SEcure Decentralised Intelligent Data MARKetplace

D3.2 Energy efficient AI-based toolset for improving data quality. Final version

Document Identification	
Contractual delivery date:	31/07/2025
Actual delivery date:	31/07/2025
Responsible beneficiary:	NUID UCD
Contributing beneficiaries:	NUID UCD, ATOS, EGM, INRIA, MYT, SIE, SURREY, UC WINGS
Dissemination level:	PU
Version:	1.0
Status:	Final

Keywords:

Data marketplace, machine learning, data quality, data curation, data processing pipeline, energy efficiency, model optimisation, trade-offs, communication cost

Document Information

List of Contributors	
Name	**Partner**
Elias Tragos Diarmuid O'Reilly Morgan Erika Duriakova Honghui Du Aonghus Lawlor Neil Hurley	NUID-UCD
César Camarazana Joaquín García Maxime Costalonga	ATOS
Luc Gasser Thomas Bousselin Franck Le Gall	EGM
Shahin Abdoul Soukour Emile Royer Maroua Bahri Nikolaos Georgantas	INRIA
Ioannis Tsogias Nikos Babis	MYT
Gabriel Danciu Stefan Jarcau	SIE
Tarek Elsaleh Peipei Wu Mahrukh Awan Sneha Hanumanthaiah Adrian Hilton	SURREY

List of Contributors	
Name	**Partner**
Pablo Sotres Laura Martín Juan Ramón Santana Jorge Lanza Luis Sánchez	UC
Panagiotis Vlacheas Grigoris Koutantos	WINGS

Document History			
Version	**Date**	**Change editors**	**Change**
0.1	13/02/2023	NUID-UCD	First draft of ToC
0.2	23/04/2025	ATOS	Added sections 4.3.1 and 5.2.3
0.3	29/04/2025	WINGS	Contributions to Sections 3.5, 4.1, 4.2.1, 4.4,
0.31	30/04/2025	SIE	Contributions to Sections 3.3, 3.4
0.35	04/06/2025	SURREY	Contributions to Sections 4.5, 5.3.9, 5.3.10
0.36	04/06/2025	EGM	Contributions to Section 5.4
0.37	04/06/2025	NUID UCD	Contributions to Sections 3.7.3, 3.8.2, 3.8.3, 5.2.3, 5.3.4, 5.3.6, 5.3.7
0.4	17/06/2025	ATOS	Contributions to Sections 4.3.1, 4.3.2,
0.5	23/06/2025	UC	Contributions to Sections 3.7.1, 5.3.5
0.51	24/04/2025	NUID UCD	Contributions to Sections 3.5, 3.7.1, 4.3.5,
0.52	24/06/2025	EGM	Contributions to Sections 3.7.2, 5.4,
0.6	25/06/2025	INRIA	Contributions to Sections 3.9, 3.10, 5.3.11,
0.7	02/07/2025	NUID UCD	Formatting, Introduction, Conclusions, Executive summary
0.8	08/07/2025	NUID UCD	Formatting and draft for internal review

Document History			
Version	Date	Change editors	Change
0.9	22/07/2025	NUID UCD	Addressing the comments of the reviewers throughout the document.
0.91	23/07/2025	SURREY	Addressing the comments of the reviewers in 5.3.10
0.92	24/07/2025	ATOS	Addressing the comments of the reviewers in 5.2.4
0.93	24/07/2025	SIE	Addressing the comments of the reviewers in 3.4
0.94	24/07/2025	EGM	Addressing the comments of the reviewers in 4.6.
0.95	24/07/2025	INRIA	Addressing the comments of the reviewers in 5.3.11.
0.96	26/07/2025	UC	Addressing the comments of the reviewers in 5.3.5
0.97	28/07/2025	NUID UCD	Merging contributions and preparing the version for quality review.
0.98	29/07/2025	NUID UCD	Addressing comments of the quality review
0.99	30/07/2025	ATOS	Quality Format Review
1.0	31/07/2025	ATOS	FINAL VERSION TO BE SUBMITTED

Quality Control		
Role	Who (Partner short name)	Approval date
Reviewer 1	Gabriel Danciu (SIE)	21/07/2025
Reviewer 2	Alberto Carelli (LINKS)	22/07/2025
Quality manager	María Guadalupe Rodríguez (ATOS)	30/07/2025
Project Coordinator	Miguel Angel Esbrí (ATOS)	31/07/2025

Table of Contents

List of Figures

List of Tables

List of Acronyms

Abbreviation / acronym	Description
ABOD	Angle Based Outlier Detection
AI	Artificial Intelligence
API	Application Programming Interface
ARIMA	Autoregressive Integrated moving averaged
AUC	Area Under Curve
AUCP	AUC Percentage
DAG	Directed Acyclic Graph
DEA	Denoising Extraction Autoencoder
DPO	Data Processing Orchestrator
DQ	Data Quality
EMA	Exponential Moving Average
ESD	Extreme Studentized Deviation
EU	European Union
FFT	Fast Fourier Transform
FL	Federated Learning
GAN	Generative Adversarial Networks
GESD	Generalised ESD
GMM	Gaussian Mixture Models
GRU	Gates Recurrent Unit
GUI	Graphical User Interface
HBOS	Histogram Based Outlier Selection
HST	Half Space Trees
HTTP	Hypertext Transfer Protocol
ICA	Independent Component Analysis
IID	Independent and Identically Distributed

Abbreviation / acronym	Description
IoT	Internet of Things
IR	Imbalance Ratio
KF	Kalman Filter
KNN	K-Nearest Neighbours
LD	Linked Data
LMS	Least Mean Squares
LOF	Local Outlier Factor
LRID	Likelihood Ratio Imbalance Degree
LSTM	Long Short-Term Memory
LSTMOD	Long Short-Term Memory Outlier Detection
MCD	Minimum Covariance Distance
MCU	Microcontroller Unit
ML	Machine Learning
MNIST	Modified National Institute of Standards and Technology database
NGSI-LD	Next Generation Service Interfaces - Linked Data
NLMS	Normalised LMS
NN	Neural Network
OCSVM	One-Class Support Vector Machine
OD	Outlier Detection
PCA	Principal Component Analysis
PPS	Predictive Power Score
QSGD	Quantized Stochastic Gradient Descent
RLS	Recursive Least Squares
ROC	Receiver Operating Characteristic
SARIMA	Seasonal Autoregressive Integrated moving averaged
SCL	Strongly coordinated learning

Abbreviation / acronym	Description
SGD	Stochastic Gradient Descent
SMA	Simple Moving Average
SMOTE	Synthetic Minority Over-sampling Technique
SNE	Stochastic Neighbour Embedding
SNR	Signal to Noise Ratio
SSA	Singular Spectrum Analysis
STL	Seasonal and Trend decomposition using Loess
SVD	Singular Vector Decomposition
SVDD	Support Vector Data Description
SVM	Support Vector Machine
UI	User Interface
URL	Uniform Resource Locator
VIF	Variance Inflation Factor
WCL	Weekly Coordinated Learning
WMA	Weighted Moving Averages

Executive Summary

Data quality is of the highest importance for companies to improve their decision-making systems and the efficiency of their products. In this current data-driven era, it is important to understand the effect that “dirty” or low-quality data can have on a business. Manual data cleaning is the common way to process data, accounting for more than 50% of the time of knowledge workers. SEDIMARK acknowledges the importance of data quality for both sharing and using data to extract knowledge and information for decision-making processes. Thus, one of the main goals of SEDIMARK is to develop a data processing pipeline that assesses and improves the quality of data generated and shared by the SEDIMARK data providers.

This deliverable presents the final version of the methods and techniques developed within SEDIMARK for processing data and improving their quality, extending the first version which was delivered in SEDIMARK Deliverable D3.1 [74]. The focus in this deliverable is to present the final version of the key techniques that are used for quality improvement of datasets, based on the requirements of the SEDIMARK platform so that they all work together smoothly.

SEDIMARK considers two main types of data generated and shared within the marketplace: (i) static/offline datasets and (ii) dynamic/streaming datasets. The project acknowledges that it is important to cater to both types of datasets equally, thus in most scenarios, separate and customised versions of the tools have been developed for static and streaming datasets. Techniques for outlier detection, noise removal, deduplication and imputation of missing values are important for improving the quality of datasets. These techniques aim to remove abnormal values or noise from the dataset, remove duplicate values or fill out gaps in some entries or add complete entries. Techniques for feature engineering such as feature extraction and selection have also been developed to enrich the datasets. Synthetic dataset creation is important in scenarios where data providers don’t want to share their real datasets (i.e. for privacy reasons) but want to share synthetic versions that mimic the real ones.

This deliverable also presents the framework to orchestrate the whole functionality of the data processing pipeline using a Data Processing Orchestration. This component enables end users to interact with the built-in data processing solutions through a simplified dashboard interface. This deliverable also presents the final version of the key quality metrics that SEDIMARK has defined for assessing the quality of datasets, both per data point and as a whole, as well as the key techniques for dataset quality improvement, designed to meet SEDIMARK platform requirements and ensure seamless integration.

Another important part is the description of techniques towards reducing the energy consumption of the components of the data processing pipeline and optimizing data efficiency, i.e. using techniques for data distillation, coreset selection and dimension reduction. Minimising the communication cost in distributed machine learning scenarios is also important for SEDIMARK, because communication can increase energy consumption. Techniques to optimise the Artificial Intelligence (AI) models both during training and inference are also presented, focusing on quantisation, pruning, low rank factorisation and knowledge distillation.

Finally, considering that minimising energy consumption can influence performance or communication, the deliverable presents the final analysis on these trade-offs, aiming to provide insights to data providers on how to better configure the pipeline or what models they should select in order to achieve their targets (energy efficiency/performance/communication).

1 Introduction

1.1 Purpose of the document

This document is the final deliverable from WP3 Tasks 3.1 and 3.5, aiming to provide the final draft of the SEDIMARK data quality pipeline. This document details how the pipeline was built to improve the quality of datasets shared through the marketplace while also addressing the problem of energy efficiency in the data value chain. This document builds on the technologies described in SEDIMARK Deliverable D3.1 showing the final ideas and implementations of the respective tools and techniques for improving data quality and energy efficiency.

The main goal of this document is to discuss how data quality is seen in SEDIMARK, what are the metrics defined in order to assess the quality of data that are generated by data providers, and what techniques are provided to them for improving the quality of their data before sharing them on the data marketplace. This helps the data providers to both optimise their decision-making systems for the Machine Learning (ML) models they train using their datasets, and to increase their revenues by selling datasets of higher quality and thus higher value. Regarding the first argument, it is well documented that low-quality data has a significant impact on business, with reports showing a yearly cost of around $3 Trillion, and that knowledge workers waste 50% of their time searching for and correcting dirty data [1]. It is evident that data providers will hugely benefit from automated tools to help them improve their data quality, either without any human involvement or with minimum human intervention and configuration.

This document presents high-level descriptions of the concepts and tools developed for the data quality pipeline and the energy efficiency methods for reducing its environmental cost, as well as concrete technical details about the implementation of those tools. Thus, it can be considered that this is both a high-level and a technical document, thus targeting a wide audience. Primarily, the document targets the SEDIMARK consortium, discussing the technical implementations, so that the integration activities of the project can easily integrate of all the components into a single SEDIMARK platform. Apart from that, this document also targets the scientific and research community, since it presents new ideas about data quality and how the developed tools can help researchers and scientists improve the quality of the data they use in their research or applications. Similarly, the industrial community can leverage the project tools to improve the quality of their datasets or also assess how they can exploit the results about energy efficiency to reduce the energy consumption of their data processing pipelines. Moreover, EU initiatives and other research projects should consider the contents of the deliverable in order to derive common concepts about data quality and reducing energy consumption in data pipelines.

1.2 Relation to another project work

This deliverable is an extension of SEDIMARK Deliverable D3.1 and presents the results of work done in the last 19 months of the project in Task 3.1 (AI based tools for data quality management) and Task 3.5 (energy optimisation of data management techniques). Additionally, this deliverable is also based on the work done in WP2, especially in Task 2.2 related to the definition of the functional and non-functional requirements for the areas of interest of this deliverable (presented in deliverable SEDIMARK_D2.1 [2]), in Task 2.3 regarding the system functional architecture and the functional components related with data quality and energy efficiency, as well as in Task 2.4 regarding the initial draft of the interfaces between the system components (presented in SEDIMARK_D2.2 [3] and SEDIMARK_D2.3 [4]). The architecture and interfaces are of major importance for the work presented in this deliverable, as they lay the foundations for the work in the tasks that provide their output to this deliverable. The output of the work presented in this deliverable will also be used for the overall system testing and validation (in WP5). Figure 1 shows the interaction of the activities within WP3 and the relation with the rest of the work packages.

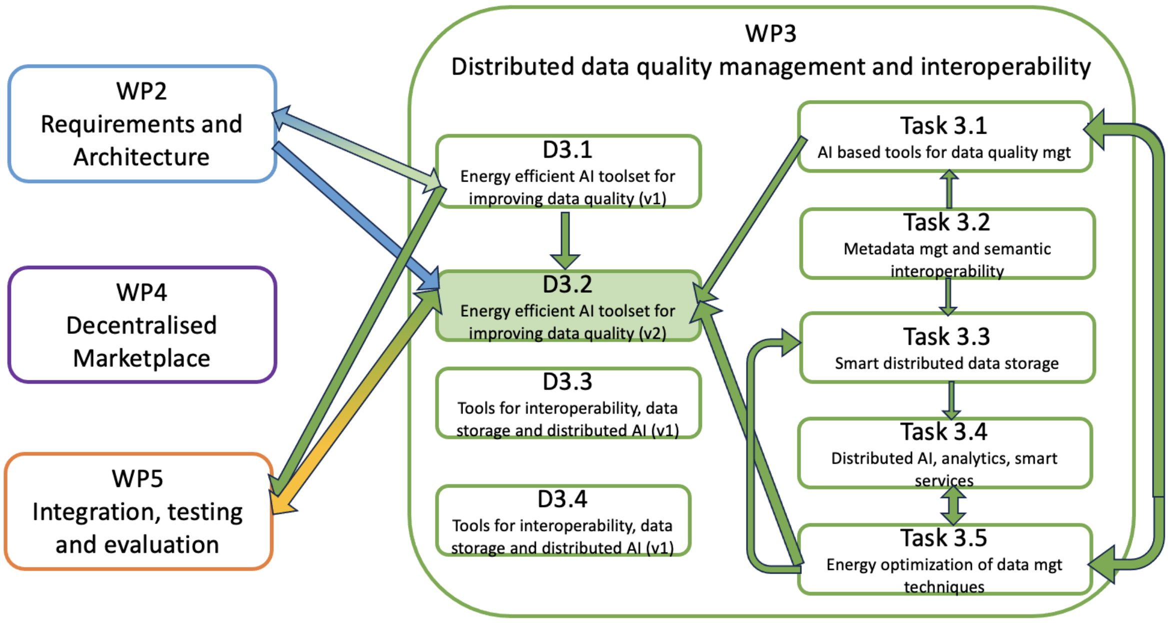

WP3
Distributed data quality management and interoperability
WP2
Requirements and Architecture
WP4
Decentralised Marketplace
WP5
Integration, testing and evaluation
D3.1
Energy efficient AI toolset for improving data quality (v1)
D3.2
Energy efficient AI toolset for improving data quality (v2)
D3.3
Tools for interoperability, data storage and distributed AI (v1)
D3.4
Tools for interoperability, data storage and distributed AI (v1)
Task 3.1
AI based tools for data quality mgt
Task 3.2
Metadata mgt and semantic interoperability
Task 3.3
Smart distributed data storage
Task 3.4
Distributed AI, analytics, smart services
Task 3.5
Energy optimization of data mgt techniques

Figure 1: Relationship between SEDIMARK_D3.1 and other deliverables, tasks, and workpackages.

1.3 Structure of the document

This document is structured into 6 major chapters:

Chapter 1 is the current chapter and presents the introduction to the overall document.

Chapter 2 presents the positioning of the deliverable within the overall project.

Chapter 3 presents the main work done within Task 3.1 and is related to tools and techniques for assessing and improving the quality of data assets.

Chapter 4 presents work done within Task 3.5 and is related to the design and development of techniques to reduce energy consumption and improve energy efficiency of tools used within the data quality pipeline or the AI pipeline of SEDIMARK.

Chapter 5 presents work done within Task 3.5 and is related to the assessment of trade-offs between energy consumption, ML model accuracy and efficiency, and communication cost. The included results aim to help data providers and researchers understand the gains/losses when choosing to optimise for energy consumption or accuracy.

Chapter 6 presents the conclusions of the document, discussing the major outcomes and conclusions of the work.

1.4 Glossary adopted in this document

The readers are referred to SEDIMARK_D2.3 which provides a complete final list of the terminology used within SEDIMARK.

2 Position within the project

This deliverable presents the final draft of the work in Tasks 3.1 and 3.5. The work in these tasks concerns two of the main pillars of the SEDIMARK project, since these tasks focus on developing tools that will enhance the quality of the datasets to be shared within the project, whilst also ensuring that the SEDIMARK tools will be energy efficient, giving also options to data providers to understand how they can tune the models and the tools in order to optimise either for energy efficiency or for accuracy and performance.

Figure 2 presents the SEDIMARK functional architecture that was described in deliverable SEDIMARK_D2.2 in detail. The functional components that are part of this deliverable are highlighted in orange. As is evident, these are part of the Data and Intelligence layers of SEDIMARK. More details about these components and their mapping to deliverable sections are given below:

- **Data Processing Orchestration:** this is described in section 3.3, where the framework for orchestrating the whole data quality pipeline is presented.
- **Pipeline Orchestrator UI**: this is described in section 3.4, where the dashboard that will be provided to the users for managing their datasets and accessing the various data processing tools is presented.
- **Data Visualisation**: this is described in section 3.4, where the roadmap for the design of his module and its integration with Data Processing Orchestration and Pipeline Orchestrator UI is described.
- **Data Quality Evaluation**: this component handles the assessment of the data quality and is described in section 3.5, including the metrics that were defined within SEDIMARK for assessing the quality of datasets.
- **Data Profiling**: this is described in section 3.6 and provides general information and statistics about a dataset that is managed by the data processing pipeline.
- **Data Curation:** this is a suite of components that process the datasets and aim to improve their quality. It is described in section 3.7, where more details about the mechanisms for outlier detection, noise removal, deduplication, missing value imputation, etc. are provided.
- **Data Augmentation**: this is described in section 3.8, discussing what tools SEDIMARK provides currently for generating synthetic data and how these can be used for removing bias and improving fairness in datasets.
- **Feature Engineering:** this is described in section 3.9, where the methods for extracting valuable features from datasets are discussed, aiming to improve the usefulness of datasets when used for training AI models.
- **Frugal AI**: this component is described in sections 4.2 and 4.3, discussing various techniques to reduce energy consumption of ML models both during training.
- **Model Optimisation:** this component is described in section 4.3, discussing techniques to optimise the ML models after they are trained, aiming to reduce energy consumption.
- **Energy Efficiency:** this component is actually a suite of techniques that help improve the energy efficiency of the tools and models of the SEDIMARK toolbox. The components and methods included in this suite are described in Section 4, while in Section 5 there is

an initial assessment of the trade-offs between energy efficiency and performance providing insights on how the methods can be tuned for either criterion. Some specific components and analysis that are focused on energy efficiency are given in sections 4.4, 4.5, and 4.6.

