D9.3 Initial Prototype of Benchmarking Toolkit

DAPHNE

Integrated Data Analysis Pipelines for Large-Scale Data Management, HPC, and Machine Learning

Version 1.2

PUBLIC

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 957407.

Document Description

In D9.3, the DAPHNE consortium presents the initial prototype of the benchmarking toolkit for the DAPHNE system.

D1.10 Refined Dissemination and Exploitation Plan			
WP9 – Initial Prototype of Benchmarking Toolkit			
Type of document	Report	Version	1.0
Dissemination level	CO/PU		
Lead partner	Hasso Plattner Institute		
Author(s)	Ilin Tolovski (HPI), Nils Strassenburg (HPI), Tilmann Rabl (HPI)		
Reviewer(s)	Pinar Tözün (ITU), Aleš Zamuda (UM)		
Contributors	HPI		

Revision History

Version	Revisions and Comments	Author / Reviewer
V1.0	[Initial draft]	Ilin Tolovski (HPI)
V1.1	[Draft before review]	Ilin Tolovski (HPI), Nils Strassenburg (HPI)
V1.2	[Draft after review]	Ilin Tolovski (HPI), Nils Strassenburg (HPI)

Abbreviations

Abbreviation	Term
HPI	Hasso Plattner Institute gGmbH
WP	Work package
T	Task
UC	Use case
BD	Big data
HPC	High performance computing
ML	Machine Learning
UMLAUT	Universal Machine Learning Analysis Utility
IDA	Integrated Data Analysis
HDF	Hierarchical Data Format
HDFS	Hadoop Distributed File System

Table of contents

Executive Summary

This deliverable describes the implementation of the Universal Machine Learning Analysis Utility (UMLAUT) prototype. We describe the implementation of our design decisions, the use cases for the system, as well as its performance implications on Integrated Data Analysis (IDA) pipelines. In this deliverable, we use terminology, definitions, and explanations from the official UMLAUT documentation, and its GitHub repository [1][2].

1 Introduction

The DAPHNE project focuses on creating a system for processing Integrated Data Analysis (IDA) pipelines. To evaluate the end-to-end performance of the system, we have conducted a survey of the state of the art in high performance computing (HPC), big data (BD), and machine learning (ML) benchmarks [3]. We find that the data analytics libraries and applications have been incorporating aspects from multiple domains to optimize the runtime performance and capabilities of the pipelines. However, there is a limited convergence across the benchmarking toolkits in these domains.

To evaluate IDA pipelines, we need to create an end-to-end toolkit that will encompass the different benchmarking aspects from the three domains. In Deliverable 9.2 – Initial Benchmark Concept and Definition, we have outlined the definition and specification of such a benchmarking toolkit [16].

Based on that concept, we created the Universal Machine Learning Analysis Utility (UMLAUT), a prototype of an end-to-end benchmarking toolkit for IDA pipelines. It consists of a tracker system for different sets of metrics, a database that persists the collected measurements, a command line interface, and a visualization tool that generates plots for result analysis.

In this deliverable, we present the first prototype of UMLAUT and its implementation. Specifically, we cover the implementation of the benchmarking aspects, metrics, and supported workloads. The benchmarking toolkit is implemented as a command line interface tool, that enables users to execute benchmarking workloads, but also analyze the collected measurements. Furthermore, the collected measurements can be visualized with the same tool.

With UMLAUT, we enhance the DAPHNE system with detailed runtime tracing and analysis of the system performance in multiple stages and with various metrics. By providing a detailed performance analysis, we open new research opportunities for innovation in various stages of the system stack, as well as implementation of different use cases.

UMLAUT is designed to analyze and benchmark Python IDA pipelines and scripts written in the DAPHNE Domain Specific Language (DSL). Throughout this document, we present the use cases covered in the prototype design phase, and showcase UMLAUTs capabilities to present some initial results. The used Python pipelines are derived from the DAPHNE Use Cases (UC 1) and various open-source pipelines. The DSL scripts cover various implementations of relational and HPC operators, such as SELECT, GROUP, JOIN, and matrix addition.

This deliverable is structured as follows. In Section 2, we present the UMLAUT system prototype, its design, and implementation. These include the benchmarking aspects considered, the metrics implementation, the data model, and the overall toolkit design. In Section 3, we outline two use cases for the benchmarking toolkit, i.e., benchmarking Python pipelines and benchmarking DAPHNE DSL Scripts. Finally, we conclude the deliverable report with a summary of the system.

2 UMLAUT - System Prototype

In this section, we introduce the Universal Machine Learning Analysis Utility (UMLAUT), our end-to-end benchmarking system prototype. In Section 2.1, we introduce the system requirements for the benchmarking toolkit. In Section 2.2, we describe the benchmarking aspects of UMLAUT introduced in Deliverable 9.2. Afterward, we describe the implementation of the aspects into metrics in Section 2.3 before we present the implementation of the data model introduced in Deliverable 9.2 and the overall toolkit design in Section 2.4. We explain how to install UMLAUT in Section 2.5, how to integrate it in Section 2.6, how to use the UMLAUT's command line interface in Section 2.7, and provide example visualization in Section 2.8.

