file in their git repository. Here, we create a run using the default parameters. In addition, we create runs that train our model but with different sets of hyper parameters, i.e. batch size and learning rate. When decreasing the learning rate, we increase the number of training epochs to compensate for the increase in steps required to naively achieve a similar amount of weight modulation.

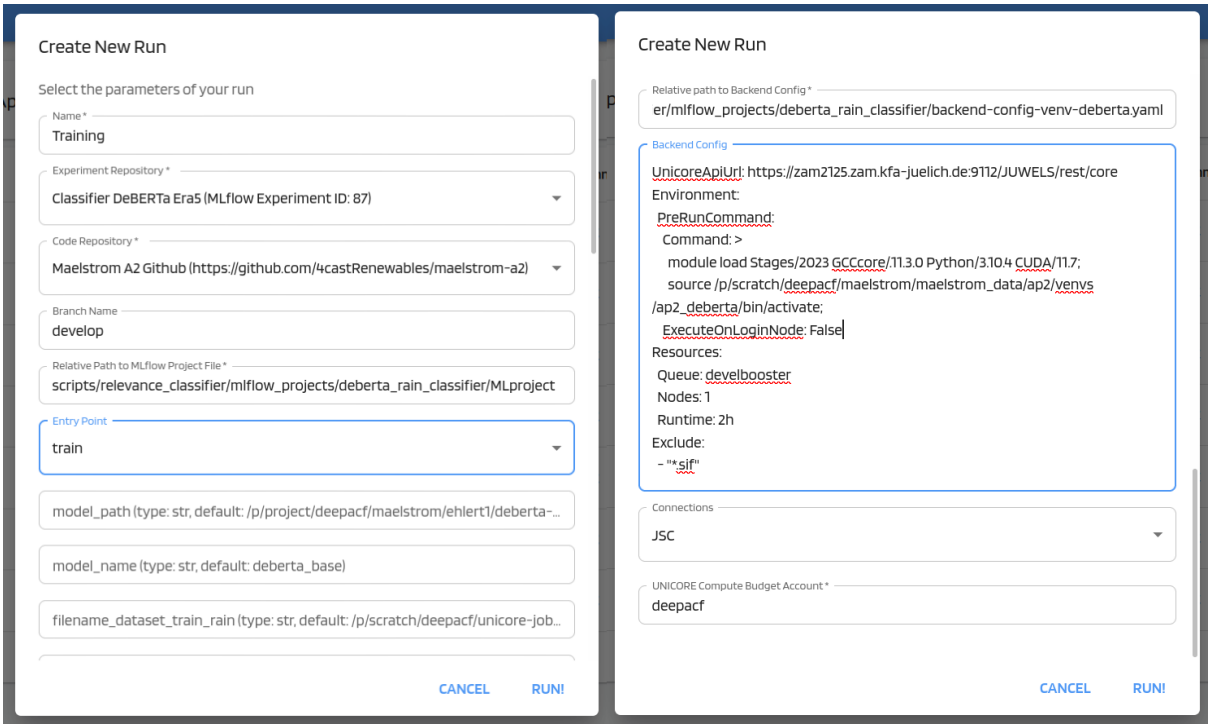


Figure 15: *Run form example from Application 2 for model training. Mantik allows the user to flexibly vary parameters and computing resources before Run execution. (Note, the same scrollable form is split into two images for clarity)*

After or during training of our model variants, we can use the integrated MLflow GUI to analyse and monitor our results. We can investigate parameters and metrics of a single run. In addition, artefacts like figures can be logged to every run, which can be viewed and downloaded from the GUI. Furthermore, runs can be compared with the inbuilt plotting functionality of the GUI. Figure 16 shows a sample plot that links the value of the hyper parameters to a couple of metrics, which are the f1-score computed on the evaluation set for the different classes “raining” and “not raining” called “eval_f1_raining”, “eval_f1_not_raining”, respectively. This enables the user to get a quick overview of their results. These can easily be shared by downloading data or figures or just providing collaborateurs access to the respective project on Mantik.