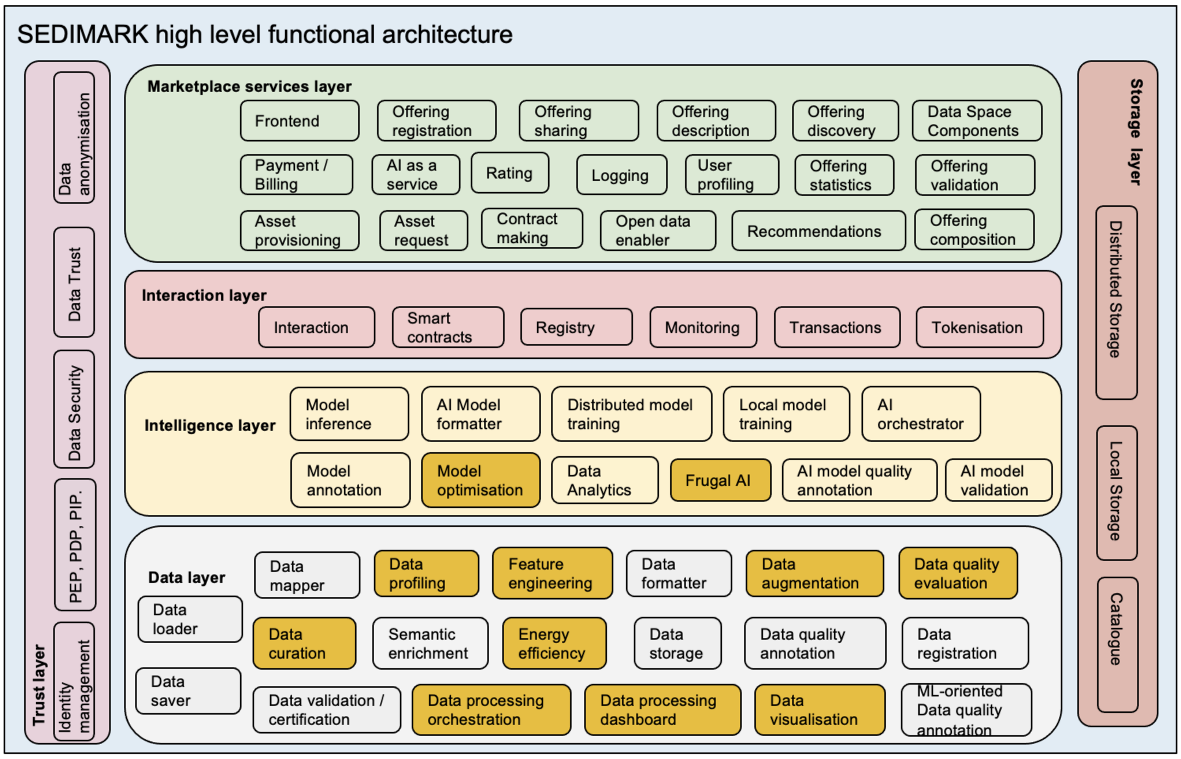


Figure 2: Mapping of SEDIMARK_D3.2 components to the SEDIMARK architecture.

As discussed in the introduction section, this is the final version of the deliverable describing the methods and tools developed within SEDIMARK related to data processing and energy efficiency. As presented in deliverable SEDIMARK_D2.3 [4], these tools are part of the Data Processing Pipeline, which is part of the SEDIMARK toolbox. The developed tools will become public at the end of the project as part of the SEDIMARK's GitHub page [73], considering that most of them will be open sourced. In detail, currently there are implementations of:

- **Data Processing Orchestration, Dashboard** and **Visualisation**, implemented as wrapper and abstraction layer of the Python based Mage.ai.
- Data Profiling and Data Quality Evaluation, implemented using Python.
- **Outlier detection**, implemented using python and building on libraries such as PyOD, tods, pythresh, pandas, scikit-learn, and river.
- **Deduplication**, implemented using python and building on libraries such as recordlinkage and dedupe.
- **Missing Value Imputation**, implemented using python and building on libraries such as hyperimpute, river and scikit-learn.
- **Feature Engineering**, implemented using python and building on libraries such as pandas, scipy, pymmh3 and numpy,

- **Data augmentation**, implemented using python and libraries such as ydata-synthetic.
- **Model Optimisation**, implemented using python and building on techniques for quantisation.
- **Frugal AI**, not implemented as a standalone component, but currently being an analysis of techniques that can be used to maximise the energy efficiency of AI model training and inference.

The table below shows the links for accessing the implemented modules described in this deliverable:

Table 1: GitHub links for the implemented modules

Module	Section	GitHub Link
Data Processing Orchestrator	3.3	/Sedimark/MageAPI, /Sedimark/mage
Data Processing Dashboard	3.4	/Sedimark/Sedimark-Orchestration-UI
Data Visualisation	3.4	/Sedimark/Sedimark-Orchestration-UI
Data Profiling	3.6	/Sedimark/sedimark_dqp
Data quality evaluation	3.6	/Sedimark/sedimark_dqp
		/Sedimark/UC_modules
Outlier detection	3.7.1.1	/Sedimark/sedimark_dqp
	3.7.1.2	/Sedimark/UC_modules
Noise cancellation	3.7.2	/Sedimark/noise_cancellation
Deduplication	3.7.3	/Sedimark/sedimark_dqp
Missing Value Imputation	3.7.4	/Sedimark/sedimark_dqp
Feature Engineering	3.9	/Sedimark/SEDIMARK_DPP/tree/main/feature_engineering
Data augmentation	3.8	/Sedimark/sedimark_dqp
Quantisation (centralised)	4.3.2	/Sedimark/Crossformer
Pruning	4.3.3	/Sedimark/Prune
Knowledge distillation	4.3.4	/Sedimark/offering-generator
Low-rank factorisation	4.3.5	/Sedimark/Recommender
Reducing energy in data storage	4.5	/Sedimark/broker

3 Data Processing Pipeline

3.1 Overview

A key objective in the development of SEDIMARK is to promote the sharing of high quality, curated data within its marketplace. The huge quantities of data made available by the spread of commercial internet technologies are widely acknowledged to be at the root of the explosion of interest in machine learning. At the same time, effectively dealing with this data presents a huge challenge, especially when much of it is of low or only medium quality. A growing trend in both industry and research has sought to give data collection and curation a central role within the development of AI systems, with many championing Data-centric rather than Model-centric AI [11]. This originates not only with observations that improving raw data quality can often be more effective than iterating on model architectures but also that the data itself can often be the root cause of many ethical concerns in the development of AI models [10]. As such, emerging approaches to ML and data science support putting the curation of high-quality datasets first and foremost within the data science workflow. Although a vast number of tools and methodologies exist for improving data quality, the notion of data quality is itself quite qualitative, and domain dependent, and often little can be done without the involvement of a large amount of expert human labour. In the work of data scientists alone, tasks such as outlier detection (OD), data deduplication and missing value imputation are widely acknowledged to take up to 50% of their time [1],[7],[8].

A key aim of SEDIMARK is to minimise the human effort needed to improve the quality of data within its marketplace, thus increasing the data's intrinsic value, and attracting a wider base of both data consumers and providers. As such, it includes a wide array of data cleaning tools that can be integrated as part of the data publishing workflow. In its current iteration, SEDIMARK includes tools for outlier detection, deduplication, missing value imputation, and noise removal from time series. SEDIMARK however recognises that the use of these tools could effectively prove a barrier to adoption by data providers, if they require excessive human effort to adapt them to new datasets. As such, a key challenge in the development of SEDIMARK concerns reducing the amount of human involvement required by the data cleaning process. As such, SEDIMARK has explored applying Automated Machine Learning (AutoML) techniques to the individual components of the pipeline. In two of the key data cleaning components, there are emerging technologies for automating the process of model selection and repair, which are discussed below. In addition, some AutoML or meta-learning techniques could in principle be used as a method for optimising the overall data pipeline as a whole, though this may not be feasible within the scope of the project. This is discussed in section 3.10.

Figure 3 shows the overview of the data processing pipeline within SEDIMARK and the interactions between the different components. As is shown, the user interacts with the pipeline through the Data Processing Dashboard, choosing the data to be analysed/curated, the processing steps and the functions that will be run, as well as the configuration of the various functions, i.e. with respect to the models that will run and the hyperparameters for each model. Then, the Data Processing Dashboard forwards all this information to the Data Processing Orchestration component, which manages the operation of the whole pipeline. The orchestration component takes the configuration file (set by the user) as an input and creates the flow of modules that need to be executed (sequentially or in parallel) in order to process the data, i.e. to format the data, to do the profiling and extract statistics, to curate and assess the quality of the data, to validate the data, extract features and annotate them to convert them to the external data format. More details about these steps (apart from data validation and annotation which are part of SEDIMARK_D3.3 [5]) are given in the below subsections.

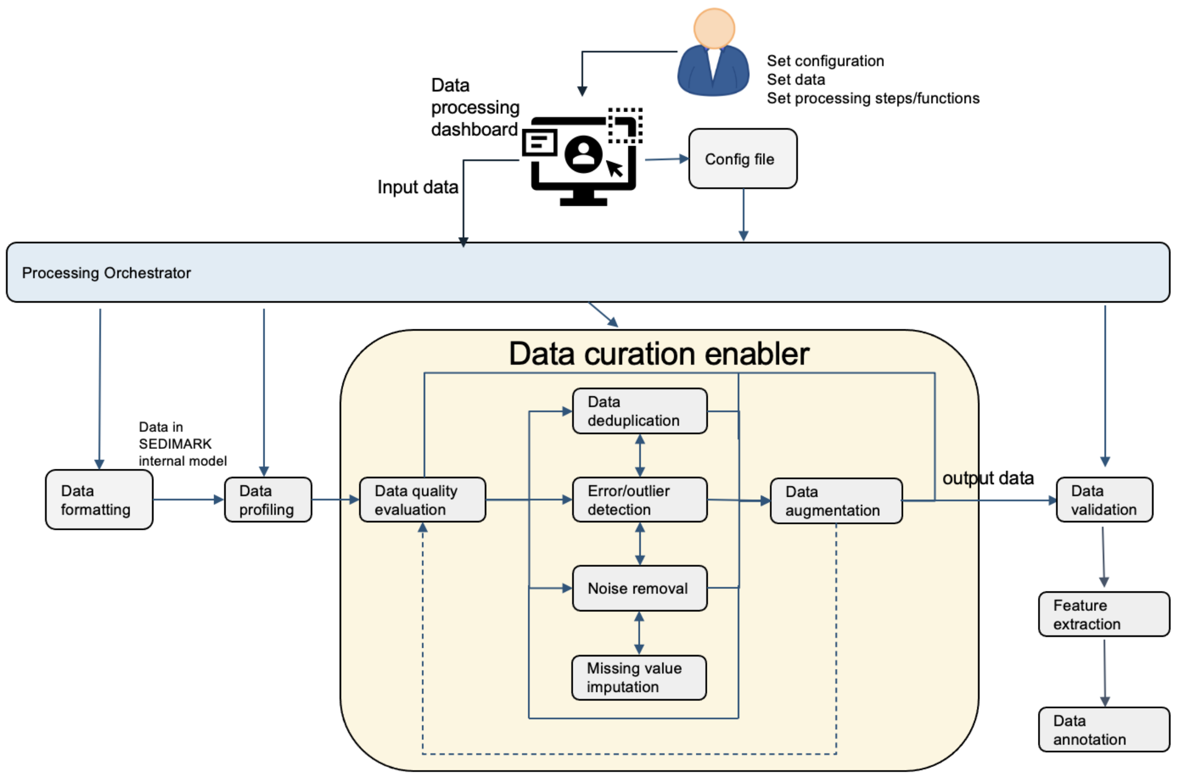


Figure 3: Overview of the data processing pipeline of SEDIMARK

The rest of Section 3 is structured mostly according to the order of the steps of the data processing pipeline, shown in Figure 3: section 3.2 discusses in general how SEDIMARK deals with both static and streaming datasets; sections 3.3 and 3.4 present the overlay data processing orchestrator and the dashboard; section 3.5 is an entry section discussing the metrics defined within SEDIMARK for evaluating the data quality; section 3.6 presents the tool developed for data profiling and data quality evaluation; section 3.7 presents the data curation techniques; section 3.8 presents the data augmentation techniques; section 3.9 presents the techniques for extracting meaningful features from the datasets; section 3.10 presents a high level view on how SEDIMARK wants to exploit automatic ML (AutoML) techniques for the data processing pipeline.

3.2 Offline dataset processing vs data streaming

Data provisioned by providers or artificially generated within SEDIMARK could be either *static* (dataset) or *dynamic* (data stream). A dataset is defined as a finite set of data instances that can be stored in memory and analysed while accessing the data several times. Static datasets are typically well-defined and can be accessed, queried, and analysed as many times as needed. Offline processing involves working with static datasets that have already been collected and stored. These datasets typically do not change during the analysis. They are ideal for scenarios where one does not need real-time insights. Examples include historical data analysis, comprehensive reports, and machine learning model training.

On the other hand, a data stream is defined as an unbounded sequence of multidimensional, sporadic, and transient instances that are generated over time. Data streams are characterised by their dynamic and evolving nature. They are crucial for scenarios requiring immediate action or analysis, such as fraud detection, real-time analytics, and monitoring systems that treat critical tasks.

So, one major difference between these data types is that a dataset can be stored in its entirety in memory to serve for analytics at any time while for a data stream, data points arrive one by one incrementally and are not available at once which requires adapted analytics methods for such a setting.

If the data are generated as a stream, they require immediate action, while for batched, historical, or less time-sensitive data, offline processing is more suitable. For businesses that require real-time insights (e.g., stock trading platforms), streaming is essential. In contrast, businesses focusing on mid and long-term strategies might rely more on offline processing. From the technical point of view, data streaming demands a more sophisticated infrastructure and real-time data processing capabilities, whereas offline processing can be more manageable with conventional data analysis tools and a more affordable budget.

For batch dataset processing, traditional machine learning algorithms are needed, and several ones have been proposed in the state-of-the-art that can be easily added to the SEDIMARK AI pipeline. On the other hand, the infinite nature of data streams presents certain technical and practical constraints that render conventional static algorithms ineffective due to, among other things, their high consumption of resources, including time and memory. Therefore, to process data streams, data stream mining algorithms adapted to handle such data generated in real time need to be used.

Processing data streams presents some specific challenges that are presented as follows:

- **Single-pass**: Unlike processing static datasets, it is no longer possible to analyse data using several passes during the course of computation because of its unbounded nature. Taking into account this constraint, algorithms work by processing each instance from the stream only once (or a couple of times) and use it to update the model – or the statistical information about data – incrementally (instance-incremental algorithms). In the case of batch-incremental algorithms, a batch (chunk or window) of instances is processed at once instead of only one instance.
- **Running time**: An online algorithm must process the incoming data points as rapidly as possible. Otherwise, the algorithm will not be efficient enough for applications where rapid processing is required.
- **Memory usage**: Because of the massive amounts of data streams that require a limitless memory to be processed and stored, it is difficult and sometimes even impossible to store the entire stream in memory. So, any stream algorithm must be able to operate under restricted memory constraints by storing a few synopses of the processed data and the current model(s).
- **High dimensionality**: In some scenarios, streaming data may be high-dimensional, for example, text documents, where distances between instances grow exponentially due to

the curse of dimensionality. The latter can potentially impact any algorithm's performance, mainly, in terms of time and memory.

- **Concept drift**: Concept drift is a common challenge in data streams, as the data distribution can change over time. Data stream mining algorithms must adapt to these changes dynamically.

The aforementioned challenges frequently arise in different data stream mining tasks. In this context, incremental data stream approaches have been developed and will be extended and used within SEDIMARK to address these requirements. To cope with the single-pass requirement, incremental data stream mining algorithms that potentially handle concept drift by being coupled with drift detection mechanisms need to be used. The SEDIMARK AI pipeline features a dimension reduction component as a pre-processing step which will address the curse of dimensionality and thus contribute to the reduction of the resource usage (e.g., memory and time) of the adopted ML algorithms.

SEDIMARK focuses on ensuring high-quality data through tools for cleaning and curation, catering to both offline and streaming datasets. Considering the requirements defined in SEDIMARK_D2.1, SEDIMARK needs to identify and manage problematic records, balancing data integrity with its size. The SEDIMARK data processing pipeline has the flexibility to process both static and real-time streaming data with attention to latency and efficiency. For that, a modular and user-customizable data curation pipeline has been developed, balancing expert needs with simplicity for non-technical users. More information is given in the sections below.

3.3 Data Processing Orchestration

This section outlines a multi-stage data processing architecture, namely the Data Processing Orchestration (DPO) designed to manage the flow, transformation and storage of data across various system components.

3.3.1 DPO Technical Description

The DPO process starts with the Data Provider, representing users who will utilize the SEDIMARK Toolbox to produce assets, ranging from datasets to models, and publish them on the SEDIMARK marketplace for other users to discover and use.

To better align data processing pipelines with the specific needs of the provider, pipelines are defined and executed through the Orchestrator UI, described in detail in section 3.4. Once defined, pipelines can be triggered using multiple methods: manually by the user or automatically at specified time intervals. This flexibility is particularly useful when new data entries are created periodically and require immediate processing.

The DPO execution process preparation steps for ensuring a smooth and efficient data processing orchestration are detailed as follows:

- **Pipeline definition:** The data provider defines a data processing pipeline using the Orchestrator UI or programmatically through the API. Once configured, the pipeline is submitted to the orchestrator engine.

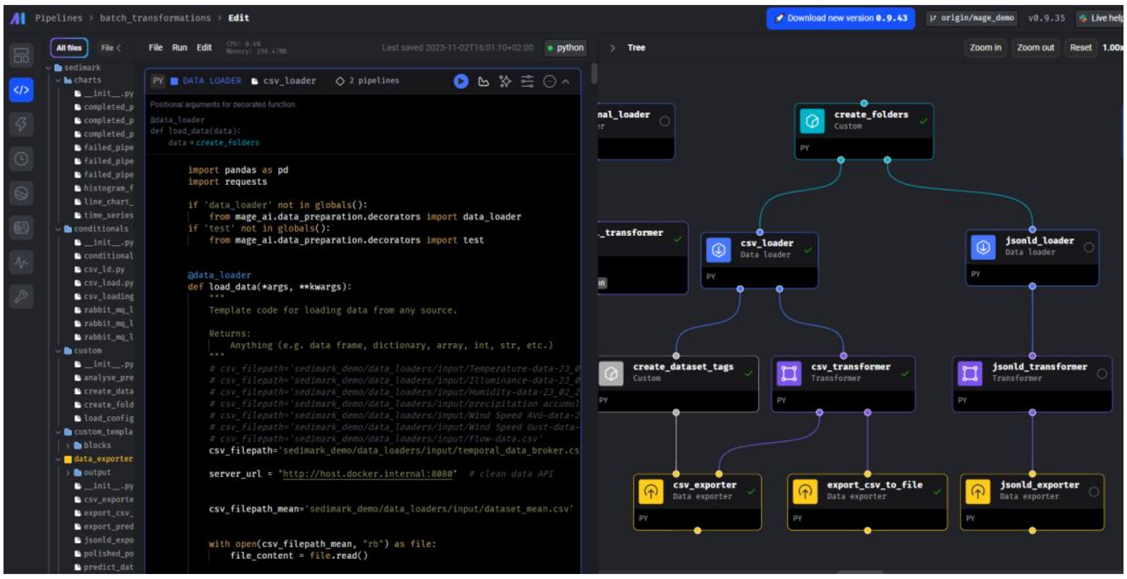
- **Pipeline triggering:** The orchestrator triggers the pipeline based on the defined method, which can be instant, periodic, or event-driven.