2.1 System Requirements

In Deliverable 9.2 [16], we described the key requirements of the benchmarking toolkit, with respect to the coverage of IDA pipelines, benchmarking aspects, system under test and workloads. In this section, we present a condensed summary of the requirements, providing the necessary context for the implementation details of the benchmarking toolkit in the following sections.

2.1.1 IDA Pipelines

Integrated Data Analysis Pipelines consist of three distinct stages: computation, data processing, and training. For each stage, separate applications and frameworks can be integrated into a single IDA pipeline. To evaluate the performance of each stage of the pipeline, the toolkit needs to measure the performance of the individual applications, and the integrated frameworks. UMLAUT toolkit is required to measure the IDA pipeline on the application and middleware layers.

2.1.2 Benchmarking Aspects - Specification

In D9.2, we specify that the benchmarking toolkit needs to cover several benchmarking aspects of an IDA pipeline. To demonstrate the convergence of the application and middleware systems that make an IDA pipeline, the benchmarking toolkit needs to capture the end-to-end performance, and the performance of individual stages of the pipeline.

2.1.3 System under Test

We define the System under Test for our benchmarking toolkit in a modular fashion. As stated in Section 2.1.2. the toolkit needs to evaluate system performance on different granularities. To this end, we specify the system under test as one of the following: an individual method call, individual system operators spanning multiple methods and pipeline stages, and complete pipelines.

2.1.4 Workloads

In the prototype phase of the development, UMLAUT supports the following workloads, as shown in Deliverable 9.2 [16]:

- Data cleaning and preparation workloads – part of the data preprocessing stage of an IDA pipeline;
- Machine learning model training – part of the training stage of an IDA pipeline;
- Inference – part of the computation stage of an IDA pipeline.

2.2 Benchmarking Aspects - Implementation

As shown in Deliverable 9.1 and in our survey of Big Data, High Performance Computing, and Machine Learning Systems benchmarks [3], current benchmarks are evaluating only individual aspects of a data processing pipeline, and only marginally overlapping with any of the other stages. Big Data, High Performance Computing, and Machine Learning Systems consist of several stages, which can often overlap across domains. Evaluating any IDA pipeline for several aspects would require using multiple benchmarking toolkits to capture the complete performance footprint of the pipeline.

UMLAUT includes all aspects of an IDA pipeline. To achieve this, we implement supervised and valued metrics, the first of which provide an insight into end-to-end performance, such as: overall runtime, memory and energy consumption, as well as an overview of the resource utilization. On the other hand, we also collect domain specific, valued metrics, that collect measurements during the runtime, and are dependent on a particular stage of the pipeline. Such metrics include: hyperparameter sets, time-to-accuracy, per-epoch/stage loss, and similar.

The metrics are implemented in modular fashion, meaning that the user can decide which measurements will be collected based on the stage of the pipeline. A supervised measurement may be performed during a single method execution, during the execution of a group of methods for stage measurement, or throughout the complete pipeline. This allows both, coarse- and fine-grained performance analysis based on the evaluated methods in the pipeline.

The valued metrics, on the other hand, are tracked inside of a method, collecting measurements based on parameters generated during the method runtime. We present the metrics and their implementation in more detail in Section 2.3.

2.3 Benchmarking Metrics

In UMLAUT, we collect two types of benchmarking metrics, supervised and valued metrics. Supervised metrics evaluate the end-to-end performance of the system, with respect to time and resource consumption. Valued metrics are measured during the runtime and their values depend on the particular stage of the pipeline. In this section, we present their implementation in UMLAUT.

2.3.1 Supervised Metrics

With the supervised metrics, we capture the system performance, with regards to runtime, throughput, latency, CPU, memory, and energy consumption. The measurements for all supervised metrics are collected by decorator functions that annotate the methods that the toolkit needs to evaluate. Tracking a single method, or a set of methods in a pipeline allows us to perform both fine and coarse-grained performance analysis.

By benchmarking a single method, we can detect any bottlenecks caused by it, whereas when benchmarking multiple stages or the whole pipeline, we can analyze the end-to-end performance of the system. In Table 1, we provide a list of all supervised metrics and their implementation.

Type	Description
TimeMetric()	Measures the time in *seconds* it takes to execute the decorated code using time.perf_counter() [4].
MemoryMetric()	Measures the memory usage of the decorated code *in MB* using psutil memory_info().rss [5].
EnergyMetric()	Measures the energy consumption in *µJoule* of the decorated code using the pyRAPL library [6].
PowerMetric()	Measures the power consumption of the decorated code in *Watt* using the pyRAPL library [6].
LatencyMetric()	Measures the latency of the decorated code in *seconds/entries* using time.perf_counter() [4] to measure time.
ThroughputMetric()	Measures the throughput of the decorated code in *entries/second* using time.perf_counter() [4] to measure time.
CPUMetric()	Measures the CPU usage of the decorated code in *percent* using psutil cpu_percent() [5].

Table 1 Supervised metrics collected in UMLAUT.

2.3.2 Valued Metrics

With the valued metrics, we capture intrinsic properties of the pipeline, that are generated with the initialization, or the runtime of the pipeline. Currently, we have implemented four trackers of valued metrics that are relevant for performance of the pipeline: confusion matrix, hyperparameter, time-to-accuracy, and loss tracker.