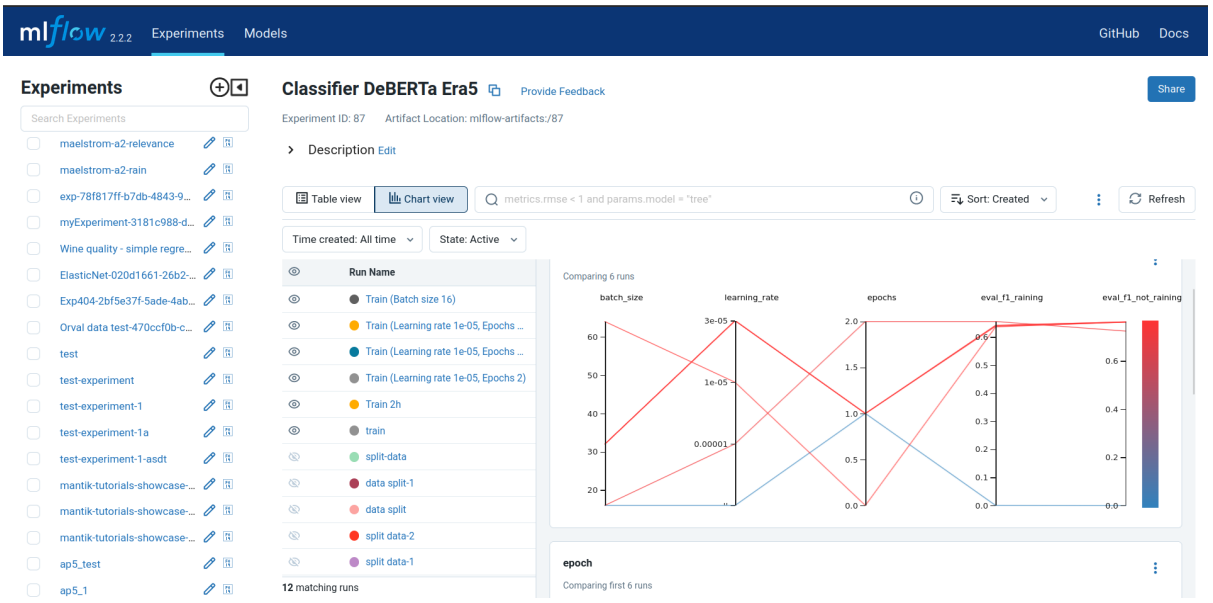


Figure 16: *Tracking performed for our Runs can be visualised via the integrated MLflow GUI, which allows the user to create figures for a quick analysis of their results.*

4.2.3 Setup automated benchmarking

Finally, we demonstrate the creation of a Run schedule. Run schedules correspond to runs that are re-executed on a user-defined schedule. The user can configure the period, time zone and final date of execution. Run schedules are initialised by specifying one of their former Runs and the execution schedule. The run will then be executed on the specified schedule. If no commit is provided, the concurrent runs will be based on the current state of the repository. Thus, if the original model is changed in the code, monitoring of the logged results of the subsequent runs will show changes in model performance. Therefore, an obvious use case is automated benchmarking.

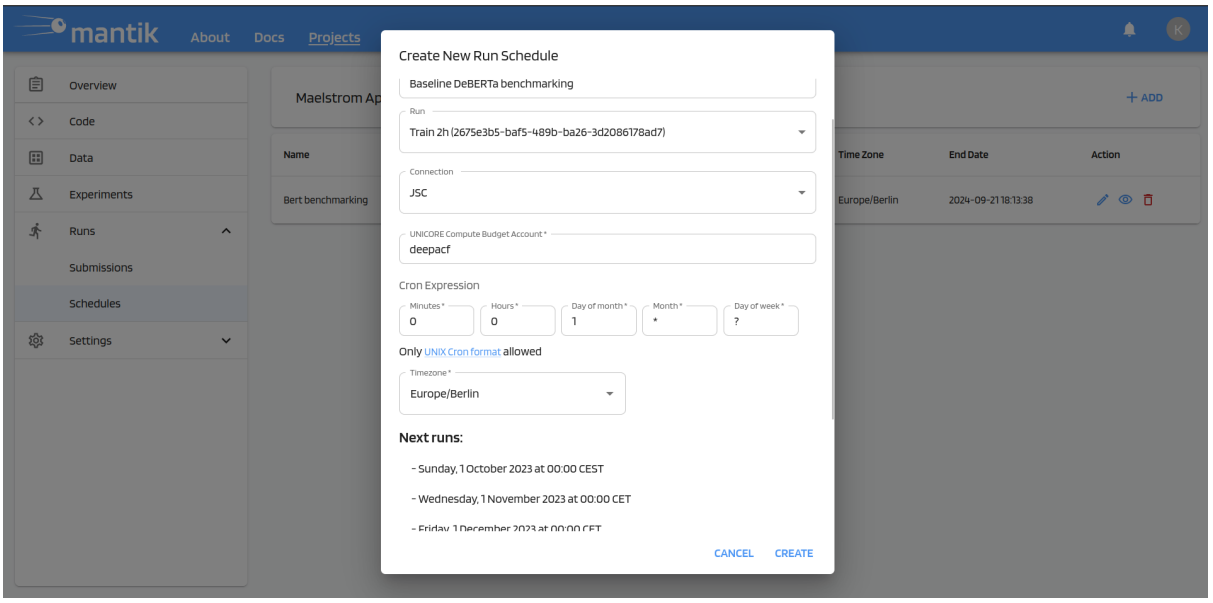
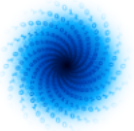


Figure 17: *We create a Run Schedule based on our training that trains our baseline model. Here, we use it to automatically benchmark our model.*



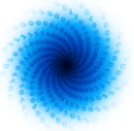
5 Conclusion

This report summarises progress on new custom solutions regarding data loading, the work performed on the GUI of the Mantik platform and the usage of CliMetLab as the basis of the weather data loading pipeline.