- **Data retrieval:** Typically, the first step of a pipeline involves data retrieval through a preconfigured data source defined during the pipeline definition step.
- **Initial Data Processing:** The retrieved data is extracted from its original format and a preprocessing step is applied over the raw dataset. This step can produce intermediate artefacts required by subsequent processes.
- **Pipeline Execution:** The pipeline execution proceeds with the application of data processing techniques if they are described in the pipeline definition.
- **Output delivery:** The resulting data asset is pushed to the NGSI-LD Broker and the Offering Sharing component, enabling its exchange and availability to Data Consumers.

While the steps described previously primarily target streaming datasets, a similar process can be adapted for static datasets as well.

As previously outlined, the DPO execution process follows a well-defined sequence, ranging from pipeline definition and triggering to data retrieval, processing, and final output delivery. At the core of the DPO framework is Mage AI [52] which plays a major role in supporting this orchestration. Mage AI is a robust platform that streamlines data preparation and transformation workflows. It automates essential tasks such as data cleaning, transformation and feature engineering using machine learning techniques. Additionally, Mage AI offers a user-friendly interface that enables both technical and non-technical users by reducing code complexity and simplifying data handling operations. Furthermore, it provides flexible integration capabilities with various data sources and machine learning tools. This flexibility is further reflected in its modular approach to data processing, which divides tasks into distinct, customizable modules, enhancing both the efficiency and adaptability of the data preparation process.

An example of such a processing pipeline visualized in the Mage.ai UI is presented in Figure 4. The pipeline facilitates the loading, processing, and prediction of SEDIMARK data, using different loaders based on the format of the input (raw) data and converting them to the internal SEDIMARK format. Moreover, Figure 4 displays two types of data loaders: one using CSV and the other using NGSI-LD.



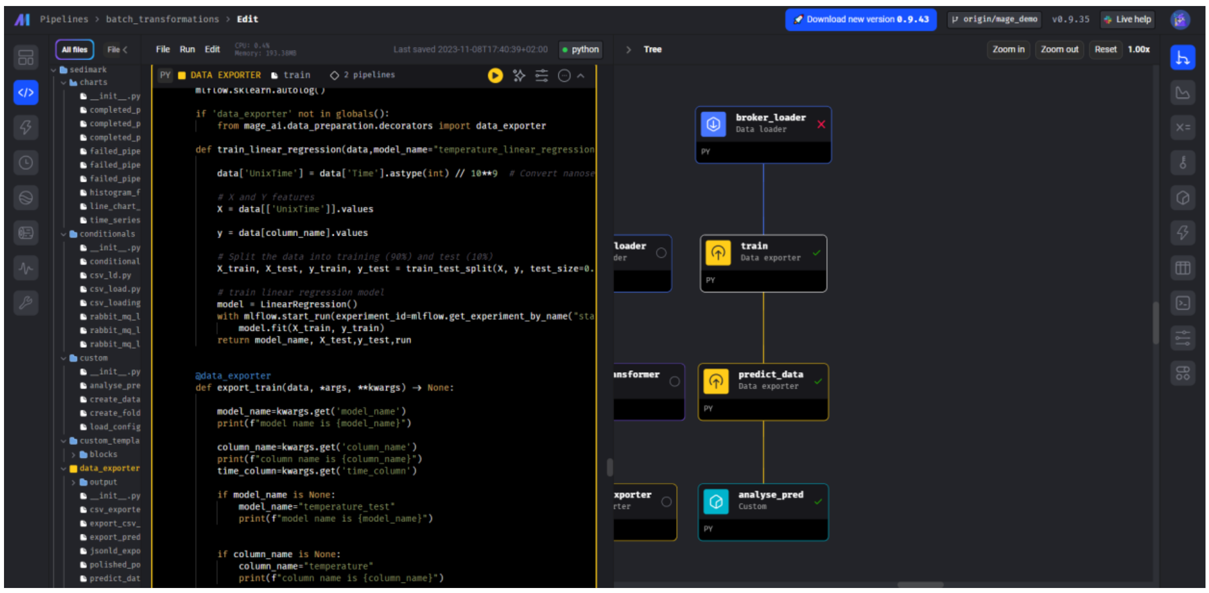
```
Positional arguments for decorated function:
@data_loader
def load_data(data):
    data ← create_folders

    import pandas as pd
    import requests

    if 'data_loader' not in globals():
        from mage_ai.data_preparation.decorators import data_loader
    if 'test' not in globals():
        from mage_ai.data_preparation.decorators import test


    @data_loader
    def load_data(*args, **kwargs):
        """
        Template code for loading data from any source.

        Returns:
            Anything (e.g. data frame, dictionary, array, int, str, etc.)
        """
        # csv_filepath='sedimark_demo/data_loaders/input/Temperature-data-23_0
        # csv_filepath='sedimark_demo/data_loaders/input/Illuminance-data-23_0
        # csv_filepath='sedimark_demo/data_loaders/input/Humidity-data-23_02_2
        # csv_filepath='sedimark_demo/data_loaders/input/precipitation accumul
        # csv_filepath='sedimark_demo/data_loaders/input/Wind Speed AVG-data-2
        # csv_filepath='sedimark_demo/data_loaders/input/Wind Speed Gust-data-
        # csv_filepath='sedimark_demo/data_loaders/input/Flow-data.csv'
        csv_filepath='sedimark_demo/data_loaders/input/temporal_data_broker.cs

        server_url = "http://host.docker.internal:8080"  # clean data API


        csv_filepath_mean='sedimark_demo/data_loaders/input/dataset_mean.csv'


        with open(csv_filepath_mean, "rb") as file:
            file_content = file.read()
```

Figure 4: Mage AI interface with sample demo flow for processing SEDIMARK data.

To ensure a common data format within the marketplace and fulfil interoperability requirements, the NGSI-LD context broker serves as the primary method for data export. It saves data with contextual metadata specific to the marketplace, thus simplifying data sharing between users.

Another view of the Mage AI UI illustrates how a simple flow for dataset analysis can be configured, leveraging Mage AI's capabilities for streamlined data handling (Figure 5). Starting with the data loading phase, the pipeline retrieves the data (i.e., weather information such as temperature forecasts) from an NGSI-LD Context Broker, specifically the Stellio context broker [12]. This data is then processed and formatted, preparing it for the subsequent stages of the pipeline, which can include data prediction or any of the modules in the data processing pipeline, such as anomaly detection, deduplication, or value imputation. Mage.ai uses a unified method to launch all underlying components.

Finally, the pipeline concludes with the data export stage, where the results are integrated back into the original dataset (depending on user preferences). There is also the option to export this enriched dataset back to the NGSI-LD context broker, closing the loop in the data processing cycle.


Pipelines > batch_transformations > Edit
File Run Edit
Last saved 2023-11-08T17:40:39+02:00
python
DATA EXPORTER train 2 pipelines
Tree
Zoom in Zoom out Reset 1.00x
broker_loader
Data loader
train
Data exporter
predict_data
Data exporter
analyse_pred
Custom

```
    mlflow.sklearn.autolog()

    if 'data_exporter' not in globals():
        from mage_ai.data_preparation.decorators import data_exporter

    def train_linear_regression(data,model_name="temperature_linear_regression

        data['UnixTime'] = data['Time'].astype(int) // 10**9  # Convert nanose

        # X and Y features
        X = data[['UnixTime']].values

        y = data[column_name].values

        # Split the data into training (90%) and test (10%)
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.

        # train linear regression model
        model = LinearRegression()
        with mlflow.start_run(experiment_id=mlflow.get_experiment_by_name("sta
            model.fit(X_train, y_train)
        return model_name, X_test,y_test,run


    @data_exporter
    def export_train(data, *args, **kwargs) -> None:

        model_name=kwargs.get('model_name')
        print(f"model name is {model_name}")

        column_name=kwargs.get('column_name')
        print(f"column name is {column_name}")
        time_column=kwargs.get('time_column')

        if model_name is None:
            model_name="temperature_test"
            print(f"model name is {model_name}")


        if column_name is None:
            column_name="temperature"
            print(f"column name is {column_name}")
```

Figure 5: Mage AI orchestration for a simple training flow using SEDIMARK data.

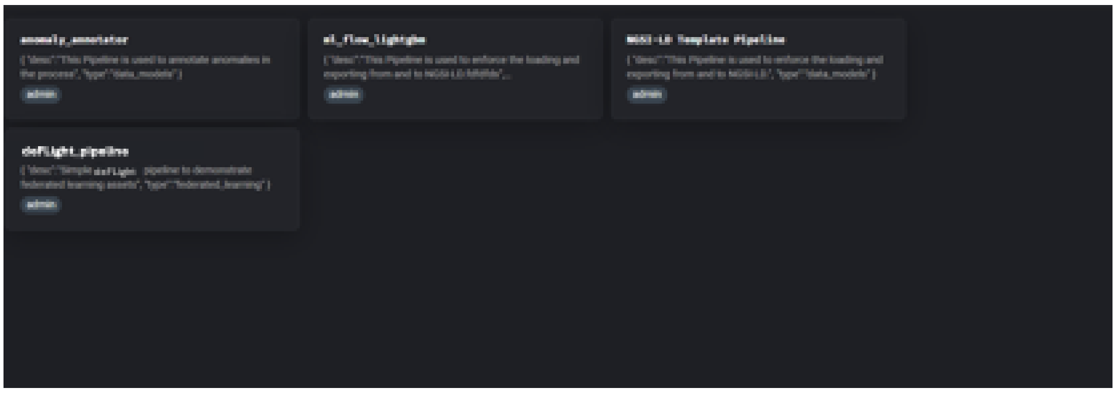
3.3.2 Mage AI Pipeline Templates

To streamline development and promote reuse across different scenarios, dedicated pipeline templates have been developed for each project, pilot, and use case. These predefined pipelines form the foundation of the SEDIMARK Toolbox templates, offering users a set of ready-made configurations to kickstart their own data processing workflows.

By providing consistent structures and reducing the need for repetitive setup, these templates help accelerate deployment while ensuring alignment with common data standards. Figure 6 showcases a selection of these templates, which are further described as follows:

- **anomaly_annotator:** An anomaly detection pipeline template that retrieves dataset assets from the SEDIMARK Marketplace via the NGSI-LD Context Broker and applies a custom-built AnomalyDetectionModule to identify and annotate anomalous data points. This pipeline template was validated in the EGM pilot's weather dataset use case.

- **ml_flow_lightgbm:** A machine learning pipeline template that integrates LightGBM with MLflow for model training, storage, and inference. LightGBM is a fast, open-source gradient boosting framework known for its high efficiency, scalability, and accuracy, particularly on large tabular datasets. This pipeline template was evaluated in the EGM pilot's water flow use case, where it trained on historical sensor data to forecast future values.
- **NGSI-LD Template Pipeline:** A utility pipeline template that facilitates the import and export of assets to and from the NGSI-LD Context Broker, enabling seamless integration with the SEDIMARK Marketplace for standardized asset sharing and interoperability.
- **deFLight_pipeline:** A utility pipeline template is available that facilitates the use of the deFLight SEDIMARK Marketplace asset service, which can be configured to operate either as a server or a client in a federated learning setup. deFLight is a SEDIMARK tool for decentralised machine learning described in SEDIMARK_D3.3 [5].



anomaly_annotator
{"desc":"This Pipeline is used to annotate anomalies in the process","type":"data_models"}
admin

ml_flow_lightgbm
{"desc":"This Pipeline is used to enforce the loading and exporting from and to NGSI-LD fdfdfds",...
admin

NGSI-LD Template Pipeline
{"desc":"This Pipeline is used to enforce the loading and exporting from and to NGSI-LD","type":"data_models"}
admin

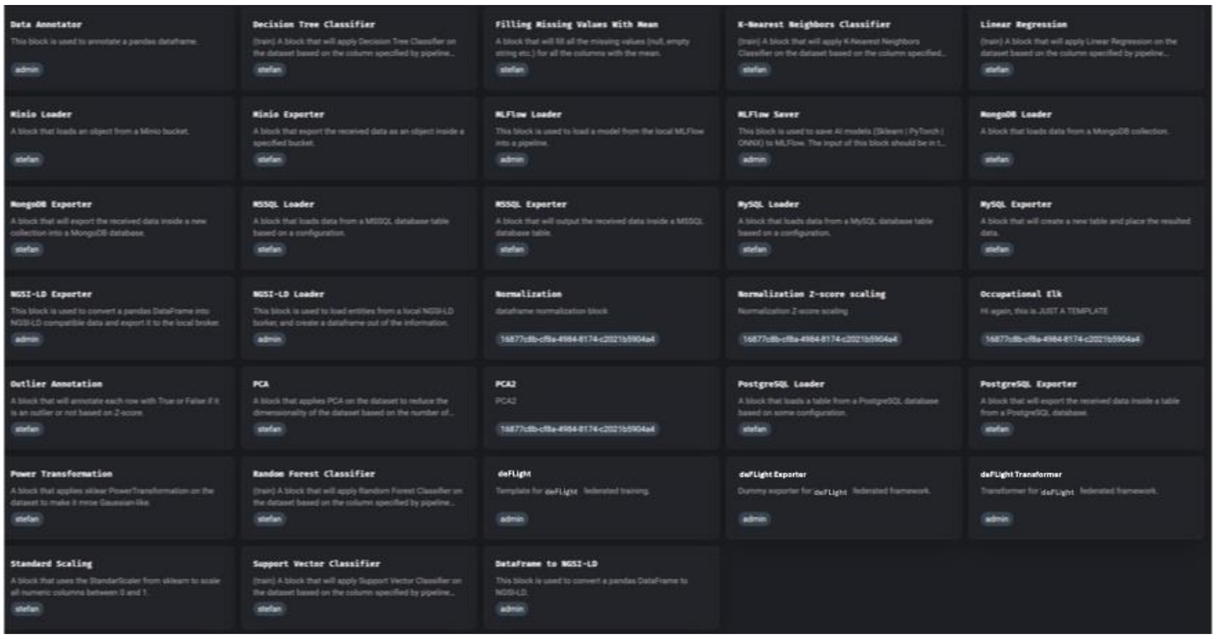
deFLight_pipeline
{"desc":"Simple deFLight pipeline to demonstrate federated learning assets","type":"federated_learning"}
admin

Figure 6: Mage AI pipeline templates.

Additionally, the Orchestrator features a pipeline-builder interface that allows users to assemble custom pipelines from predefined template blocks. When further customization is needed, users can even generate new blocks dynamically by leveraging large language models, enabling AI-assisted code generation and supporting a low-code or no-code development approach, as detailed in the next section. Some of the pre-built template blocks are illustrated in Figure 7 and can be categorized as follows:

- **Data Loaders:** Usually, the starting point of a pipeline, data loader blocks are used to load various types of resources, ranging from files and databases to services. These resources serve as the main input for the transformer blocks, some examples of such blocks are PostgresSQL Loader, NGSI-LD Loader and deFLight.
- **Transformers:** These blocks apply transformations to the input data, such as cleaning, normalization, enrichment, or feature engineering. They act as intermediate steps in the pipeline, modifying and preparing the data for subsequent processing or analysis. Some examples of blocks include Normalization, Standard Scaling and Support Vector Classifier.
- **Data Exporters:** Positioned at the end of the pipeline, data exporters handle the output of the transformed data, saving it to a target destination such as a file, database, or

external service. Some examples of data exporter blocks are MLFlow Saver, NGSI-LD Exporter and Minio Exporter.



Data Annotator
This block is used to annotate a pandas dataframe.
admin

Decision Tree Classifier
(train) A block that will apply Decision Tree Classifier on the dataset based on the column specified by pipeline...
stefan

Filling Missing Values With Mean
A block that will fill all the missing values (null, empty string etc.) for all the columns with the mean.
stefan

K-Nearest Neighbors Classifier
(train) A block that will apply K-Nearest Neighbors Classifier on the dataset based on the column specified...
stefan

Linear Regression
(train) A block that will apply Linear Regression on the dataset based on the column specified by pipeline...
stefan

Minio Loader
A block that loads an object from a Minio bucket.
stefan

Minio Exporter
A block that export the received data as an object inside a specified bucket.
stefan

MLFlow Loader
This block is used to load a model from the local MLFlow into a pipeline.
admin

MLFlow Saver
This block is used to save AI models (Sklearn | PyTorch | ONNX) to MLFlow. The input of this block should be in t...
admin

MongoDB Loader
A block that loads data from a MongoDB collection.
stefan

MongoDB Exporter
A block that will export the received data inside a new collection into a MongoDB database.
stefan

MSSQL Loader
A block that loads data from a MSSQL database table based on a configuration.
stefan

MSSQL Exporter
A block that will output the received data inside a MSSQL database table.
stefan

MySQL Loader
A block that loads data from a MySQL database table based on a configuration.
stefan

MySQL Exporter
A block that will create a new table and place the resulted data.
stefan

NGSI-LD Exporter
This block is used to convert a pandas DataFrame into NGSI-LD compatible data and export it to the local broker.
admin

NGSI-LD Loader
This block is used to load entities from a local NGSI-LD borker, and create a dataframe out of the information.
admin

Normalization
dataframe normalization block
16877c8b-cf8a-4984-8174-c2021b5904a4

Normalization Z-score scaling
Normalization Z-score scaling
16877c8b-cf8a-4984-8174-c2021b5904a4

Occupational Elk
Hi again, this is JUST A TEMPLATE
16877c8b-cf8a-4984-8174-c2021b5904a4

Outlier Annotation
A block that will annotate each row with True or False if it is an outlier or not based on Z-score.
stefan

PCA
A block that applies PCA on the dataset to reduce the dimensionality of the dataset based on the number of...
stefan

PCA2
PCA2
16877c8b-cf8a-4984-8174-c2021b5904a4

PostgreSQL Loader
A block that loads a table from a PostgreSQL database based on some configuration.
stefan

PostgreSQL Exporter
A block that will export the received data inside a table from a PostgreSQL database.
stefan

Power Transformation
A block that applies sklear PowerTransformation on the dataset to make it more Gaussian-like.
stefan

Random Forest Classifier
(train) A block that will apply Random Forest Classifier on the dataset based on the column specified by pipeline...
stefan

deFLight
Template for deFLight federated training.
admin

deFLight Exporter
Dummy exporter for deFLight federated framework.
admin

deFLight Transformer
Transformer for deFLight federated framework.
admin

Standard Scaling
A block that uses the StandarScaler from sklearn to scale all numeric columns between 0 and 1.
stefan

Support Vector Classifier
(train) A block that will apply Support Vector Classifier on the dataset based on the column specified by pipeline...
stefan

DataFrame to NGSI-LD
This block is used to convert a pandas DataFrame to NGSI-LD.
admin

Figure 7: Mage AI block templates.