Collecting these metrics allows us to combine results from different methods in a pipeline, such as measuring the influence of the batch size on the accuracy or perform any comparison between the stages of an IDA pipeline. In Table 2, we present all implemented value metrics and their trackers.

Type	Description
ConfusionMatrixTracker()	Tracks the confusion matrix.
HyperparameterTracker()	Tracks sets of hyper parameters across multiple executions.

TTATracker()	Tracks a list of accuracy values.
LossTracker()	Tracks a list of loss values.

Table 2 Valued metrics collected in UMLAUT.

2.3.3 Extensibility

The current implementation of UMLAUT allows limited extensibility of the set of metrics. To integrate a customized metric the user would need to implement an adequate decorator method for a supervised metric, or a tracker for a valued metric. To incorporate measurements from other software or hardware monitoring tools, the tool needs to be invoked inside an UMLAUT method. The users would need to define the method as a decorator, or a tracker method depending on the collected metric. All measurements for newly defined metrics are integrated under the corresponding metric type in the data model defined in Section 2.4.

2.4 Data Model & Toolkit Implementation

UMLAUT is implemented in Python as a standalone command line tool with an interactive interface. Using the CLI tool we can run benchmarking workloads and analyze completed pipelines and their results. All measurements are stored in a local instance of an SQLite database. Each database can contain a single or multiple set of measurements, based on the pipelines that were executed.

The storage location and naming are defined using a Benchmark class, that is mapped to a single database file and manages the measurements for one or several pipelines. For each metric that is to be collected, the user defines a method and pipeline decorator BenchmarkSupervisor. The decorator collects all specified metrics connected to it and transfers them to the Benchmark class.

The data model of the database consists of two tables, one for the Measurements, and one for the Pipeline Run. The PipelineRun table stores an ID, description, and a starting time of the run. The Measurement table stores the PipelineRun ID as a foreign key, a measurement ID as a primary key, a timestamp, description, datatype, binary data, and the measurement unit.

Multiple runs can be stored in a single database file, which allows access and comparison of multiple pipelines at once. A visual representation of UMLAUT's data model is shown in Figure 1.

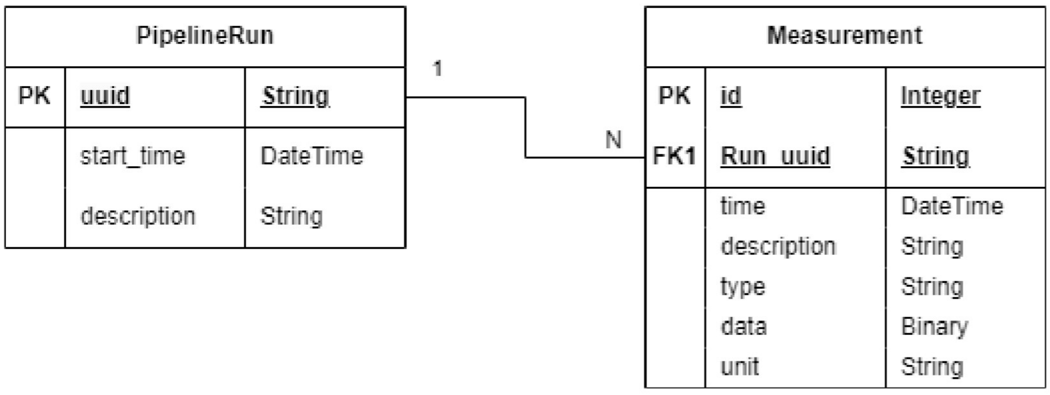


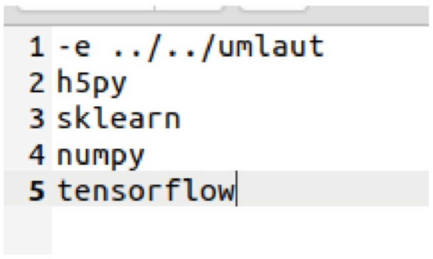
Figure 1 UMLAUT's data model.

2.5 System Installation

The current prototype is developed in Python 3.9 and can be locally installed as a Python package. The benchmarking toolkit is currently designed for internal use only and is not available as a PyPi package. The installation process consists of two steps. First, the GitHub repository is cloned with the following command:

```
git clone git@github.com:hpides/End-to-end-ML-System-Benchmark.git
```

Next, the UMLAUT package needs to be added as a local package to the requirements file that manages the python dependencies for the project. A requirements file, showing UMLAUT as in the dependency list is shown in Figure 2.



```
-e ../../umlaut
h5py
sklearn
numpy
tensorflow
```

Figure 2 UMLAUT as a dependency in a requirements file.

Once installed, it can be imported in any local Python projects or pipelines. All decorators and metrics are available upon import. In Figure 3., we show an example of using UMLAUT in an existing Python pipeline.

```python
import subprocess
from umlaut import Benchmark, TimeMetric, BenchmarkSupervisor, //
ThroughputMetric, LatencyMetric, MemoryMetric, PowerMetric, EnergyMetric, CPUMetric
```

Figure 3 Importing UMLAUT in a Python pipeline.