The features of the Mantik GUI were developed according to the requirements given by the scientists of WP1 as part of the Co-Design-Cycle. The platform now allows users to submit runs to HPC clusters without using a CLI, while still providing flexibility when specifying the exact work load, model parameters and cluster configurations. The Mantik platform empowers researchers to evaluate, monitor and share their results on-the-fly via the integration of the MLflow UI. Users can form groups to organise and share their projects and insights. A labelling system enables recommendations and quickly finding public projects of interest, which opens doors to research for interested aspiring scientists already during the early research phase. .

Broadly there are three directions to consider when creating optimal data loading systems for machine learning: performance, flexibility and accessibility. Through the use of CliMetLab plugins, access to datasets for five of the MAELSTROM applications is provided via a standardised and easy to use Python interface. This approach emphasises accessibility, but can also provide flexibility. Performance is undoubtedly a key component, particularly for larger machine learning models where inefficient data pipelines can lead to wasteful use of accelerator resources. In this report, we present how bespoke solutions to improve the performance of their data loading pipelines can help for the examples of MAELSTROM Applications 1 and 4. These adaptations significantly increase performance. Future work will be to incorporate these improvements into the CliMetLab plugins, or where sufficiently generic, provide standard CliMetLab tools for other researchers to build into their own applications.

As a research project dealing with typically large datasets, MAELSTROM presents to be an excellent candidate for collaboration with other research projects that develop software to increase efficiency when using hardware infrastructure. Benchmarking endeavours targeting a better monitoring of data flows and its energy consumption on exascale architecture are carried out by the EuroHPC sister projects IO-SEA, DEEP-SEA and RED-SEA. Scientists from MAELSTROM worked together with DEEP-SEA25, leading to the development of the Data-Centric Machine Learning framework (DaCeML; Rausch 2022). Tools developed by the RED-SEA project were used to profile the AP1 application and the traces were shared. A joint investigation with the IO-SEA project is planned for the upcoming months.



References

[1] Ashkboos, Saleh, et al. "ENS-10: A Dataset For Post-Processing Ensemble Weather Forecasts." Advances in Neural Information Processing Systems 35 (2022): 21974-21987.

[2] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. Hong, A. Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Fen Xie, Corey Zumar. “Accelerating the ML life cycle with MLflow.” In IEEE Data Engineering Bulletin

[3] Stefan Kesselheim, Andreas Herten, Kai Krajsek, Jan Ebert, Jenia Jitsev, Mehdi Cherti, Michael Langguth, Bing Gong, Scarlet Stadtler, Amirpasha Mozaffari, Gabriele Cavallaro, Rocco Sedona, Alexander Schug, Alexandre Strube, Roshni Kamath, Martin G. Schultz, Morris Riedel, Thomas Lippert. “*JUWELS Booster – A Supercomputer for Large-Scale AI Research.*” In High Performance Computing. ISC High Performance 2021. Lecture Notes in Computer Science(), vol 12761. Springer, Cham. https://doi.org/10.1007/978-3-030-90539-2_31

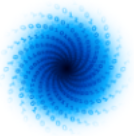
[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. “Attention is all you need.” In NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems. 2017 (pp. 6000–6010)

[5] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, Quoc V. Le. “XLNet: generalized autoregressive pretraining for language understanding.” In NIPS'19: Proceedings of the 33rd International Conference on Neural Information Processing Systems. 2019 (pp. 5753–5763)

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

[7] 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 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pages 66–77, Los Alamitos, CA, USA, may 2019. IEEE Computer Society.

[8] Muñoz Sabater, J. "ERA5-Land hourly data from 1950 to present. Copernicus Climate Change Service (C3S) Climate Data Store (CDS)". DOI: 10.24381/cds.e2161bac (Accessed on 27-04-2023) (2019)



Document History

Version	Author(s)	Date	Changes
0.1	Oliver Kindler (4cast)	04/09/2023	Initial draft
0.2	Kristian Ehlert (4cast), Fabian Emmerich (4cast)	07/09/2023	Additions and other edits
0.3	Saleh Ashkboos (ETH), Thomas Nipen (MetNorway), Matthew Chantry (ECMWF)	19/09/2023	Additions and other text passages
1.0		29/09/2022	Final version

Internal Review History

Internal Reviewers	Date	Comments
Peter Dueben (ECMWF)	28/09/2022	Minor comments and suggestions provided
Thomas Nipen (MetNorway)	28/09/2023	Minor comments and suggestions provided

Estimated Effort Contribution per Partner

Partner	Effort
ETH	1 PM
4cast	2 PM
Total	**3 PM**