3.4 Pipeline Orchestrator UI

The Orchestrator UI is designed as a user-friendly platform accessible to both technical and non-technical users. It enables users to create, execute, and monitor complex AI workflows with ease. To support this, the dashboard is built to emphasizes accessibility with a clean layout that organizes related information using colour coding and icons for quick identification.

The interface includes a suite of tools designed to provide detailed insights into pipeline metrics, helping users effectively monitor performance. The Orchestrator UI is tightly integrated with a Mage.ai client, which manages all pipelines and provides access to predefined templates. Both the Orchestrator and Mage.ai will be deployed as part of the SEDIMARK toolbox, and the templates are stored inside Mage.ai. The Mage.ai client URL is specified in the deployment configuration.

The platform enables users to create new pipelines either from predefined templates or by assembling them from blocks—these can be predefined or custom-generated. Additionally, users can load existing pipelines as needed.

The pipeline loading process is done using the Pipelines menu, which renders the new pipeline in a dedicated tab, as depicted in Figure 8. The platform supports multiple instances of the same pipeline running in parallel, enhancing workflow efficiency. Moreover, for each rendered pipeline, users can access logs at the block level and across all pipelines runs, along with hardware-related performance metrics.

Customization of pipelines is facilitated by setting variables on each block in Magei.ai, with values inputted upon execution of the pipeline.

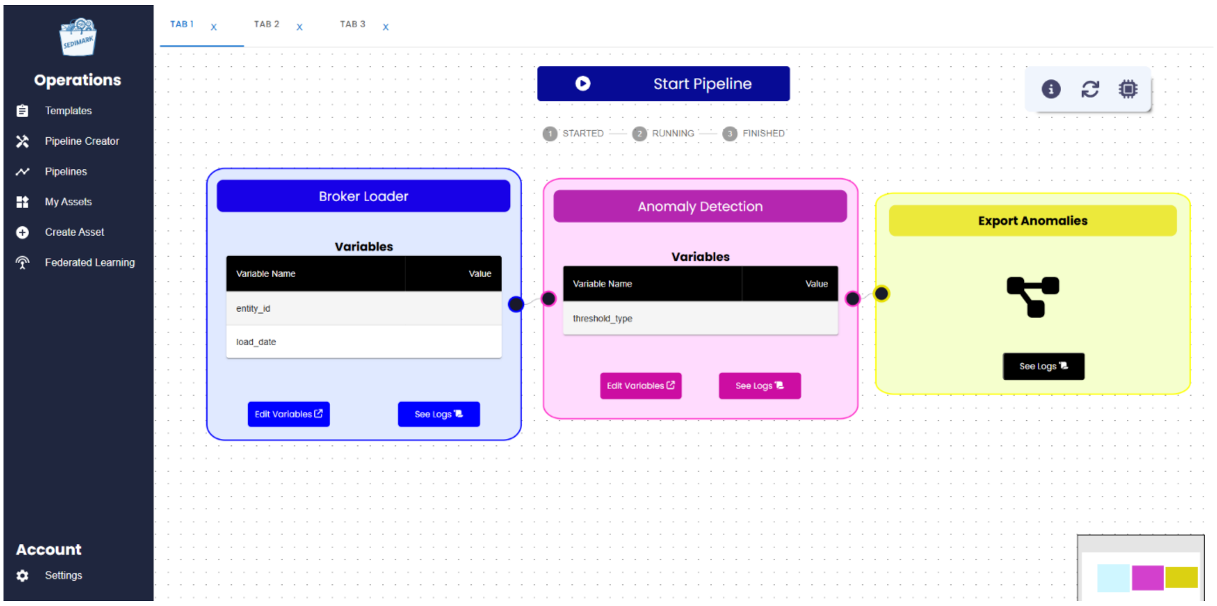
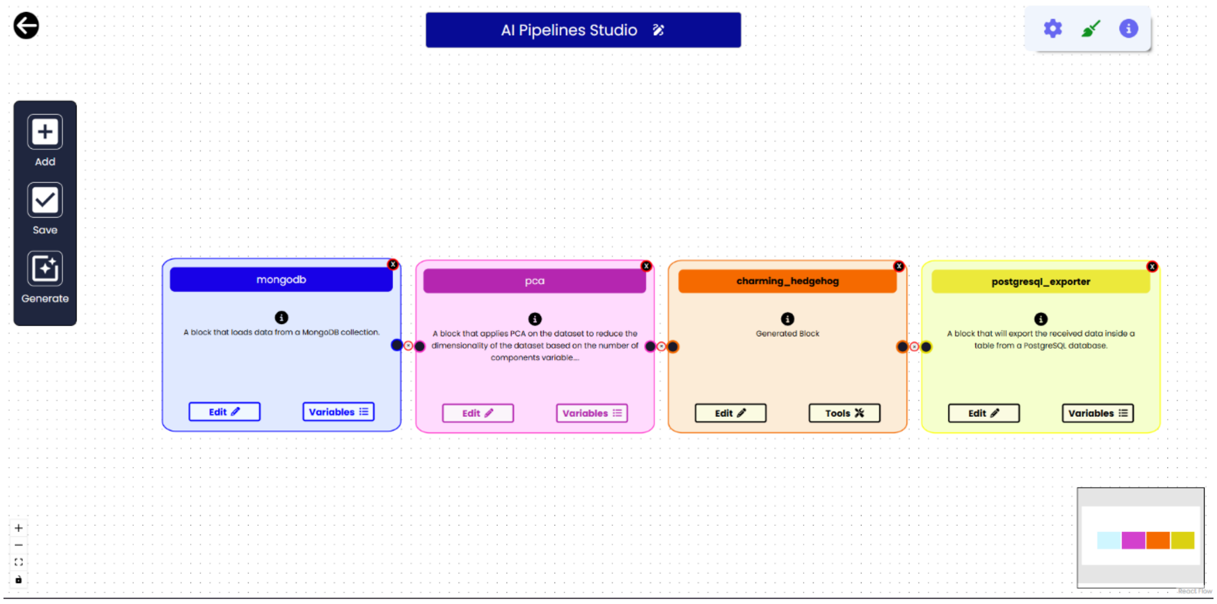


Figure 8: Pipeline Loaded in Orchestrator UI.

Another feature of the Pipeline Orchestrator UI is the Pipeline Creator, interface illustrated in Figure 9, which enables users to build custom pipelines using predefined block templates stored in Mage.ai, while also offering the ability to create custom blocks using an integrated block generator function. This function allows users to provide a prompt, resulting in a custom block that can be used immediately in the Orchestrator UI or saved for future use and modifications within Mage.ai.



A block that loads data from a MongoDB collection.

A block that applies PCA on the dataset to reduce the dimensionality of the dataset based on the number of components variable...

A block that will export the received data inside a table from a PostgreSQL database.

Figure 9: Pipeline Creator Interface.

The Pipeline Orchestrator also includes support for two federated learning services: Fleviden and deFLight, described in detail in section 5.2. These allow users to either offer a federated learning asset or utilize one they have previously purchased. Assets can be managed through dedicated menus, where users can view existing assets as illustrated in Figure 10, or create new ones directly from the interface.

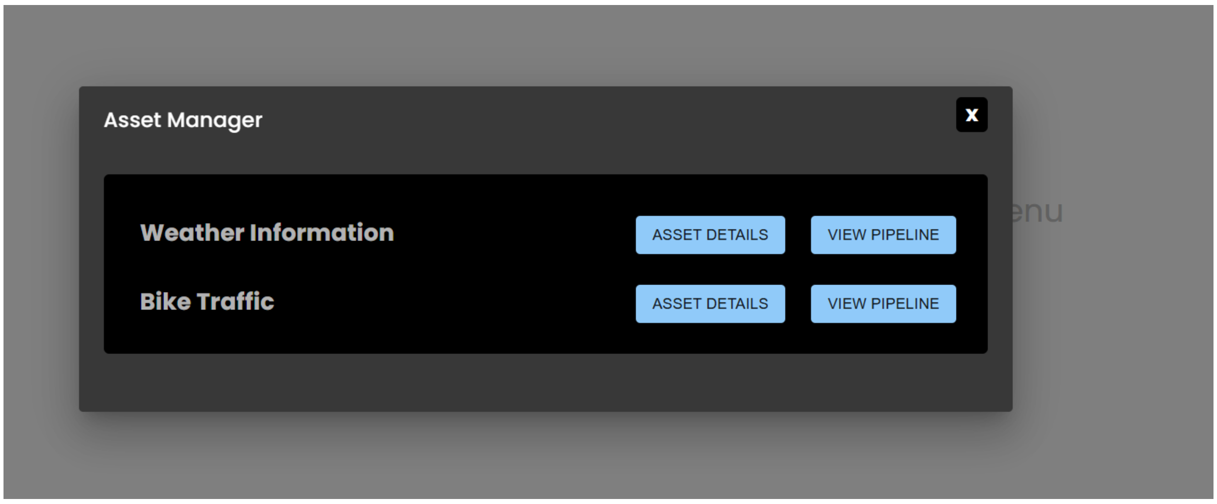


Figure 10: Asset Manager Interface.

3.5 Data quality metrics

As previously stated, improved data quality is of utmost importance to SEDIMARK, allowing both for data providers to increase their revenue, and for the training of improved ML models There has been a lot of research into data quality metrics and tools, which has resulted in four main data quality “dimensions” that basically look at the structure of the dataset, without going into much detail on the usability of the dataset for some specific purpose. SEDIMARK acknowledges that most of the datasets that will be generated, processed and shared will be mostly used for machine learning purposes. Data providers can use their datasets to gain knowledge from them, extract predictions or use them to guide their decision-making processes, while consumers can use the datasets i.e. for research purposes.

The four main data quality dimensions considered in the literature [9],[13] are accuracy, completeness, consistency and timeliness. However, SEDIMARK extends the data quality dimensions and defines more detailed and ML-related data quality metrics, to give (i) data providers more insights into their datasets and (ii) researchers and data consumers more in-depth information regarding how useful the datasets might be for them. Considering that SEDIMARK assumes two different types of datasets, data quality metrics can be split into two categories: (i) metrics per data point, useful for data streaming datasets and (ii) overall dataset metrics, useful mainly for static datasets. However, some overall dataset metrics are computed based on calculations using the metrics per data point.

3.5.1 Generic data quality metrics

The following are the considered generic metrics for data quality within SEDIMARK:

3.5.1.1 Accuracy

This is considered as the most important metric for the quality of a whole dataset, but usually it is the most difficult to measure, since it assumes that there is some knowledge about the “correct” points or the model of the “real world” entity that the dataset represents, so that it is possible to measure the distance of the dataset points from the “real-world”. Usually, this requires external input to measure. Inspired by [9], SEDIMARK defines the following metrics:

- **Datapoint Accuracy:** This indicates how close the measurement value is to the ground truth. The equation below shows the distance between the observed value and the reference value, taking the units of the observation value:

$$Datapoint\ Accuracy = |observedValue - refValue|$$

- **Dataset Accuracy**: There have been many different proposals about ways to measure data accuracy for an entire dataset, but, considering the metric about the datapoint accuracy defined above, usually the equation used is:

$$Dataset\ Accuracy = \frac{number\ of\ "accurate"\ observations}{total\ number\ of\ observations}$$

  However, this assumes that there will be a way to assess the “accuracy” of the observations in the dataset, probably using the “Datapoint Accuracy” metric, if the reference value is known.

3.5.1.2 Completeness

The Completeness metric can be split into two separate metrics, based on if it is a static or a streaming dataset.

- **Streaming completeness:** Streaming completeness represents the number of missed measurements within a given time window. The equation below incorporates three key parameters: rate, inter-arrival time value; n, number of missed measurements observed; and window, time window of observation. This completeness parameter can be expressed either as a part per unit or as a percentage:

$$Streaming\ Completeness = \frac{window - n \cdot rate}{window}$$

- **Static Dataset Completeness:** This is a metric inspired by [9] that measures if a dataset is complete or if there are some records or observations that are missing either as a whole or parts of the entries. Considering that a dataset might comprise entries that have different fields (columns), missing values can be either complete rows of entries or some specific fields in a row. A measure of completeness is given in the following equations:

$$Dataset\ Completeness\_1 = 1 - \frac{total\ number\ of\ missing\ fields\ per\ entry}{total\ number\ of\ fields}$$

  which sums the number of missing cells in a table over the total number of cells or

$$Dataset\ Completeness\_2 = 1 - \frac{total\ number\ of\ missing\ entries}{total\ number\ of\ expected\ entries}$$

  which sums the number of missing rows in a table over the number of expected rows.
  or

$$Dataset\ Completeness\_3 = 1 - \frac{num\ of\ entries\ with\ at\ least\ one\ field\ missing}{total\ number\ of\ entries}$$

  which sums the number of rows with at least one cell missing over the total number of rows.
  The usage of the respective metric depends on the target usage of the dataset.

3.5.1.3 Precision

Precision refers to the dispersion of values within a dataset, typically assessed through the dataset's standard deviation. The equation below shows the standard deviation formula used, where the variables involved are: μ, mean of the values in the dataset; n, number of samples in the set; and x_i, the i-th element of the dataset. The precision outcome aligns with the units of the dataset's values:

$$Precision = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}}$$

3.5.1.4 Consistency

Normally consistency is measured for datasets that are stored on different systems and is used to measure the number of records that match across the different systems. However, SEDIMARK inspired by [9] adopts a different interpretation of consistency, which measures how well the data points relate to their semantic rules or constraints. In this respect, the consistency metric uses the following equation:

$$Consistency = \frac{total\ number\ of\ entries\ following\ the\ semantic\ rules}{total\ number\ of\ entries}$$

3.5.1.5 Timeliness (Recency)

This metric normally measures how "current" or "recent" is a data entry for a specific task at hand. For example, when dealing with measurements from sensors, the timeliness of the data point can be measured on the difference between the current time and the timestamp when the measurement was taken. Within SEDIMARK, timeliness is measured via the following equation inspired by [9]:

$$Timeliness = max(0, 1 - \frac{Recency}{Volatility})$$

where

$$Recency = (time\ of\ storage) - (time\ observation\ was\ last\ updated)$$

and Volatility is the time length for which data remain valid.

Timeliness can also be calculated in data streaming scenarios in a recursive way by calculating a weighted average between the previous and the newly calculated mean update time. This allows consumers to appreciate the age or punctuality of data items. The equation below involves several parameters: α, correction factor belonging to the range [0,1]; rawTimeliness, raw value of the newly calculated update time; meanTimelines$_{i-1}$, mean value of the update time of the previous iteration; and meanTimeliness$_i$, mean value of the update time calculated in the current iteration.

$$meanTimeliness_i = \alpha \cdot meanTimeliness_{i-1} + (1 - \alpha) \cdot rawTimeliness$$

3.5.2 ML-based data quality metrics

Apart from the above mentioned "generic" data quality metrics, SEDIMARK defines and uses quality metrics that are more oriented towards the machine learning usage of the datasets.

3.5.2.1 Uniqueness

This metric shows how many unique records or labels (categories) exist in each of the features of the dataset.

3.5.2.2 Consistency

This metric is computed as presented above, comparing the values against a predefined range of values, which is assumed to be defined by the data provider.

3.5.2.3 Class parity

This metric is used for measuring the imbalance between the different classes/labels for each feature of the dataset. Within SEDIMARK, class imbalance is measured using three metrics:

- **Imbalance ratio,** which is the most commonly used metric in the literature and simply calculates the ratio of the sample size of the majority class over the sample size of the minority class, using the following formula when there are only two classes in the sample set:

$$Imbalance\ Ratio = \frac{number\ of\ samples\ in\ majority\ class}{number\ of\ samples\ in\ minority\ class}$$

  When there are multiple classes, the calculation of the imbalance ratio becomes more complicated and there have been multiple approaches to doing it, (i) by just using the majority and the minority class, (ii) by summing the minority classes or (iii) by using entirely different metrics specifically designed for multiclass problems, such as the "imbalance degree" [14].

- **Normalised cross entropy** [18],[19], which measures the imbalance of the classes in a feature of a data set by computing the normalised entropy of the class sizes, measured by:

$$Normalised\ Class\ Entropy\ (NCE) = -\frac{1}{log(c)}\sum_{k=1}^{c}\frac{n_k}{n}log\frac{n_k}{n}$$

- **Likelihood ratio imbalance degree (LRID)** [20], which was proposed to handle multi-class imbalance based on the likelihood ratio test, computed by:

$$LRID = -2\sum_{k=1}^{c}n_k ln\frac{N}{cn_k}$$

3.5.2.4 Regularity

This is measured for time-series datasets as the mean difference between consecutive entries of timestamps in the dataset. This can show i.e. if the dataset has entries taken at regular intervals or not and can be used also as metric of time consistency and of time completeness.

3.5.2.5 Time Completeness

This metric is basically used for time-series datasets that are gathering measurements at fixed intervals. The regularity of the dataset can be used to compute if the dataset is complete in the given time period of the measurements by comparing the number of observations against the number of expected observations. For example, if there is a sensor that measures temperature every 10 minutes, the expected number of observations within 1 day is 6*24=144 observations. If the dataset for that day consists of 110 observations, it means that the completeness is 110/144=76.39%. So, the formula to compute time completeness in SEDIMARK is the following:

$$Time\ Completeness = \frac{number\ of\ observations}{number\ of\ missing\ observations + number\ of\ observations}$$

where the "number of missing observations" is calculated using the regularity and finding out gaps between consecutive measurements that are bigger than the median difference between consecutive measurements.

3.5.2.6 Feature Correlation

This is measured by comparing couples of features of the dataset and measuring how well/badly they correlate with each other. Taking the average of all couples gives the average feature correlation of the dataset. High correlation between features shows a linear dependency, so this metric can be exploited by users to remove highly correlated features from input to ML model training processes, since they will have very similar effect. To measure that the *"corr()"* function of *pandas* data frames is used.

3.5.2.7 Collinearity

This metric measures the dependency of couples of features on a regression model. To measure that, inspired by [16] SEDIMARK built a function that aims to predict each feature by using the other "N-1" features and predict the Pearson correlation between the prediction results.

3.5.2.8 Class Overlap

This metric measures the overlap of classes/labels within a feature and how they can be differentiated (or not) using the other features. Class overlap occurs when instances of some of the labels of a feature are mapped on the same area of a feature space, showing that they have many similarities. Although there are many metrics used to measure class overlap, currently SEDIMARK uses Fisher's discriminant ratio as defined in [21],[22] due to its simplicity.

3.5.2.9 Unalikeability

This metric measures how similar or different are the values within a feature column. SEDIMARK adopts the unalikeability coefficient as described in [24] using the following equation:

$$Unalikeability = 1 - \sum_{k=1}^{c}(\frac{k_i}{n})^2,$$

where k_i is the number of samples of class "i", "n" is the total number of observations in the feature column and "c" is the number of classes of the feature column.

3.5.2.10 Number of outliers

This metric uses the output of the outlier detection models of SEDIMARK's data curation pipeline and measures the number of values detected by the model to be outliers or noise.