2.6 System Integration

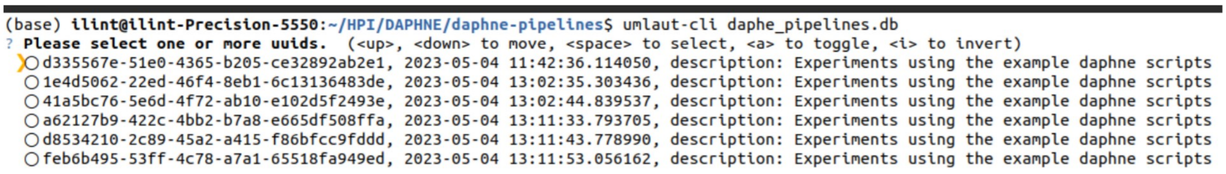
UMLAUT can be imported in any Python pipeline, upon installation. We provide a short description of installing and using the system in Section 2.5. The current version of UMLAUT is developed in Python 3.9 and compatible with commonly used Python libraries, such as scikit-learn [7], numpy [8], pandas [9], sqlalchemy [10], PyTorch [11], Tensorflow [12] and others. UMLAUT can be imported into existing pipelines by initiating the Benchmark class with a description and a database handle. Additionally, the user needs to specify the set of metrics that need to be associated with the benchmark. An example of initiating UMLAUT is shown in Figure 4.

```python
bm = Benchmark('group.db', description="Experiments using the example daphne scripts")
metrics = {
    "time": TimeMetric('sql group time'),
    "memory": MemoryMetric('sql group memory', interval=0.1),
    "power": PowerMetric('sql group power'),
    "energy": EnergyMetric('sql group energy'),
    "cpu": CPUMetric('sql group cpu', interval=0.1)
}
```

Figure 4 Initialization of UMLAUT in a pipeline

2.7 Command Line Interface

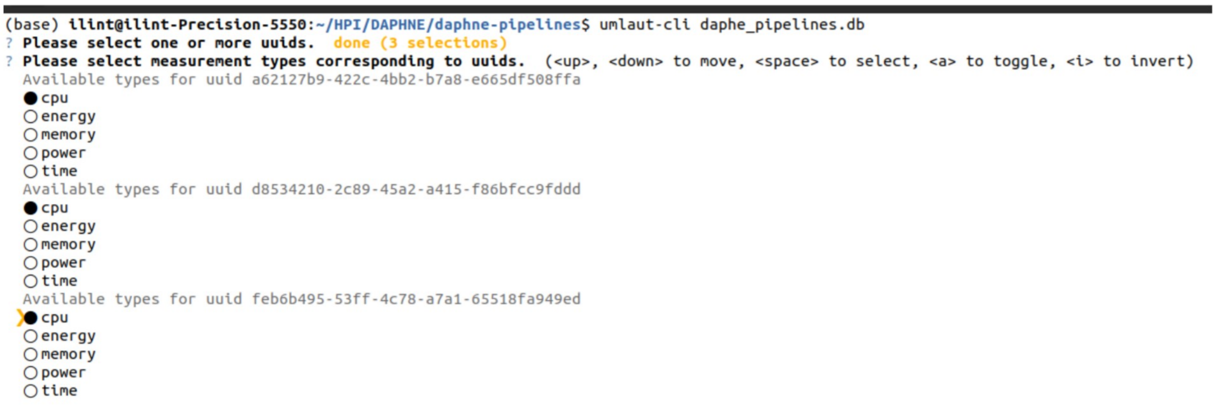
UMLAUT is managed through a command line interface (CLI) toolkit. To access the collected measurements, the user needs to provide the database handle for the workloads.



```
(base) ilint@ilint-Precision-5550:~/HPI/DAPHNE/daphne-pipelines$ umlaut-cli daphe_pipelines.db
? Please select one or more uuids.  (<up>, <down> to move, <space> to select, <a> to toggle, <i> to invert)
 ❯○d335567e-51e0-4365-b205-ce32892ab2e1, 2023-05-04 11:42:36.114050, description: Experiments using the example daphne scripts
  ○1e4d5062-22ed-46f4-8eb1-6c13136483de, 2023-05-04 13:02:35.303436, description: Experiments using the example daphne scripts
  ○41a5bc76-5e6d-4f72-ab10-e102d5f2493e, 2023-05-04 13:02:44.839537, description: Experiments using the example daphne scripts
  ○a62127b9-422c-4bb2-b7a8-e665df508ffa, 2023-05-04 13:11:33.793705, description: Experiments using the example daphne scripts
  ○d8534210-2c89-45a2-a415-f86bfcc9fddd, 2023-05-04 13:11:43.778990, description: Experiments using the example daphne scripts
  ○feb6b495-53ff-4c78-a7a1-65518fa949ed, 2023-05-04 13:11:53.056162, description: Experiments using the example daphne scripts
```

Figure 5 Overview of the pipeline set in the UMLAUT database.