3.5.2.11 Artificiality

As it is critical to improve the quality of data assets, it is also important to keep track of the proportion to which a data asset being produced is by artificial (synthetic) processes rather than occurring naturally. This is especially the case in relation to handling original data assets with missing data points. Inspired by [15], a measure for this can be as follows:

$$Artificiality = \frac{A_{syn}}{A_{syn} + A_{org}}$$

Whereby A_{org} is the number of the original data in the dataset, and the A_{syn} is the amount of artificial data that have been generated through a synthetic process.

3.5.2.12 Stationarity

Metrics to analyse time series data can be very useful for data providers that aim to use their data for modelling and interpretation. A time series data can be stationary when its statistical properties, such as mean, variance and covariance, aren't dependent on the time upon which the data series is observed [75]. In other words, a data series is stationary when there is no link between a value and its previous values. To calculate stationarity in SEDIMARK, the Augmented Dickey Fuller test [75] and the Kwiatkowski-Phillips-Schmidt-Shin test [76] have been implemented using the statsmodels [77] python library. Both tests provide several values for a result i.e. the test statistic, the p-value and the critical values at 1%, 5% and 10% levels. The easiest way to interpret the result is using the p-value and comparing it to a significance level (usually 5%). If the p-value is higher than the significance value then it means that the data series is non-stationary.

3.5.2.13 Entropy

Entropy in machine learning can be used to measure the uncertainty in the dataset about the outcome of a variable and the level of disorder in the dataset. This can be used to evaluate the ability of a model to make predictions using that dataset. Entropy in Information theory was defined by Shannon in the seminal paper [78] using the below formula, which has been implemented in the data profiling module using python:

$$\text{Entropy} = -\sum_{k=0}^{n} p_k * \log_2 p_k$$

3.5.2.14 Variance Inflation Factor (VIF)

For measuring the multicollinearity in multivariate time series, SEDIMARK has implemented the Variance Inflation Factor (VIF) [21]. This metric indicates how much the variance of an estimated regression coefficient increases when the predictors are correlated. A high VIF signifies that the feature can be linearly predicted from others. In SEDIMARK, we use the “variance_inflation_factor” of the statsmodel python library [78]. This function gives a double value as a result and assumes that if the value is above 5 then the two variables are highly colinear.

3.5.2.15 Predictive Power Score (PPS)

Another metric for correlation between two variables in a multivariate data series is the Predictive Power Score (PPS), defined in [79]. The score aims to detect both linear and non-linear correlations between two variables. In SEDIMARK we use the ppscore library [79] for calculating the PPS. The output is a value between 0 and 1, with 1 showing the highest correlation (or predictive power) between the two variables.

3.6 Data Profiling and Data Quality Evaluation

Data Profiling is a functional component of the SEDIMARK architecture that provides a single solution for users of the data processing pipeline to gather insights about the quality of their data. Data Profiling is usually the process of examining, analysing and presenting useful summaries to the users so that they can identify problems with their data and find out ways to improve them. There has been a lot of work done to build data profilers in the past, but most of the existing tools are either dedicated to specific types of data or are focused on general descriptions of datasets, without going into too much detail about the usability of the datasets for machine learning purposes.

Example of tools for data profiling are the following:

pandas describe(): Pandas is one of the most used libraries in Python for handling data frames and performing data analytics tasks. Pandas offers the “*describe()*” function to provide descriptive statistics about the data frames, analysing both numerical and categorical data series. The SEDIMARK Data Profiling component exploits the pandas "describe" function to get a quick overview of some statistics of the dataset, including number of rows, unique rows, mean values, etc.

ydata-profiling [25]: This is a python library that provides simple ways to get statistics about datasets, featuring an easy to use interface and graphs, together with a large number of metrics, including both general statistics and more data-analysis oriented statistics.

IBM data quality [26]: IBM provides an API for assessing the quality of data, including data profiling. It provides insights about class parity, class overlap, data homogeneity, completeness, outlier detection, etc. However, the API and the backbone code for computing these statistics are not open source.

The Data Profiling component of SEDIMARK is part of the Data Curation Enabler as discussed in SEDIMARK_D2.3 [4] and shown in Figure 3. The Data Profiling component is considered as the first step in data curation, taking as input the dataset formatted in the internal SEDIMARK format (which is a new data source class as a wrapper above a *pandas* data frame). The Data Profiling component is tightly integrated with the Data Quality Evaluation component which assesses the quality of the incoming data. In the current implementation, both components are integrated into a single python module that does both the profiling and the quality assessment using the metrics described in the previous section.

Within SEDIMARK, the Data Profiling module is split into two parts:

- Providing **generic** statistics about the dataset (which corresponds to the Data Profiling component):
  - Number of features.
  - Number of entries (observations).
  - Total number and percentage of missing cells.
  - Total size of the dataset and size per feature column.
- Providing **feature-specific** statistics based on the data type of the feature column (which corresponds to the Data Quality Evaluation component).

The feature-specific statistics of the dataset are based on the data type:

- **Numerical** columns:
  - Percentage of missing, duplicate, unique cells.
  - Min/max/average/std/median values.
  - 25%, 50%, 75% percentiles, which shows the value point below which lie the 25%, 50% or 75% of the data values.
  - Histogram, which shows the frequency of the distribution of the data points across the range of the values.
  - Consistency (data within ranges that are pre-defined).
  - Skewness, which measures the shape of the distribution of the feature, and basically shows if the distribution is symmetric or asymmetric. It can be negative (showing that the tail of the distribution is on the left side), zero (symmetric distribution) or positive (showing that the tails is on the right side).

- Kurtosis, shows the shape of the distribution, with a normal distribution having a value of 3. Values less than 3 mean that the distribution is platykurtic, while a value greater than 3 means that the distribution is leptokurtic.
    - Mean absolute deviation, which is the mean absolute distance of each data point from the mean of the distribution.
- **Boolean** columns:
    - Name, type, missing, duplicates, unique.
    - Label frequency.
    - Imbalance statistics.
- **Object** (string) columns:
    - Name, type, missing, duplicates, unique.
    - Min/max/average/median length of string and the histogram.
    - Label frequency per label (assuming it's a label/class columns).
    - Imbalance statistics (imbalance ratio (IR), imbalance degree, log likelihood index, tangential imbalance index, normalised entropy).
    - Class overlap, measured by Fisher's discriminant ratio: higher values == higher complexity - classes can't be differentiated by the features.
    - Unalikeability, as a measure of how similar/different the values in the column are (0 == all the same, 1==all different).
- **Datetime** columns:
    - Name, datatype, missing, duplicates, unique.
    - Minimum/maximum data difference between consecutive values.
    - Mean, median, interquartile range (iqr), standard deviation (std), regularity (mean/median difference between consecutive dates) for the column values.
    - Histogram of date differences.
    - Regularity.
    - Number of missing values based on regularity.
    - Completeness based on regularity.

Streaming data profiling

For streaming data, the profiling of the data and the evaluation of their quality within the SEDIMARK streaming pipeline, relies on a trusted external source for accuracy (AEMET [23] in the case of temperature in the city of Santander), and on an NGSI-LD Context Broker with support for storing temporal values for the rest of the dimensions. The output of the streaming data quality module is based on an aggregation of the values of the assessed dimensions, selected upon the user's request. This way, a data point quality characterisation is obtained, allowing the user or the consuming application to have all the necessary knowledge to decide which pieces of data to use or not, i.e., whether they meet their quality requirements or not. Figure 11 shows a representative scheme of the module depicting the input needed and the outputs expected.

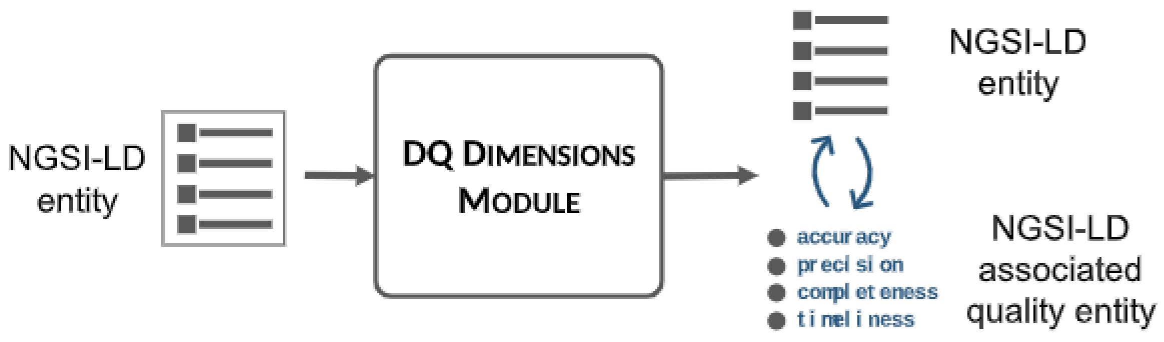


Figure 11: DQ Dimensions Module for streaming environments

3.7 Data Curation

As shown in Figure 3, data cleaning within SEDIMARK consists of four main modules: (i) outlier detection, (ii) noise removal, (iii) deduplication and (iv) missing value imputation. Aiming to have a consistent and homogeneous way to execute the functions in these modules, so that it is easier to be launched by the Data Processing Orchestration, all modules have been developed in a similar way, as it is depicted in Figure 12. All modules have the same predefined way to get the data as input and these are being managed by a “pre-processing” function, which transforms the data to the format required by the internal cleaning module. Then, the processed data are forwarded to the cleaning algorithm, which does the cleaning job and forwards the results to the “post processing” module that creates the output data in a predefined format to be handled by the Data Processing Orchestration. Depending on the user configuration, the output data can be either the “cleaned” data (i.e. with the outliers or the duplicates removed) or the original input dataset with annotations, flagging the entries that were found to be outliers, noise or duplicates or the entries that were imputed.

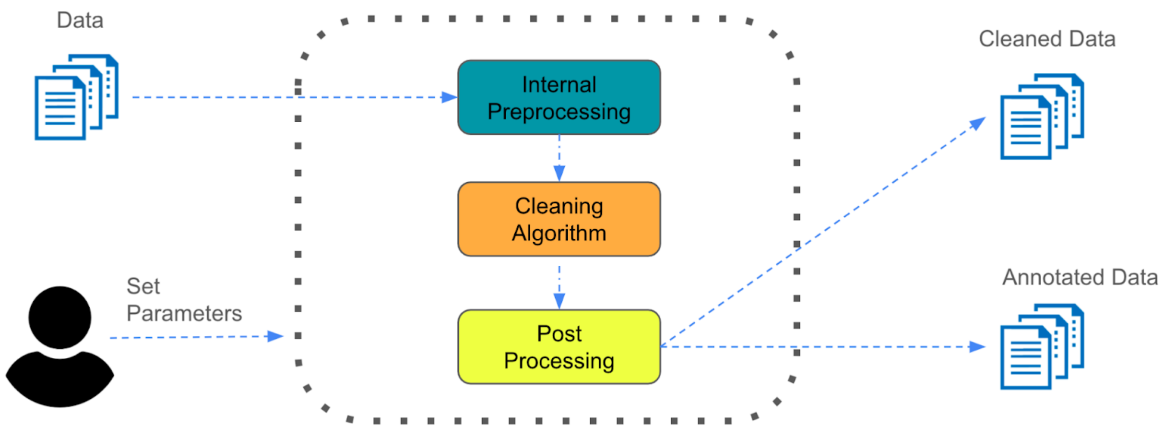


Figure 12: A generic SEDIMARK data cleaning module.

3.7.1 Outlier Detection

An AI-Based Outlier Detection (OD) tool harnesses the power of machine learning to identify anomalies in multi-dimensional data. Utilizing advanced algorithms like the Isolation Forest and One-Class SVM, outlier detection scans through datasets to differentiate regular data points from outliers. Combining several approaches in a common module provides a robust mechanism for outlier detection, allowing data scientists and analysts to clean their datasets effectively and ensuring the integrity of any subsequent data analysis or machine learning model training.

SEDIMARK considers outlier detection as a key mechanism to improve the quality of datasets, reducing the amount of low-quality data, while contributing to reducing the size of datasets and in the end improving the machine learning models built on top of these cleaned datasets. Considering the type of dataset, SEDIMARK has developed components for outlier detection on both offline and streaming datasets, as discussed in the next paragraphs.

3.7.1.1 Outlier detection for offline datasets

The goal of Outlier Detection (OD) in a data pipeline is to identify and potentially remove data points that are highly anomalous and appear to be generated by a data generation process differing from the one that the data scientist wants to model. The presence of anomalies in data can cause severe problems for downstream ML models, and can also indicate that the data has been corrupted or even maliciously poisoned [38] SEDIMARK will aim to either remove or clearly flag as many anomalies as possible within the datasets published to its marketplace, based on the configuration or preferences set by the data provider.

There are numerous algorithms for tabular anomaly detection, with popular approaches including Local Outlier Factor (LOF), One class Support Vector Machine, and recently deep learning based approaches such as Deep Support Vector Data Description (SVDD) [35] or classifying anomalies per their autoencoder reconstruction error [33]. These algorithms can be classified broadly as based on a) a statistical model, b) density c) distance d) clustering e) isolation f) ensembles and g) subspaces [27]. However, the same outlier detection methods cannot be naively applied in the case of time series data, where the sequential dependency between data points introduces an extra dimension that can be key for recognising outliers [37]. As such, while tabular anomaly detection algorithms will most often work on the raw data points, in the case of time series data the algorithms generally process chunks or trajectories of the data, classifying these as anomalous.

An additional problem in anomaly detection is that the OD algorithms themselves generally output only raw outlier scores and probabilities, with classification determined by the application of a threshold. While it is customary to set this according to an a priori assumption about the dataset (e.g. that 1% will be outliers) the number of outliers in a dataset is not generally known by the data scientist cleaning the data. As such a number of techniques such as AUC Percentage (AUCP) [36], gamma Gaussian Mixture Models (gammaGMM) [34] and Generalised Extreme Studentized Deviation ESD (GESD) [28] exist to automatically determine this threshold automatically by processing the raw anomaly scores.

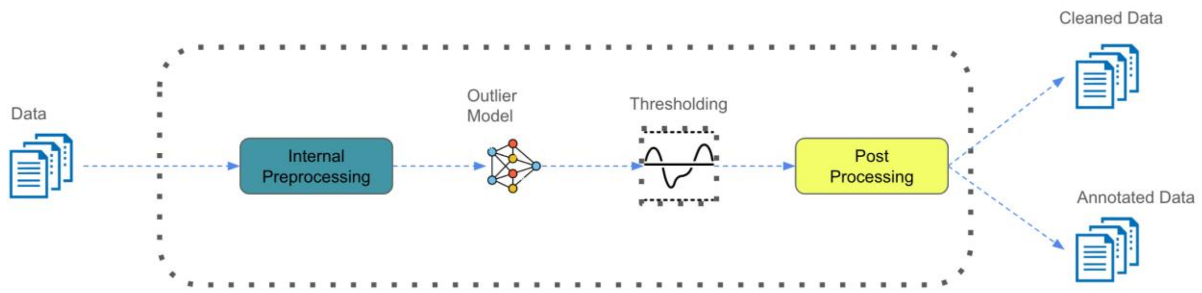


Figure 13: SEDIMARK outlier detection module.

The SEDIMARK outlier detection module is built in a generic way so that it can easily incorporate a number of existing libraries, avoiding the effort of re-implementing well-known outlier detection algorithms from scratch. The SEDIMARK outlier detection module currently makes use of two key libraries for outlier detection, namely PYOD [40] for the case of tabular data, an adaption of the benchmarking suite TimeSeAD [85]. Both PYOD and TimeSeAD include a large number of unsupervised OD approaches, such as traditional machine learning methods like KNN based methods [29], one class Support Vector Machines (SVMs) [30], or simple autoregression models [31] in the case of time series, as well as modern deep learning algorithms such as autoencoders, or in the case of time-series data, Long Short Term Memory (LSTM) based methods [32]. In addition, SEDIMARK makes use of the PyThresh library [39] to automatically set anomaly thresholds, with included methods such as AUCP, gammaGMM and GESD.

As such, the Anomaly detection module follows the flow shown in Figure 13 where input data first undergo pre-processing to be PYOD/TimeSeAD compatible and then pass through outlier detection and thresholding algorithms. After that, the data are post processed, where they are either annotated with outlier descriptors (a Boolean indicating threshold result, and the raw outlier score), or the outliers are discarded.

At present, SEDIMARK includes the following algorithms from PYOD:

- **ABOD:** Angle based outlier detection, classifying anomalies based on the variance of their cosine scores to their neighbours.
- **AutoEncoder:** Neural anomaly model, classifying data points according to their reconstruction error when passed through an autoencoder.
- **Feature Bagging:** Takes the average of several base detectors, trained on subsets of the data features.
- **HBOS:** Histogram Based Outlier Selection, assumes feature independence and then constructs histograms to infer the extent to which data points are outliers.
- **IForest**: Isolation Forest, which classifies outliers based on their average path length in a forest of partition trees.
- **KNN:** A K-Nearest-Neighbours based approach which takes a data points distance to its kth neighbour as an estimate of its outlierness.
- **LOF:** Local Outlier Factor, which compares the densities of data points with respect to their neighbours.
- **MCD:** Takes the Minimum covariance distance in a gaussian distributed dataset as the degree of outlierness.

- **OCSVM:** One-class Support Vector Machine, which fits a decision boundary around the assumed normal training data.
- **PCA:** Projects the data using Principal Component Analysis, taking the projected distance of a data point as its degree of outlierness.
- **DeepSVDD:** Deep One Class classification for outlier detection, which encloses learned representations of the data within a hypersphere, with the distance from the centroid then used as the degree of outlierness.

TimeSeAD provides four general classes of algorithms, namely:

- **Baselines:** A number of well known baseline algorithms such as those based on nearest neighbours, PCA, or decision trees.
- **Reconstruction:** Autoencoder based models that are trained to compress and then reconstruct their own input, with the reconstruction error used as a signal for outlier detection.
- **Prediction:** Models based on autoregressive prediction, where the prediction error of subsequent timesteps is used as the main signal for outlier detection.
- **Other:** Contains a number of not readily classified algorithms such as THOC [86] and NCAD [87].

A drawback of the traditional approach to outlier detection is that it requires either domain expertise, or a number of labelled outliers, in order to be able to properly assess and thus finetune an outlier detection pipeline component to a new dataset. Outliers can themselves often be highly domain dependent, and thus without expert involvement it is difficult to assess whether retrieved data points are in fact outliers or not.

3.7.1.2 Outlier detection for data streams

In the realm of data cleaning, streaming data processing presents unique challenges compared to offline (batch) data processing. Streaming data cleaning involves the continuous and real-time cleansing of data as they are generated, necessitating immediate actions to ensure data quality and reliability. This is in stark contrast to offline data cleaning, where data are accumulated over time and cleaned in bulk. The critical difference lies in the immediacy and ongoing nature of streaming data cleaning: it requires rapid and continuous cleaning mechanisms, which are crucial for applications demanding quick and accurate decision-making, such as real-time monitoring systems, fraud detection, and live financial trading. On the other hand, offline data cleaning is more suitable for scenarios where immediate data quality is not as critical, allowing for more thorough and comprehensive cleaning processes for large datasets, albeit with some delay in data readiness. Streaming data cleaning demands high scalability, performance, and robust error handling due to its dynamic nature, while offline cleaning allows for more extensive and deep cleaning processes, suitable for large-scale data analyses where time is not a pressing factor.

River [41], a specialised Python library for online machine learning, is particularly adept at handling the complexities of streaming data. It provides a range of tools and algorithms for real-time analysis and processing, making it an ideal choice for applications that require immediate, continuous data handling, such as in Internet of Things (IoT) devices, financial market analysis, or web activity monitoring. In the context of SEDIMARK, River plays a crucial role in the data cleaning process for streaming data. Recognizing the inherent challenges in managing and maintaining the quality of streaming data, SEDIMARK leverages River's advanced functionalities for effective data cleaning. This includes a comprehensive suite of outlier detection methods, each tailored to specific types of data and anomalies.

For outlier detection, SEDIMARK includes several robust methods inherited from River:

- **GaussianScorer:** This method utilises a Gaussian distribution to score anomalies, identifying data points that significantly deviate from the expected distribution.
- **Half-Space Trees (HST):** Ideal for high-dimensional data, HST uses space-partitioning trees to detect regions with lower data point density, indicating potential anomalies.
- **LocalOutlierFactor (LOF):** LOF measures the local density deviation of a data point compared to its neighbours, making it effective for identifying local outliers within the data.
- **OneClassSVM:** This method employs a one-class support vector machine to isolate outliers, particularly useful in datasets where outliers and normal data points are distinctly separable.
- **QuantileFilter:** It filters out data points that lie beyond defined quantile ranges, based on a scoring function, effectively identifying extreme values (values that are above a specific percentage).
- **StandardAbsoluteDeviation:** This technique scores outliers based on their absolute deviation from the mean, relative to the standard deviation, ideal for datasets where such deviations are indicative of anomalies.
- **ThresholdFilter:** It applies a predefined threshold to identify anomalies, suitable for situations where the threshold for anomalous behaviour is known.

Furthermore, a novel anomaly detection algorithm based on the Exponentially Weighted Moving Average [120] has been also implemented. This algorithm is intended to achieve the highest possible detection precision while maintaining a computational time compatible and appropriate to the system. Three variants of the algorithm, each considering different options for its model updating, have been defined. Each variant shows different adaptability to different use cases and scenarios.

Given the dynamic nature of such data streams, it is appropriate to consider moving average-based solutions. EWMA performs better than the other methods in terms of capturing the trend and seasonality of the data. In other words, EWMA-based methods have the lowest forecasting error as they capture substantial changes faster than others. Generally, this kind of method is used to remove noise from the time series and smooth it, thus allowing further data processing techniques to face fewer challenges. This method can be represented by the equation below, where x stands for the observation being evaluated and α for the smooth factor. As can be seen, this α factor denotes the weight given to the most recent values, thus allowing the EWMA method to be more robust against changes in the trend.

$$EWMA_i \equiv \begin{cases} x_i, & i = 1 \\ \alpha \cdot x_i + (1-\alpha) \cdot EWMA_{i-1}, & i > 1 \end{cases}$$

The proposed algorithm for real-time anomaly detection in data streams is based on a threshold approach. As a core operation, for each received observation, an expected value is calculated through EWMA, and upper and lower bound values are set using the standard deviation of the dataset. When the current observation falls outside these limits, it is considered an anomaly. It is worth noting that the estimation of both the expected value and the limits is performed using the information available in the model prior to the arrival of this new observation, in order to avoid biasing the results.

Three variants of the algorithm have been developed, each improving the performance and detection results of the previous one, as well as its capacity to adapt to changes and variations of the time series to be evaluated. They all allow for the following input parameters: records, indicating the number of values to be used in the EWMA calculation; threshold, being the sigma factor; α, corresponding to the EWMA importance factor. Additionally, there is another configurable parameter that determines the maximum number of values included in the model.

The first variant, so-called "without update", is characterised by not updating the model as new observations arrive. In other words, the model is based on a static photograph of the training data against which estimations are made. Although such a model may not seem to be useful for any scenario, it is possible to find use cases where the time series does not vary significantly from its initial state, except for the so-called anomalies. Hence, this would allow for a simpler and computationally faster model. Nevertheless, considering the use case on which we are focusing (i.e., variable time series from IoT environments), this option is not so well suited.

The second variant compensates the previous one's main problem by updating the model, hence we have named it "with update". In this case, the model used for performing the estimations is updated every time an observation is recorded, without differentiating between detected anomalies and inliers (i.e., the model update is done whether the observation is qualified as outlier or inlier). This approach adapts better to the behaviour of the time series that have been used to assess the proposed solutions, although the occurrence of several anomalies together can spoil the performance of the algorithm, as they are biasing the model towards dirty data. However, in use cases where sudden changes can occur and last over time, this version of the algorithm is capable of adapting even in the face of a sequence of false positives.

Finally, the last variant of the algorithm performs a smart update of the model, i.e., the update only occurs when the assessed observation is determined not to be an anomaly (i.e., the model update is done only if the observation is qualified as inlier). Thus, the model data is clean and truly captures the trend and seasonality of the time series. This latter variant is known as "with smart update".

The streaming outlier detection module of SEDIMARK follows the same concepts as discussed in the main data cleaning section, namely it is generically built as a wrapper on top of River to allow the easy execution of the cleaning models from the Data Processing Orchestration. In the end, the output of the module includes the processed data point (or batch) with the detected outliers flagged as such.

3.7.2 Noise cancellation

The IoT process generates an enormous amount of data. When the data collected from IoT sensors is of poor quality, due to measurement error or environmental factors such as atmospheric or electromagnetic disturbances, it will create noisy data (random undesired fluctuations or errors) and low signal-to-noise ratios (SNRs). This leads to distortion of the original signal, difficult analyses and interpretations of the base patterns, reduction in the quality of predictions, and in fine misdirection of Smart Services. Beyond the IoT domain, noisy data exists in many application domains, such as communications, acoustics, or biomedical engineering [42].

Although the noisy values can be reduced using high precision sensors with inbuilt noise reduction mechanisms, such sensors come at a cost and that limits their deployment. Hence, noise reduction or cancellation is a crucial step in signal processing, to produce quality diagnostics and forecasting results. Noise cancellation techniques enhance the signal in the original form by removing unwanted elements from the measurements [43].

The complex and misunderstood causes of noise in measurements, and the time-varying environment, result in multiple kinds of noise, to name a few: Gaussian noise, white noise, periodic noise, impulse noise, coloured noise, and heteroscedastic noise. Noise prediction for noise cancellation is hence a very arduous task and should ideally consist of a system that can automatically adapt to the environmental changes [42]. The challenge is to detect the actual useful signals within a strongly noisy background, particularly when the noise is non-stationary. If the chosen processing method can effectively predict and eliminate the noise, then the result is ideal. Otherwise, the noise will not be cancelled, and the original signal will be weakened [42].

There exists a very broad range of methods and techniques that try to cancel or mitigate noise in time series data. The choice of the method depends on the specific characteristics of the noise and data system, the presence of trends, seasonality, the SNR level, the objectives of noise reduction.

3.7.2.1 Specific to noise cancellation method

Adaptive and Neural network based noise cancellation methods are specific to a signal. They rely on learning from historical or reference data to distinguish signal from noise. That makes them efficient and able to remove complex noise but decrease the ease of use of such methods.

Adaptive noise cancellation filters are widely used in communication and signal processing systems. A comparison of three popular adaptive filtering algorithms used for real-time noise reduction is given in [42]: (1) Least Mean Squares (LMS) filter: Minimizes the error between the clean signal and the filter output. It is limited in the occurrence of outburst interferences.; (2) Normalized LMS (NLMS) filter: Normalizes the filter coefficients depending on the input signal's energy. Useful when the amplitude varies significantly; (3) Recursive Least Squares (RLS) filter: Minimizes the sum of errors with time. Outperforms LMS and NLMS techniques due to its high convergence speed and efficiency.

Among neural network-based approaches, a particularly promising technique is the Denoising Autoencoder (DAE) [121]. DAE is a type of deep learning model designed specifically to remove noise from input signals by learning to reconstruct the original, clean signal from noisy data. The DAE is trained using pairs of clean and artificially corrupted signals, allowing it to learn the underlying statistical regularities of the clean signal. Once trained, it can be used to denoise previously unseen noisy inputs.

3.7.2.2 Generic noise cancellation method

Generic noise cancellation methods have the advantage of working out of the box for every signal. This means that data services using those methods will work out of the box on any received signals

We focused on three methods that needs minimal configuration and only use the last values of the signal. the number of values used for the calculation is called window size and can be configured. Specifying a greater window size will smooth the signal at the cost of decreasing peaks value in the signal.

The weighted moving average (WMA) [122] smooths a signal by assigning varying weights to past values, typically giving more importance to recent observations. This makes the filter more responsive to short-term changes while still reducing noise. Unlike a simple moving average, the WMA minimizes the lag by favouring the latest data, which is often more relevant in real-time processing contexts (see Figure 14).

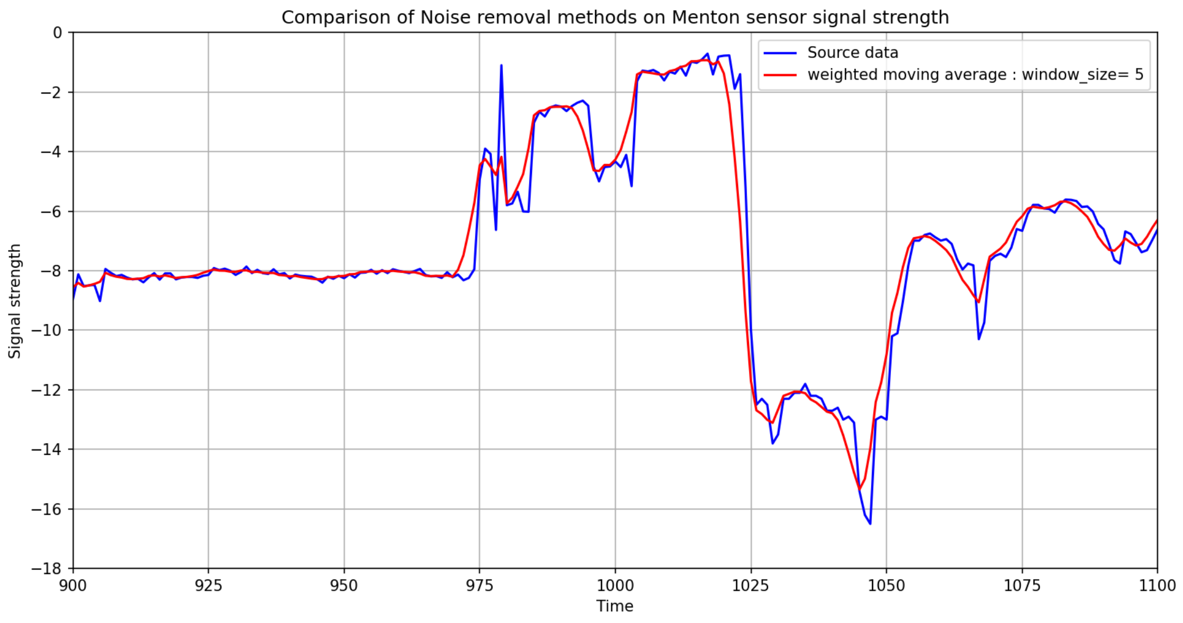


Figure 14: Weighted moving average applied on Menton sensor signal

The Savitzky-Golay filter [123] performs polynomial regression to smooth data while preserving the shape and features of the signal. Unlike moving averages, the Savitzky-Golay filter maintains sharp transitions in the data, making it suitable for biological and environmental signals where feature preservation is crucial. In addition to the window size the polynomial order can be adjusted to control the level of smoothing and fidelity (see Figure 15).

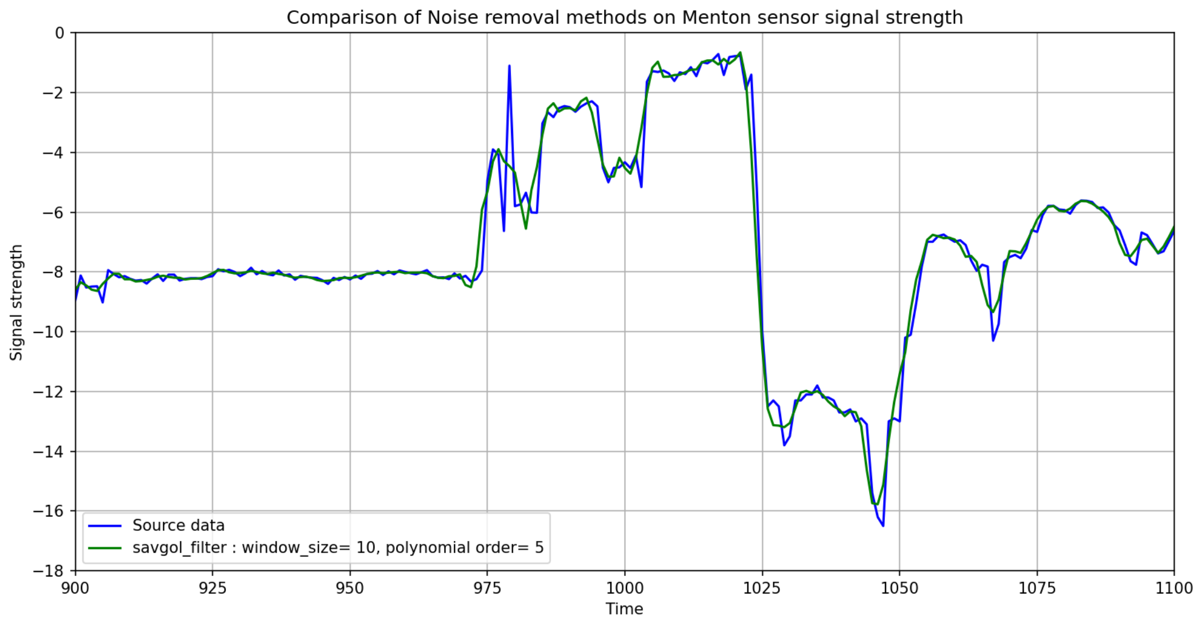


Figure 15: Savitzky-Golay filter applied on Menton signal

The Butterworth low-pass filter [124] is designed to have a maximally flat frequency response in the passband. It removes high-frequency noise components while minimizing distortion in the low-frequency signal range. In addition to the window size the cutoff frequency and filter order can be adjusted to control what frequency the filter will keep or remove (see Figure 16).

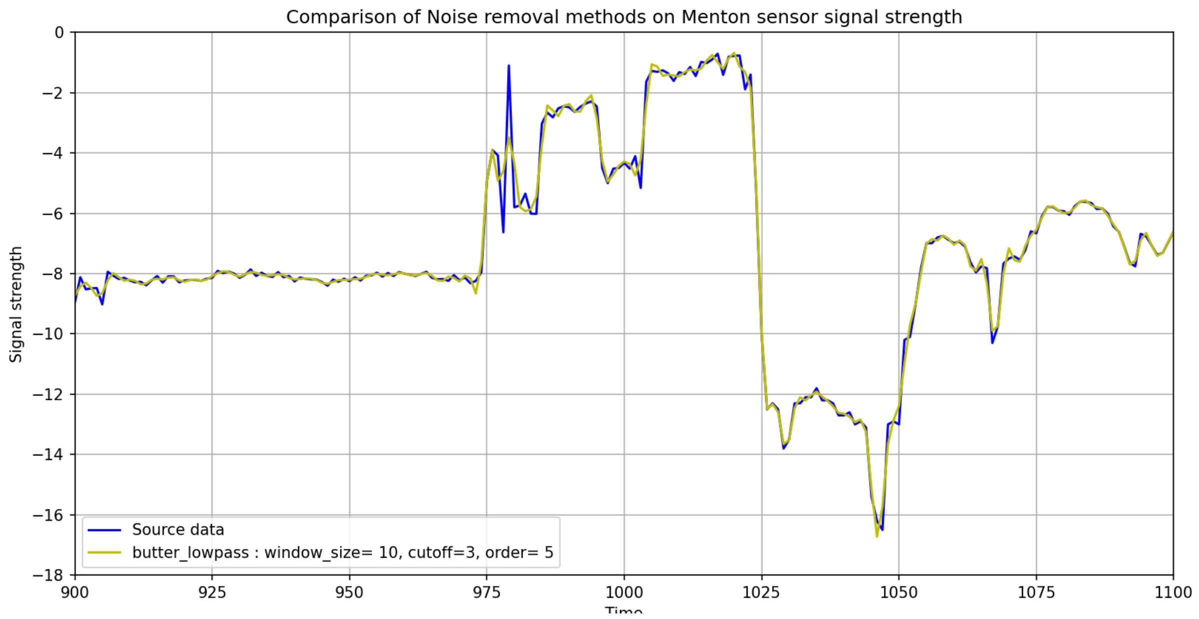


Figure 16 Butter low pass applied on Menton signal

The different generic algorithms tested before are available on GitHub SEDIMARK repository for noise cancellation [119]. The code is in python for easy integration in SEDIMARK data pipeline. They also support being integrated in data streams using only the latest values to compute the new value.

3.7.3 Deduplication

Real world datasets can often include a large number of duplicate records, for instance due to incorrect data entry or errors during the data collection process. This presents a pernicious problem for data scientists seeking to model the underlying data distribution, as the duplicates can often be hard to detect, and can often even require expert evaluation, while skewing statistics and causing problems for the evaluation of downstream tasks. Broadly speaking, the task of data deduplication aims to match pairs of records that are likely to be duplicates. Generally, this is accomplished by introducing some notion of similarity between record pairs, either by utilising a set of users specified rules, or with e.g. an ML classifier. The records can then be clustered according to this similarity metric. Data deduplication is generally considered to be a demanding computational problem, given the naive need to compare every pair of records within the dataset with each other. Also, often there is only a very small amount of labelled 'training data' to build an ML approach. More advanced recent approaches, such as ActiveClean [44] seek to involve human input in guiding an ML process, by for instance having an algorithm iteratively request labels for the samples about which it is least certain.

In its current state, SEDIMARK makes use of the record-linkage python library [45], which relies upon user specified rules for data deduplication. These include e.g. setting specific metrics for string similarity (e.g. Jaro-Winkler [125], [126] or Levenshtein distance [127]) and allow the computing of how similar two records are to one another across a number of fields, with a threshold then used to determine whether the records are duplicates or not. A downside of this process is that it requires at least some human assessment of the dataset, and the choice of which fields are likely to be worth inspecting for duplicate comparisons.