Measurements can be selected individually for a single pipeline or compared against several pipeline runs. Similarly, multiple measurements for a single pipeline can be queried in the CLI interface. An example query is shown in Figures 5 and 6.



```
(base) ilint@ilint-Precision-5550:~/HPI/DAPHNE/daphne-pipelines$ umlaut-cli daphe_pipelines.db
? Please select one or more uuids.  done (3 selections)
? Please select measurement types corresponding to uuids.  (<up>, <down> to move, <space> to select, <a> to toggle, <i> to invert)
  Available types for uuid a62127b9-422c-4bb2-b7a8-e665df508ffa
  ●cpu
  ○energy
  ○memory
  ○power
  ○time
  Available types for uuid d8534210-2c89-45a2-a415-f86bfcc9fddd
  ●cpu
  ○energy
  ○memory
  ○power
  ○time
  Available types for uuid feb6b495-53ff-4c78-a7a1-65518fa949ed
 ❯●cpu
  ○energy
  ○memory
  ○power
  ○time
```

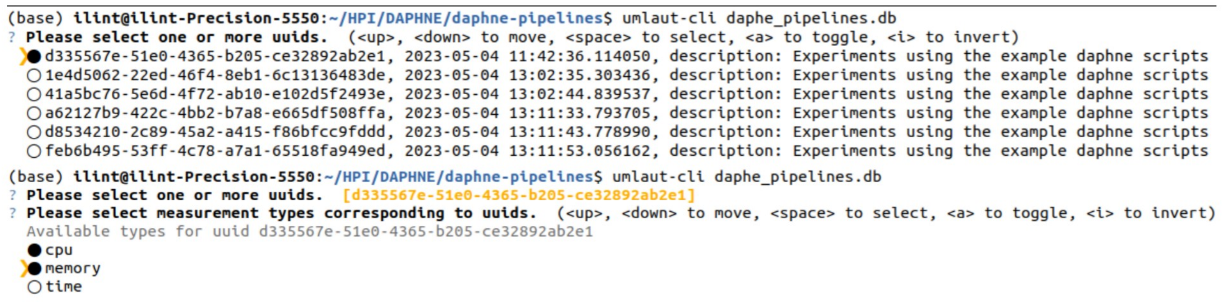
Figure 6 Comparing the CPU usage for 3 pipelines.

2.8 Visualization

The CLI toolkit also enables users to generate plots from the collected measurements. Measurements can be shown individually, i.e., a single plot can present a single pipeline run, or provide a comparison between multiple runs. Each plot collects the measurements for a single metric. They are generated and shown through the standard Python Matplotlib Viewer, where users can interact with the plot to hover on individual values or zoom in for more detail. In the next two sections, we show examples of the types of plots that can be generated through the CLI interface.

2.8.1 Visualizing measurements from single pipeline

We can query the UMLAUT database for the results of a single pipeline. This is done by choosing one of the recorded pipelines, as shown in Section 2.6. For the selected pipeline, we choose the metrics that we want to visualize. For each metric, a separate plot is generated. An example of generating the CPU and Memory usage for one pipeline is shown in Figures 7 and 8.



```
(base) ilint@ilint-Precision-5550:~/HPI/DAPHNE/daphne-pipelines$ umlaut-cli daphe_pipelines.db
? Please select one or more uuids.  (<up>, <down> to move, <space> to select, <a> to toggle, <i> to invert)
 ❯●d335567e-51e0-4365-b205-ce32892ab2e1, 2023-05-04 11:42:36.114050, description: Experiments using the example daphne scripts
  ○1e4d5062-22ed-46f4-8eb1-6c13136483de, 2023-05-04 13:02:35.303436, description: Experiments using the example daphne scripts
  ○41a5bc76-5e6d-4f72-ab10-e102d5f2493e, 2023-05-04 13:02:44.839537, description: Experiments using the example daphne scripts
  ○a62127b9-422c-4bb2-b7a8-e665df508ffa, 2023-05-04 13:11:33.793705, description: Experiments using the example daphne scripts
  ○d8534210-2c89-45a2-a415-f86bfcc9fddd, 2023-05-04 13:11:43.778990, description: Experiments using the example daphne scripts
  ○feb6b495-53ff-4c78-a7a1-65518fa949ed, 2023-05-04 13:11:53.056162, description: Experiments using the example daphne scripts
(base) ilint@ilint-Precision-5550:~/HPI/DAPHNE/daphne-pipelines$ umlaut-cli daphe_pipelines.db
? Please select one or more uuids.  [d335567e-51e0-4365-b205-ce32892ab2e1]
? Please select measurement types corresponding to uuids.  (<up>, <down> to move, <space> to select, <a> to toggle, <i> to invert)
  Available types for uuid d335567e-51e0-4365-b205-ce32892ab2e1
  ●cpu
 ❯●memory
  ○time
```

Figure 7 Interaction with the CLI to generate plots for a single pipeline.

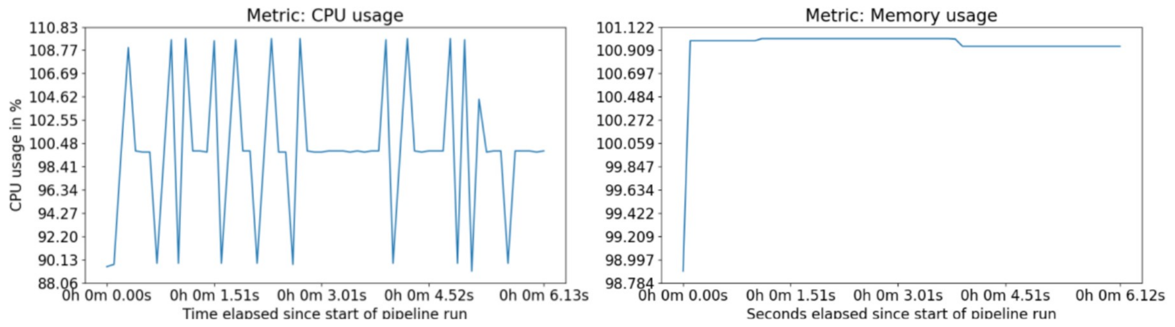
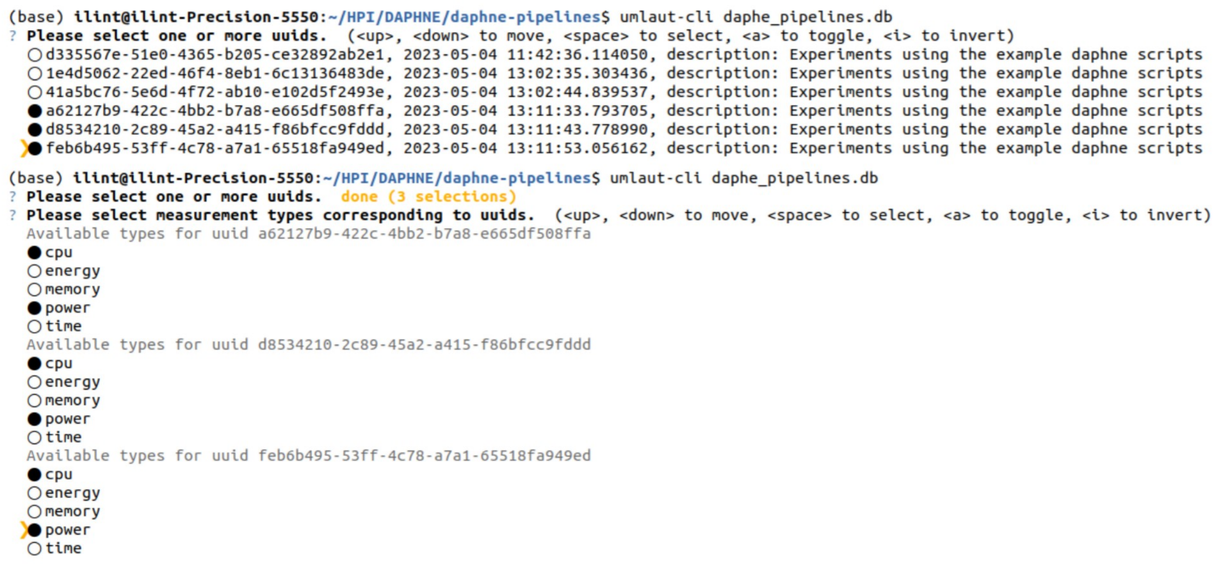


Figure 8 Visualizing CPU and Memory usage for a single pipeline.

2.8.2 Visualizing measurements from multiple pipelines

In UMLAUT we can also visualize the measurements for multiple pipelines in a single plot for better perfomance analysis and comparison across multiple pipeline runs. To do so, we select the pipelines of interest from the measurement database. For each pipeline, the same metric needs to be selected so they can be plotted on the same figure. In Figures 9 and 10, we show an example of generating the plots for CPU and Power usage for three pipelines through UMLAUT's CLI.



```
(base) ilint@ilint-Precision-5550:~/HPI/DAPHNE/daphne-pipelines$ umlaut-cli daphe_pipelines.db
? Please select one or more uuids.  (<up>, <down> to move, <space> to select, <a> to toggle, <i> to invert)
  ○d335567e-51e0-4365-b205-ce32892ab2e1, 2023-05-04 11:42:36.114050, description: Experiments using the example daphne scripts
  ○1e4d5062-22ed-46f4-8eb1-6c13136483de, 2023-05-04 13:02:35.303436, description: Experiments using the example daphne scripts
  ○41a5bc76-5e6d-4f72-ab10-e102d5f2493e, 2023-05-04 13:02:44.839537, description: Experiments using the example daphne scripts
  ●a62127b9-422c-4bb2-b7a8-e665df508ffa, 2023-05-04 13:11:33.793705, description: Experiments using the example daphne scripts
  ●d8534210-2c89-45a2-a415-f86bfcc9fddd, 2023-05-04 13:11:43.778990, description: Experiments using the example daphne scripts
 ❯●feb6b495-53ff-4c78-a7a1-65518fa949ed, 2023-05-04 13:11:53.056162, description: Experiments using the example daphne scripts
(base) ilint@ilint-Precision-5550:~/HPI/DAPHNE/daphne-pipelines$ umlaut-cli daphe_pipelines.db
? Please select one or more uuids.  done (3 selections)
? Please select measurement types corresponding to uuids.  (<up>, <down> to move, <space> to select, <a> to toggle, <i> to invert)
  Available types for uuid a62127b9-422c-4bb2-b7a8-e665df508ffa
  ●cpu
  ○energy
  ○memory
  ●power
  ○time
  Available types for uuid d8534210-2c89-45a2-a415-f86bfcc9fddd
  ●cpu
  ○energy
  ○memory
  ●power
  ○time
  Available types for uuid feb6b495-53ff-4c78-a7a1-65518fa949ed
  ●cpu
  ○energy
  ○memory
 ❯●power
  ○time
```

Figure 9 Interaction with the CLI to generate plots for multiple metrics and pipelines.