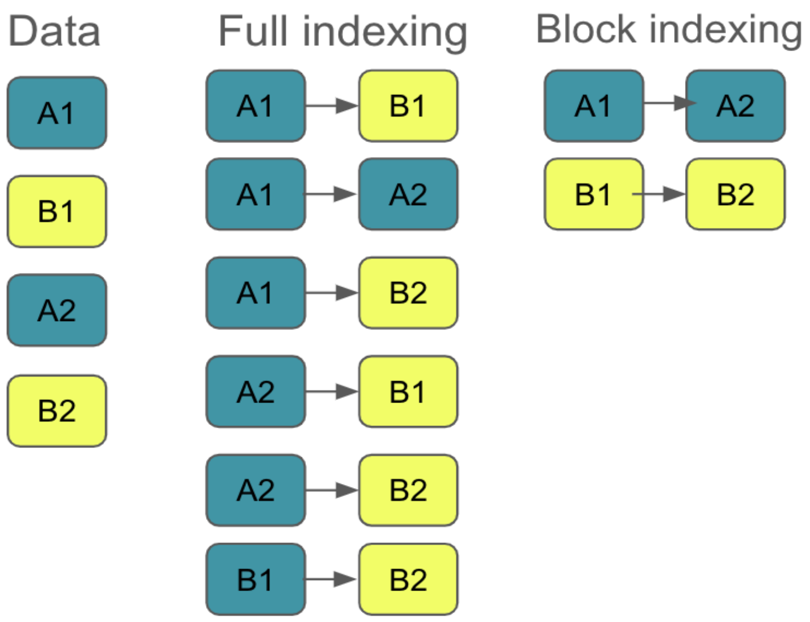


Figure 17: Comparison of full and block indexing methods.

As mentioned previously, one factor that significantly affects the performance and efficiency of a data deduplication algorithm is the indexing method used. In a naive 'full' implementation, all records within the dataset/database are compared against one another, which makes the complexity O(n^2), but this can be greatly reduced by the choice of an appropriate heuristic and method for retrieving likely pairs. The *record-linkage* library [45] allows for the use of Block indexing - only comparing blocks of records that share a certain attribute in common, and the additional SortedNeighbourhood indexing method, which allows the blocks to also include records within e.g. a short edit distance. As shown in section 5.3.2 indexing methods greatly increase the speed of the procedure, but with a resulting loss in terms of recall, as less potential matches are considered. A comparison of full and block indexing methods is given in Figure 17. The raw data is shown on the left, with letter and number representing two features. In the middle, n^2 comparisons are made between the data points. On the right, as the data points are indexed by letter, much less comparisons are made.

Recent research has shown that large language models (LLMs) can be highly effective out of the box for deduplication and the related task of entity matching, often surpassing state of the art benchmark results despite not seeing any task specific training data [81][82]. As such, they can be considered as something of an AutoML approach to deduplication, as they remove the necessity for data scientists to provide labelled examples. Despite this, their use can often incur a prohibitively high cost, especially for the most state of the art models. To this end, the SEDIMARK deduplication module extends the popular Dedupe [83] package to integrate the use of LLMs. Whereas Dedupe employs active learning to iteratively query users for labels for uncertain data points, the SEDIMARK deduplication module instead roots these queries to a user specified choice of LLM. In such a way, the labelling cost of training data is greatly reduced. Once its internal ML model has been fitted to these labelled data points, Dedupe then predicts duplicates for the rest of a dataset.

3.7.4 Missing Value Imputation

There are numerous reasons why missing values, or incomplete records, may appear in a dataset, ranging from problems with the database that stores them, to network issues or faulty data collection or entry. These missing values pose a severe problem to downstream ML algorithms, which struggle to effectively handle "NaN" entries. While, in the case that the number of missing values is small, the incomplete entries can simply be removed, above a certain threshold this will impact the performance of any model trained on the resulting data [46]. Thus, a popular area of research considers the correct method for imputing or replacing the missing entries with their likely real values. Numerous methods exist for missing value imputation, targeting multiple different types of data. They can generally be classified as either statistical (e.g. Mean/Mode imputation, Linear/Logistic regression, or singular value decomposition) or ML based techniques [46]. In general, ML based missing value imputation methods will treat imputation as a classification or regression problem, attempting to predict the missing values of a feature by modelling them based on other features in the dataset. A generic functionality of a Value Imputation Module is shown in Figure 18, where the module fills the gaps that it identifies in the dataset.

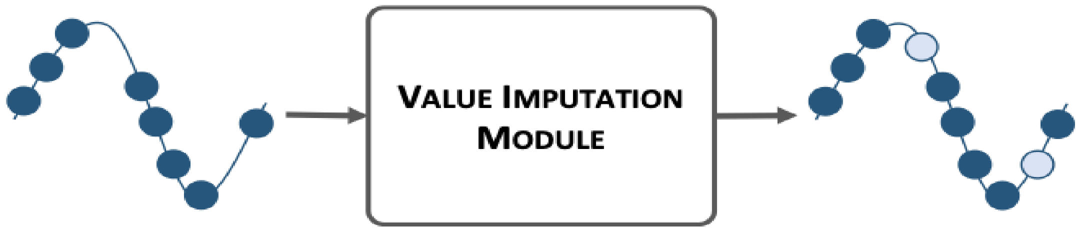


Figure 18: Module for imputing missing values.

3.7.4.1 Offline data

When it comes to imputing values, methods such as interpolation of Artificial Intelligence techniques can be employed. Interpolation involves fitting known data to a function that allows the estimation of missing data. On the other hand, AI-based methods estimate missing values by leveraging available information, often using supervised learning approaches. In particular, the k-Nearest Neighbour (kNN) algorithm is widely used for this purpose. It involves classifying data values into clusters or categories based on their proximity to their *k* nearest neighbours. Another popular ML based approach to missing value imputation for tabular data is iterative imputation, whereby features are imputed one at a time using the other features in the dataset as regressors, with the process iterated upon until some stopping criteria (such as accuracy) is met. However, in its naive implementation iterative imputing is limited to one class of regressor model, which might not necessarily be optimal for all of the features in the dataset [47]. Other approaches exploit regularities of their particular domain to infer missing values, such as for instance in the case of time series data where filling gaps by interpolating between their neighbouring timesteps can often be quite effective.

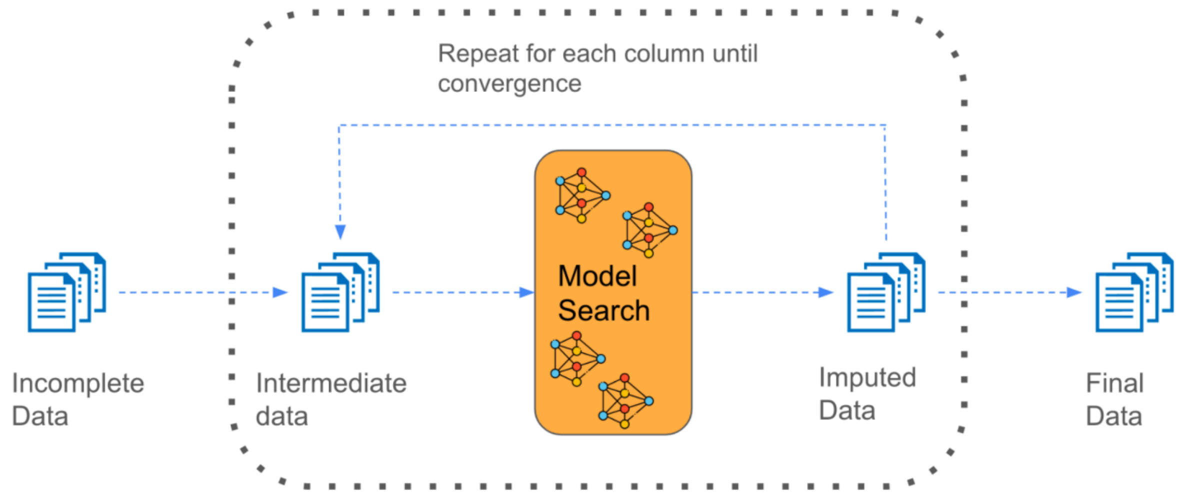


Figure 19: Iterative imputation (HyperImpute).

In SEDIMARK, the focus is on imputing missing values for both tabular and time series data. SEDIMARK includes interpolation for imputing gaps in time series and leverages the impute module of the scikit-learn python library to offer KNN and iterative imputation methods. At its current stage of development, SEDIMARK also includes the HyperImpute algorithm [47] (Figure 19), a method that efficiently iterates through rounds of model selection for each feature column in a dataset. As such, it doesn't require the user to go through the process of choosing the appropriate regressor for each feature in their dataset, and this can in some sense be considered an "AutoML" solution to the problem of model selection for missing value imputation.

3.7.4.2 Streaming data

Similar to streaming outlier detection, SEDIMARK utilises River's functionalities for effective missing value imputation. The PreviousImputer replaces missing values with the last observed value, suitable for time-series streaming data, while the StatImputer uses statistical measures (mean, median, mode) for imputation, effective in datasets with randomly distributed missing values. These methods collectively enhance SEDIMARK's ability to maintain high data quality standards in its streaming data marketplace. By integrating River's capabilities, SEDIMARK not only ensures the integrity and reliability of its data but also streamlines the data cleaning process, significantly reducing the need for extensive manual intervention.

Moreover, the EWMA algorithm that has been developed and presented in Section 3.7.1.2 have been adapted to update datastreams with missing data points by applying the inliner model and including an inliner datapoint when a gap is detected.

3.8 Data Augmentation

3.8.1 Overview

Data augmentation is a popular technique that can serve multiple purposes within a machine learning pipeline. Techniques such as interpolation can be used in time series data to upsample or fill in missing gaps in the series. If given sufficient prior domain knowledge, data augmentation techniques can be used to increase the volume of the training data, resulting in the training of more robust models, with heightened model accuracy. This is especially effective in the image processing/computer vision domain, where it is relatively easy to produce additional images that intuitively fit a given class label, using simple transformations like rotations, or adding noise or colour to the raw image. More recently, the model-based generation of synthetic data has been pursued as a method to solve certain data problems. On the one hand, a data provider might not want to grant access to their raw dataset, and in this instance, it might be preferable to instead release synthetic data, on which an equivalent machine learning model can still be learned. In the other instance, synthetic data might be used to rebalance a dataset, especially in the instance where it perhaps presents a harmful bias against one particular class. SEDIMARK will include data augmentation tools to address all of these distinct scenarios - augmentation/generation for increasing the performance of downstream ML models, augmentation for debiasing and rebalancing datasets, and synthetic data generation for replacing the underlying dataset completely.

3.8.2 Synthetic Data

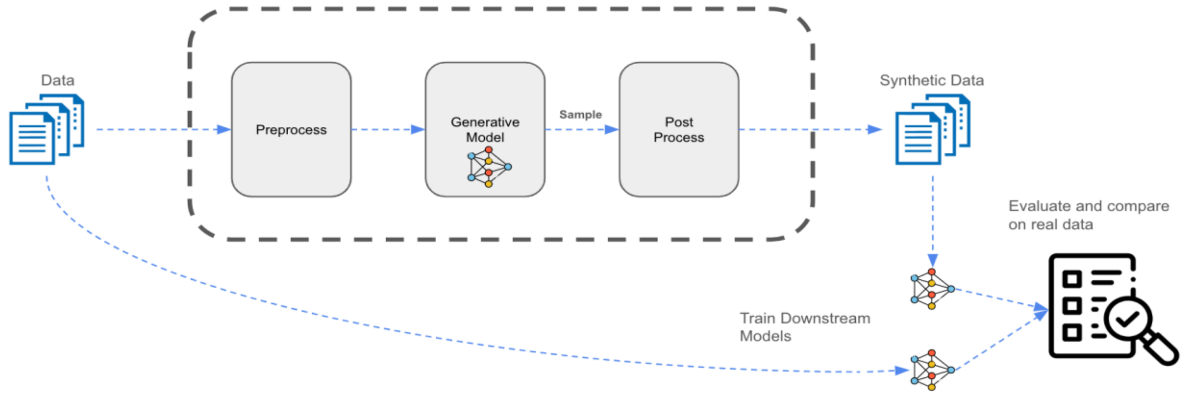


Figure 20: Synthetic data generation in SEDIMARK

A data provider might not wish to release their underlying dataset, but instead opt to make available a synthetic alternative. The rise of deep learning has ushered in huge advances in the modelling and then generation of synthetic data. Both Variational AutoEncoders (VAEs) and Generative Adversarial Networks (GANs) are deep learning based methods that aim to learn and then sample from the underlying distribution of the data. In the case of VAEs, this is accomplished by training an autoencoder model to encode and reconstruct data samples, with the use of a 'reparameterization' trick [48] allowing for the drawing of samples from a learned gaussian latent space. In the case of GANs [49], two model components, a generator and a discriminator play an alternating min-max game, in which the discriminator learns to separate generated samples from true ones, while the generator aims to fool the discriminator into classifying its samples as coming from the real data distribution. While the true 'likeness' between true and generated data is hard to evaluate when the underlying distribution is not known, for practical purposes the fidelity of the generated synthetic data can be evaluated intuitively, through comparing the performance of downstream machine learning models trained on either the original or the synthetic data, as shown in Figure 20.

SEDIMARK will include tools allowing data providers to generate synthetic data. In its present iteration, SEDIMARK makes use of methods present in the python library YData-Synthetic to generate synthetic tabular data. This tool allows the user to select from a number of GAN based models such as CTGAN [50] for tabular data generation. In addition, it includes a less computationally demanding method (Gaussian Mixture Model) which represents the data distribution as a mixture of gaussians from which new samples can be drawn, with the implementation iteratively testing an increasing number of gaussian components to find the best fit.

However, it is expected that much of the data processed by SEDIMARK will be of a time series nature, where tabular methods will not work out of the box, as they generate Independent and Identically Distributed (IID) samples without considering the time sequential dependency between data points. In the case of time series generation, the user will be looking to sample whole trajectories of data points. The process often suffers from the low number of distinct trajectory examples that are available in the dataset to be modelled, compared to other generative fields such as text or image generation, where there are huge public datasets available allowing for the creation of general purpose generative models. Luckily, there has been a significant amount of recent interest in the problem of time series generation, with several methods published in high profile machine learning conferences.

The most recent version of the SEDIMARK data curation pipeline integrates the python library TSGM [84], which features various GANs and VAEs that can be readily applied to time series. These models generally run general generative approaches for synthetic data, but extend them to generating sequences/trajectories rather than just singular data samples, thus ensuring that time dependency is preserved in the generated sequences. We provide some experimental results, displaying this tool's computational energy efficiency in section 5.3.6.

3.8.3 Augmentation for balancing and debiasing datasets

It is widely acknowledged that issues with data collection can inject harmful biases into downstream machine learning models, for instance when data points corresponding to at risk minorities are underrepresented in the training data. Additionally, datasets may be generally unbalanced, with some minority classes having few examples present in training data. A further use of data augmentation considers the problem of debiasing or balancing these datasets. A general method for this is to employ one of the synthetic data generation methods described in the previous example, but to reweight its training data in favour of the minority classes one wishes to remove bias against. The current version of the SEDIMARK data augmentation module integrates the popular tool TabFairGAN [51]. TabFairGAN employs two phases of GAN based training. In the first regular GAN training phase, the model is trained to match the regular distribution of the training data. In the second phase, a fairness constraint is added to the loss function, penalising the model for generating unfair samples. In this way it can be used to generate synthetic samples that can then be used to debias a regular dataset.

3.9 Feature Engineering

Feature Engineering is one of the principal components in the SEDIMARK AI pipeline. It refers to the pre-processing steps that select and transform the most relevant features, aka attributes, from raw data, into features to be used in machine learning algorithms, such as predictive models. Whence the goal of simplifying and speeding up data transformations while enhancing model accuracy by keeping the most relevant attributes. Dimension reduction tackles the curse of dimensionality that occurs due to the use of high-dimensional data which may increase the cost of any mining algorithm and consists of mapping high-dimensional instances onto a lower-dimensional representation while conserving the distances between instances. Dimension reduction within SEDIMARK could also be used in non-high-dimensional spaces to represent the data more effectively by compacting them differently.

Two main different dimensionality reduction categories are considered within SEDIMARK (see Figure 21 that depicts the difference between them):

- **Feature selection** is the process of selecting a subset of the input features, i.e., the most relevant and non-redundant features, without operating any sort of data transformation or extraction. To do so, feature selection techniques mainly analyse and rank various features to determine which ones are irrelevant and should be removed, which ones are redundant and/or correlated, and which ones are most relevant and useful for the model and should be prioritized.
- **Feature extraction** that involves reducing the number of features to be processed using dimensionality reduction techniques. It consists of extracting features from a dataset or data stream to identify useful information. Without distorting the original feature space, this compresses the set of features into manageable quantities for algorithms to process. E.g., constructing from a set of input features in a high-dimensional space, a new set of features in a lower-dimensional space.

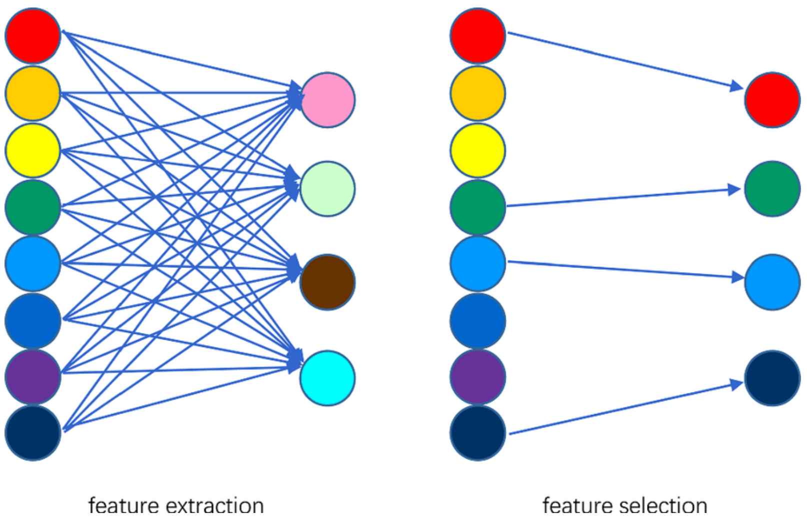


Figure 21: Feature extraction vs. feature selection

Dimension reduction techniques are widely used in machine learning algorithms and operate by transforming and using the most relevant feature combinations, handling thus the curse of dimensionality, in turn reducing space and time demands; this can be crucial for applications such as classification and visualization. For this to happen, SEDIMARK implements and uses some common dimension reduction techniques from the state-of-art and available in python libraries, such as PCA, feature hashing, correlation, and random projection. These techniques require an input parameter specifying the size of the output space of data dimensionality.

Several dimension reduction techniques have been proposed and used for offline purposes, namely for static datasets. Hence, they cannot be used with data streams and have to be adapted. For this to happen, multiple static techniques have been extended to the stream setting. For instance, an extension of SEDIMARK could use an incremental version of PCA, available in the scikit-learn python library, called IncrementalPCA which uses a window to make PCA incremental, hashing trick, and random projection. Thanks to the fact that they are data-independent techniques, the extension of the latter to the stream setting is easy. In the SEDIMARK data processing and AI pipelines, dimension reduction is provided as a component, and its techniques can be extended and used with batch and stream data.

AutoML could be used within SEDIMARK to facilitate the task and serve the purpose of automatic hyperparameter optimization without the need to precisely specify which technique to use with what parameter.

3.10 Auto-ML in the data processing pipeline

Machine learning algorithms have multiple hyper-parameters that are set prior to the learning process and can directly affect the performance of the models. However, this task requires domain knowledge and human expertise to perform manual hyper-parameter tuning (manually trying out different hyper-parameter sets by hand), which is a hard and tedious task both for expert and non-expert users. Another major drawback of the manual search is the fact that the process is not easily reproducible. The latter is definitely important, especially for the progress and improvement of scientific research in the machine learning field and for non-experts who intend to use ML. Moreover, it is difficult to manage the manual hyper-parameter tuning task when the number of parameters and the range of values is high, i.e., if one aims to apply an algorithm with five hyper-parameters, then these hyper-parameters need to be manually tuned with different values on different datasets, and the best combination that achieves the highest performance will be chosen.