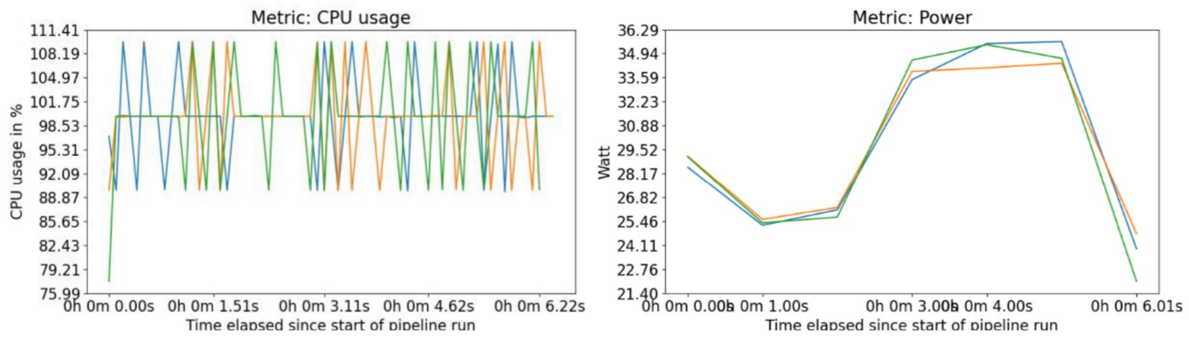


Figure 10 Visualizing CPU and Power usage for multiple pipelines.

3 Use Cases

UMLAUT can benchmark Python pipelines and DAPHNE DSL scripts. In this section, we present two use cases: (1) benchmarking the Python pipelines for DAPHNE Use Case 1 – Earth Observation and other open-source data analytics pipelines, and (2) benchmarking DAPHNE DSL example scripts for GROUP, SELECT, JOIN, and matrix addition workloads. Current implementations of the Python pipelines were executed on a single node, with UMLAUT running on the same node. We would consider distributed benchmarking as future work.

3.1 Benchmarking Python Pipelines

In this section, we outline two examples of Python pipelines that were used to develop UMLAUT. We focused on DAPHNE Use Case pipelines, such as the ones included in Use Case 1 – Earth Observation. We also included open-source analytics pipelines. In this deliverable, we present an Anomaly Analysis use case published by Backblaze [14]. Including open-source pipelines is an important part of the development of UMLAUT also according to the DAPHNE project timeline since the benchmarking toolkit will be available after the project completion.

3.1.1 Earth Observation (DAPHNE UC-1)

The Earth Observation Use Case classifies climate zones based on information collected from satellite images and other sensor data. The images provide data on the surface structure of the area as well as other anthropogenic attributes. The image data is collected from the Sentinel-1 and Sentinel-2 corpora. They consist of satellite imagery in various resolutions and auxiliary datasets [13].

The labeled datasets are available for research and analysis purposes. They can be queried with different resolutions, and image sizes, resulting in a wide range of datasets that differ in size and number of samples. Considering the ranging size of the datasets, we can evaluate the different stages of the pipelines at different scales.

The complete use case workload consists of three pipelines: training, testing, and benchmarking pipeline. The training pipeline consists of three stages: data loading, model compilation, and model training. It is important to note that the data preparation stage is minimal in this use case since the provided Sentinel datasets are already curated and labeled. Hence, the data preparation phase is considered an integral part of both the training and testing stages.

The testing pipeline consists of the same data preparation phase, and an inference phase. Similarly, to the training pipeline, the data preparation per testing sample is integrated in the testing phase.

We used UMLAUT to measure the time, and resource utilization by the training and testing phases.

3.1.2 Backblaze Anomaly Analysis

The Backblaze Anomaly Analysis Use Case is based on open-source data published by the cloud storage company Backblaze. The published data contains information on the operability of hard drives in their data centers collected over several years. Data on different hard drive models of different capacities is provided, mainly considering their failure frequency, and SMART statistics monitoring the health of the drives.

The Backblaze Anomaly Analysis workload consists of three pipelines: data preparation, train and test pipeline, and benchmarking pipeline. The available datasets for this use are raw hardware measurements, that need to be prepared before further analysis can be performed. These measurements represent use cases where raw data is collected and need to be prepared before any statistical or machine learning analysis can be performed. With this use case, we perform analysis on four stages of an IDA pipeline: data loading, data preprocessing, training, and testing stage.

The data loading consists of reading HDF files from HDFS file systems and parsing them to a tabular format, so feature engineering can be performed. In the next stage, these files are analyzed, and features are extracted. To prepare the data for the training stage, we apply data preprocessing techniques, such as missing value imputation, normalization and categorization, and under/over-sampling to balance the class representation.

The training and testing phases are similarly designed as in the Earth Observation use case, the main difference being the training algorithm used. Instead of deep neural networks, in this use case, we use a Random Forest Classifier.

UMLAUT was used to measure the system's performance on a single node with respect to time and resource consumption during all stages, as well as to, track and collect metrics specific to the training process, such as tracking the confusion matrix, collecting the classifier's hyperparameters.

3.2 Benchmarking DAPHNE DSL Scripts

For our reference DAPHNE DSL pipelines, we used the implementations of the SQL GROUP, JOIN, and SELECT operators that are prototyped in the DAPHNE system repository [15]. The DAPHNE DSL scripts are called from a Python pipeline that executes the DAPHNE binary through a system call. Such scripts and execution pipelines have been included in various demonstrations of DAPHNE's system performance shown by other Work Packages and Consortia Partners [18]. The DSL scripts consist of relational operator implementations, as well as an efficient matrix addition workload. These workloads cover three stages of an IDA pipeline: data generation, efficient data processing, and data storage.

The data generation stage invokes DAPHNE's random data generator that creates matrix-like data in a designated shape. The matrices are then processed by invoking DAPHNE operators, such as SELECT, GROUP, JOIN, and a matrix addition in each of the pipelines. The resulting matrix is then stored locally.

In contrast to the Python pipelines presented in Section 3.1, the DSL scripts are self-contained, providing all stages of the workload in a single pipeline. In this use case, we use UMLAUT to measure the end-to-end performance of the DSL scripts. To microbenchmark the individual stages of the DSL scripts, an adequate interface to UMLAUT's database needs to be added to the DAPHNE system and further implemented in UMLAUT.

4 Extensions of UMLAUT

During the development of the UMLAUT prototype, we have noted a set of improvement points that can be addressed in the production version of the system. In this section, we present them as a list of planned extensions for the final version.

One aspect that can benefit UMLAUT to obtain more detailed insights into the performance of DAPHNE pipelines is the microbenchmarking of DAPHNE DSL scripts. On that level, the user can benefit from the performance analysis of individual DAPHNE DSL operators and their impact on the DSL scripts. To achieve this, we plan a collaboration with the Consortia Partners from Work Packages 4 and 5, to connect DAPHNE's runtime engine together with the UMLAUT interface.

Additionally, we plan to extend UMLAUT to evaluate the performance of distributed IDA pipelines. In this scenario, we plan to evaluate a distributed version of the Use Case Pipelines described in Section 3.1.

5 Conclusion

In this deliverable, we present the initial prototype of the UMLAUT benchmarking toolkit. Based on the concepts and specification outlined in Deliverable 9.2 [16], we present their implementation in a working prototype. We present the implementation of different benchmarking aspects. Also, we describe the implementation of the supervised metrics available in UMLAUT that measure the system performance of the workloads, and the implementation of valued metrics that are used to track stage-specific parameters of the pipeline.

UMLAUT is implemented as a command line interface tool with visualization capabilities. We present instructions for the system's installation and integration in working pipelines, as well as an example of querying the database of the measurements. Finally, we also present two use cases that were used in the development stage of UMLAUT. We outline the stages, and individual pipelines of each of the use cases, as well as the aspects that can be evaluated by UMLAUT. In the next deliverable, we plan to further test and expand the UMLAUT features into a final prototype that supports end-to-end benchmarking of IDA pipelines in the DAPHNE system.

6 References

[1] UMLAUT Documentation. https://hpides.github.io/End-to-end-ML-System-Benchmark/index.html, accessed 03.05.2023.

[2] UMLAUT GitHub Repository. https://github.com/hpides/End-to-end-ML-System-Benchmark, accessed 03.05.2023.

[3] Ihde, Nina, et al. "A Survey of Big Data, High Performance Computing, and Machine Learning Benchmarks." Technology Conference on Performance Evaluation and Benchmarking. Springer, Cham, 2021.

[4] Python Standard Library Documentation – time: https://docs.python.org/3/library/time.html#time.perf_counter, accessed 03.05.2023

[5] psutil Documentation: https://psutil.readthedocs.io/en/latest/, accessed 03.05.2023

[6] pyRAPL Documentation: https://pyrapl.readthedocs.io/en/latest/quickstart.html, accessed 03.05.2023

[7] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

[8] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2.

[9] McKinney, W. Data Structures for Statistical Computing in Python. In Proceedings of the 9th Python in Science Conference (pp. 56-61) (2010). doi:10.25080/Majora-92bf1922-00a.

[10] Bayer, M. (2012). SQLAlchemy. In Brown, A. & Wilson, G. (Eds.), The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks.

[11] Paszke, A. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv preprint arXiv:1912.01703.

[12] Abadi, M. et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[13] Corpernicus Open Access Hub. https://scihub.copernicus.eu/, accessed 05.05.2023.

[14] Backblaze Hard Drive Data and Stats. https://www.backblaze.com/b2/hard-drive-test-data.html, accessed 05.05.2023.

[15] DAPHNE System Repository. https://github.com/daphne-eu/daphne, accessed 16.05.2023.

[16] DAPHNE Deliverable 9.2: Initial Benchmark Concept and Definition.

[17] DAPHNE Deliverable 9.1: A survey of Benchmarks from DM, HPC, and ML Systems.

[18] DAPHNE DSL HPC Experiments. https://github.com/daphne-eu/daphne/tree/deploy-Vega/deploy/experiments-CIDR-2022-Vega-microbenchmarks, accessed 23.05.2023.