To cope with the aforementioned issues, automatic search techniques have been proposed under the emerging automated Machine Learning (AutoML) topic. AutoML supports researchers and practitioners with the tedious work of manually designing ML and AI pipelines, which include performing algorithm selection and tuning hyper-parameters.

A possible extension of the SEDIMARK data processing pipeline might use AutoML to automate various tasks, mostly related with the configuration of the models that are being used by the components of the pipeline. As discussed in the previous sections, users of the data processing pipeline will be presented with many options for processing their data. One can imagine that non-expert users will easily be confused regarding which component they have to use, which model/algorithm of that component and which configuration of the hyperparameters of that model/algorithm in order to best process their dataset. Manual testing can take time, so SEDIMARK aims to provide the tools to automate the process of selecting the best-performing model and identifying its corresponding optimal set of hyper-parameters that best fit each dataset and each user preferences.

Users of the data processing pipeline will be provided with AutoML tools which will make the use of the data processing components both easier and more accessible for SEDIMARK participants. With the integration of AutoML into SEDIMARK, the next integration steps could involve enhancing other aspects of the data processing pipeline to fully leverage the potential of automated model selection and tuning relative to the data fed to the platform. This holistic approach will ensure that each stage of the pipeline is optimized for efficiency and effectiveness.

One of the key aspects is that data pre-processing and feature engineering to handle diverse data types and structures will be executed more effectively. Expanding the range of explored models through AutoML and automating model validation will also be an effect of choosing AutoML. Creating intuitive interfaces for interacting with the AutoML system and providing educational resources to assist users of varying expertise levels will also be a benefit of this choice.

Currently, we have developed an AutoML approach for data stream learning. The few existing AutoML solutions for stream learning mainly rely on random search or genetic algorithms, which struggle to maintain high performance in dynamic environments. By contrast, leading methods in batch learning such as the Sequential Model-based Algorithm Configuration (SMAC) [107] leverage model-based approaches, suggesting opportunities for improvement in stream settings.

In an attempt at filling this gap, we introduce OnlineSMAC, a model-based optimizer for data streams. OnlineSMAC combines Bayesian optimization with an extension of the SMAC optimizer to dynamically select optimal processing pipelines and hyperparameters.

The basic principle of Bayesian optimisation is to hold an incumbent model (the current best one) and generate new configurations to try to overcome the incumbent. The new configurations are generated according to a surrogate model, a model that estimates the behaviour of the function we are optimising. The surrogate model will pick configurations it estimates they will have a chance of being better than the incumbent or will explore a new part of the domain.

In OSMAC, we split the stream in a succession of 1000-instance long windows. At the start of a window, we use the surrogate model to generate 3 new configurations and add 2 fully random ones and a copy of the incumbent. We train these 6 candidate configurations over the duration of the window. When the window ends, we compare the performance of the candidates and the best one becomes the new incumbent for the next window.

This setup provides an efficient optimiser for AutoML on data streams, showing that Bayesian optimisation is possible and useful beyond batch AutoML.

Through this current and future efforts, SEDIMARK is set to offer a comprehensive, automated, and user-friendly Data Processing Pipeline, enhancing the efficiency and applicability of machine learning models across various user groups and applications.

4 Techniques for reducing energy consumption of data technologies

4.1 Overview

The environmental impact of computing technologies has become an increasing concern, particularly in big data processing and machine learning, where computationally intensive processes consume vast amounts of energy and generate significant carbon emissions. As state-of-the-art ML models grow exponentially in size, their demand for computational power scales accordingly, amplifying their carbon footprint and increasing the environmental cost associated with data storage and transfer. In response to these challenges, recent research has focused on optimising ML model training and inference to improve energy efficiency while maintaining performance. SEDIMARK aligns with these efforts by implementing strategies to minimise environmental impact of its ML processes. This chapter outlines several approaches that SEDIMARK will adopt to enhance energy efficiency in ML and data management based on what performed in the first version of the deliverable and the work so far towards this direction. Section 4.2 explores strategies for reducing energy consumption during ML training, including coreset selection, data distillation and dimension reduction. Section 4.3 presents techniques to optimize model efficiency, such as model compression, quantisation, pruning and low-rank factorisation. Section 4.4 examines how dimension reduction can lower the environmental costs of data sharing. Section 4.5 addresses energy-efficient data storage solutions. Section 4.6 develops the concept of a lightweight SEDIMARK toolbox designed for edge device compatibility.

4.2 Reducing energy consumption during ML training

4.2.1 Optimising Data Efficiency in ML Training

Training modern ML models requires substantial computational resources, often involving repeated processing of large datasets using power-intensive hardware such as GPUs and TPUs. This escalation in energy use has multiple implications:

- Increased energy consumption makes ML training financially prohibitive for smaller organizations posing high operational costs.
- Environmental constraints as ML-driven energy demand raises carbon emissions, particularly when relying on non-renewable energy sources.
- High power usage can stress existing energy grids, challenging the long-term sustainability of AI development causing infrastructure problems.

To mitigate these effects, SEDIMARK prioritizes improving data efficiency during ML training. By reducing the volume of data required for model development, the framework lowers computational costs while maintaining model accuracy. This approach is critical for minimizing the environmental footprint of ML workflows.

This section highlights three key techniques for optimizing data efficiency in ML training; coreset selection (reducing dataset size while preserving essential information), data distillation (compressing knowledge from large datasets into smaller, more efficient forms), and dimensionality reduction (eliminating redundant features to minimise computational overhead).

4.2.1.1 Coreset Selection

Coreset selection [57],[58] is a compression technique that selects a smaller representative subset of data from a larger dataset while preserving its key properties. This enables machine learning models to be trained more efficiently, using less data and computational resources, while still aiming to preserve the quality and accuracy of the models trained on the complete data. Coreset selection is particularly valuable in scenarios where computational efficiency is crucial and when dealing with massive datasets that are costly or impractical to use in their entirety for training purposes. In the context of SEDIMARK, coreset selection could be applied in several innovative ways to enhance its functionality and efficiency:

- **Distributed Training Efficiency**: SEDIMARK can utilise coresets to facilitate efficient distributed training across its decentralised network. By allowing each node to hold a coreset instead of the full dataset, the network can minimise the data transmission overhead and reduce training times, leading to significant energy savings.
- **Bandwidth Optimization**: When nodes need to communicate or synchronize datasets, transmitting coresets instead of full datasets can reduce the required bandwidth and enable faster, more efficient data sharing.
- **Resource-Constrained Environments**: For nodes operating in environments with limited computational resources, such as IoT devices or mobile phones, coresets can enable these devices to participate in the ML training process without the need for high computational power.
- **Rapid Prototyping**: When developing new models or experimenting with different algorithms, SEDIMARK users could employ coresets for rapid prototyping, allowing for quick iterations over model design with less computational cost.

4.2.1.2 Data Distillation

Different from coreset selection, data distillation [59],[60] in machine learning involves significantly reducing the size of a dataset by creating synthetic data that captures the essential information of the original data. This process allows the training and tuning of machine learning algorithms efficiently, akin to using the complete dataset. In SEDIMARK, in addition to achieving energy reductions akin to those from coreset selection, data distillation can also be leveraged to enhance its privacy protection capabilities in several ways:

- **Privacy-Preserving Data Sharing**: By distilling data to only its essential features, SEDIMARK can share insightful synthetic data across its network without exposing the raw data, which might contain sensitive user information.
- **Anonymization**: Distilled datasets can be designed to exclude personally identifiable information, allowing the platform to utilize and share data while adhering to privacy regulations and user consent.
- **Reduced Data Footprint**: A smaller, distilled dataset minimizes the risk of data breaches, as less real information is stored and transmitted, reducing the exposure of user data.

4.2.2 Balancing the model training cost and the communication overhead

In decentralised ML three main costs are associated with the learning process: (i) the communication cost, (ii) the cost of training time and (iii) the final model accuracy. Achieving high accuracy while keeping the cost of training time low in decentralised ML often incurs high communication costs. On the other hand, decreasing the communication costs while maintaining high accuracy leads to high model training costs, since with sparse communication between the working nodes it takes longer for the decentralised ML model to converge. The goal of SEDIMARK is to develop energy-efficient ML solutions, therefore the ML models implemented within the project aim to find a balance between high accuracy and low communication and training time overhead. Inspired by [61], in decentralised ML settings, two distinct contexts can be distinguished:

- **Weakly coordinated learning (WCL)**, where the amount of coordination between computing nodes is sparse. In this setting, some global coordination is still required. Computing nodes must agree on the hyperparameters of the learning algorithm, such as the number of latent factors, and must have commonly agreed identifiers for the items in the product database.
- **Strongly coordinated learning (SCL)**, where each computing node knows the number of cooperating computing nodes and global synchronisation is possible, with any pair of computing nodes able to send and receive parameters and coordination data between each other.

One of the main advantages of SCL is that it is closely following the implementation of a centralised ML algorithm and therefore, it generally follows the accuracy of the centralised algorithm. However, this comes at the expense of a heavy communication overhead, requiring *n* all-to-all synchronisations per epoch (*n* being the number of nodes participating in the learning process) to ensure that computing nodes keep their model parameters in sync. Note that originally this approach was developed to suit large supercomputers, where it is possible to control the overall number of workers and balance the data distribution in order to find the best trade-off between the level of parallelisation (i.e. the overall number of computing nodes involved) and the communication overhead. However, in SEDIMARK, the data is already pre-allocated, and the overall number of participating computing nodes cannot be easily controlled. The high communication cost can have a significant impact on the efficiency of the distributed computation.

In contrast, WCL provides a communication-efficient alternative. The main advantage of this approach is that a computing node communicates only with a small subset of other computing nodes during the exchange of the model data. However, this reduced communication can mean that the global ML algorithm needs a substantial number of epochs to converge since this approach is inherently slower at distributing the contribution of training data to the model across the entire network of participating computing nodes. SEDIMARK aims to develop an efficient decentralised ML interface, focusing on minimising the communication overhead and minimising the training time of the global model. Therefore, SEDIMARK aims to develop a hybrid approach combining weak and strong coordination. As demonstrated in [61], the communication-intensive strongly coordinated algorithm can be used to boost the model parameters into a part of the solution space in which the model can converge to a good solution. The remaining convergence steps can then be computed by the weakly coordinated approach, reducing the high communication overhead in the later stages of learning when it is no longer necessary.

The goal of this hybrid approach is two-fold: (i) minimise the communication and model training costs while simultaneously (ii) maximise the model accuracy. As discussed above, SCL will address the model's training costs, while WCL will keep the communication overhead at a minimum.

One of the main reasons for the slow training time of WCL is the fact that all computing nodes start from random model parameters. This, together with sparse communication of the algorithm, results in longer training times as the algorithm requires some time to distribute the data across the entire network.

For simplicity and for proof-of-concept scenarios SEDIMARK has opted for using the SCL, assuming that the number of cooperating computing nodes is known beforehand. This is realised within SEDIMARK because all participating nodes are using the marketplace to get the model assets and thus the initiators of the decentralised ML process can know at any given point in time which participants are members of the process.

4.3 AI Model Optimisation

The accuracy of today's machine learning models has significantly improved over the past decade. However, this improvement comes with the price of massive, over-parametrised models causing high energy consumption and unacceptable delays in inference which is often performed in real-time. To address the above issues a large body of research has formed with the main goal of improving the efficiency of machine learning models while also maintaining optimal accuracy. Taking inspiration from [62], SEDIMARK focuses on reducing the memory footprint and computational cost of the machine learning models and hence focuses on five main approaches for model compression:

- **Lossless compression -** methods that exploit statistical redundancy in the data to compress the size of the model without error.
- **Quantisation** - the goal of this approach is to decrease the size of model weights. Broadly speaking, this is achieved by reducing the precision of model weights from the high-precision floating point representation to the low-precision floating point or integer representation. This achieves a compact model representation.
- **Pruning** - the focus of this approach is to remove neurons with small saliency (sensitivity) from the neural network. This achieves a sparse computational graph with a smaller memory footprint. The approach can be further structured into:
    - Unstructured pruning, which removes all salient neurons achieving aggressive pruning. However, this leads to sparse matrix operations which can be hard to accelerate.
    - Structured pruning focusing on removing groups of parameters. On the plus side, aggressive unstructured pruning still allows for dense matrix representation but can lead to a significant loss of model accuracy.
- **Knowledge distillation** - this approach uses the large model as an input in the AI algorithm to produce a smaller, more compact model.
- **Low-rank factorisation** - these techniques use factorisation to represent the weight matrix with a product of two smaller matrices.
- **Mixed Precision Training and Inference** - Mixed precision is an optimisation technique that uses lower numerical precision such as 16-bit or 8-bit formats instead of standard 32-bit floating point for model training and inference. This approach reduces memory usage and computational load while maintaining comparable model accuracy. It is supported by

most modern hardware accelerators and is commonly used to improve execution speed and energy efficiency across a range of machine learning applications. It offers a balanced trade-off between efficiency and accuracy and is commonly used as a standard configuration for benchmarking other optimisation methods such as quantisation or low-rank adaptation.

Different model compression techniques can be applied to different ML models. For instance, Deep neural networks can be compressed using all compression methods listed above. In contrast, more traditional machine learning models, such as decision trees or random forests, can be compressed using pruning or quantisation techniques [63]. SEDIMARK aims to develop a small number of approaches covering all four compression methods. This will allow the potential user to choose the compression technique best suited to their model.

4.3.1 Lossless compression in federated learning

Lossless compression techniques can be applied in federated learning (FL) to reduce the size of exchanged model parameters without affecting model accuracy. By compressing local updates before transmission, communication efficiency can be improved while preserving the integrity of the training process.

In typical FL setups, multiple edge devices train models locally on sensitive data and send the resulting updates to a central server for aggregation. However, the repeated exchange of large model parameters introduces significant communication overhead and energy consumption—especially in vanilla FL topologies, where the central server handles all aggregation. Efficient compression methods are therefore essential to mitigate these bottlenecks.

4.3.1.1 Related work

Lossless compression is a type of data compression that exploits statistical redundancy in the data to reduce its size without any information loss. It is mainly used in applications where preserving the data's integrity is required or highly desirable (such as in text documents or executable files). In contrast, lossy compression is that in which the uncompressed data is not identical to the original, which usually comes as a trade-off between the accuracy or fidelity of the data and the achieved compression rates, allowing for significantly smaller file sizes at the cost of some loss in quality or precision.

Some common lossless compression methods are: substitution coders [100]-also known as dictionary coders-, where the data is converted to text strings that are then substituted by the index in a dictionary; entropy coding [101], which encodes common sequences in data using fewer bits than less frequent or rare sequences; or run-length encoding [102], which compresses consecutive occurrences of the same value by representing them as a single value followed by the number of repetitions. Although general purpose encoders are common and widely used, most techniques are optimized for a particular type of data (images, text, audio, etc.) to better exploit its statistical properties. Many algorithms combine two or more techniques to further increase the compression ratio they can achieve. Some of the most extended are:

- **Huffman coding,** which assigns variable-length codes to the input symbols, with frequent symbols being encoded using shorter codes.
- **Lempel-Ziv compression,** which identifies repeated sequences in the data with a sliding window and dynamically builds a dictionary of variable size. The two main algorithms from

this family are LZ77 and LZ78 [103], which serve as basis for other variants such as LZMA (Lempel-Ziv-Markov) or LZW (Lempel-Ziv-Welch).

- **Deflate:** [104] a widespread algorithm that combines LZ77 and Huffman coding.
- **Asymmetric numeral systems:** [105] a family of entropy coding algorithms that encodes the information as natural numbers.
- **Burrows-Wheeler transform,** [106] which reorders the input data to achieve longer sequences of repeated symbols before applying run-length encoding techniques.

These algorithms have been implemented in several software that are widely used for many applications. For example, GZIP, Zlib, Brotli, Zstd, Zopfli, Bzip2, Snappy or LZ4, to name a few. Although modern algorithms are high performant, lossless compression techniques are limited by the statistical properties of the input data, as data with no redundancy cannot be compressed.

In the context of machine learning, deep learning models typically present high entropy, consisting of large set of floating-point numbers with a seemingly random distribution. Since data with high entropy is difficult to compress, lossless methods usually fail to achieve high compression ratios. As a conclusion, lossless compression should only be used in applications where achieving the highest accuracy is prioritized over communication costs and storage size.

4.3.1.2 The Fleviden compression solution

In Federated Learning, inter-agent communication is one of the most critical steps for the training to be carried out successfully. Since bandwidth is limited, efficient communication is required to avoid bottlenecks.

Fleviden [5] offers a lossless compression solution to optimize the size of the exchanged messages in use cases where model integrity and high accuracy are desirable. As Fleviden messages contain more than just the model's updates, the solution is agnostic to the type of input data contained in the messages (formatted as JSON-serializable dictionaries). The solution is implemented as a package of compression pods that encode and decode Python dictionaries efficiently, with several available options that can be tuned to the case at hand. To comply with the JSON-serializable requirement of all Fleviden modules, these pods perform the following steps:

- **Binarization,** where dictionaries are converted to byte objects. Two binary serialization formats are implemented, namely, MessagePack [129] and CBOR2 (Concise Binary Object Representation [128]). The first is slightly faster and more compact, but the latter supports advanced data types, which expands on the agnosticism of Fleviden's compressor.
- **Compression,** applying one of the algorithms available as Python packages, such as Zstd (Zstandard) or LZMA. Zstd is faster and the default choice for most use cases. LZMA is more computationally expensive but achieves greater compression ratios. A trade-off must be made between speed and ratio, so users must select one or the other according to their needs.
- **String encoding.** Because byte objects are not JSON-serializable, they must be encoded as strings before generating the final output. A base64 encoding block is used for this task, as it provides the best possible code with printable ASCII characters.

The steps are executed in reverse order for the decoding and decompression of previously encoded data. For the decoding pipeline to work as expected, the same compressor and binary serialization format must be agreed upon. To enable interoperability with other pods, the output message is another Python dictionary that contains the compressed and encoded information, as well as additional metadata about the process, such as the achieved ratio. Because the model is reconstructed without error, its performance is not compromised in any way. Therefore, the only trade-offs are related to the cost in energy and time of the algorithms themselves.

In a typical federated learning scenario, participants may use the compression pod as the first contact layer between inter-agent communication, or to store models in a more compact file. The attached metadata of the pod can be used to check that the decompression process was successful or to gather compression metrics from different clients.

4.3.2 Quantisation in federated learning

To address the communication efficiency challenge, quantization techniques have been implemented to compress the size of messages exchanged between the clients and the server during the FL process. These techniques fall under the category of lossy compression, enabling a significant reduction in update sizes without the need to fully preserve the original data, while still ensuring convergence of the global model.

In this context, the Quantized Stochastic Gradient Descent (QSGD) algorithm [130] was integrated as a quantization method, due to its proven convergence guarantees for both convex and non-convex optimization problems. QSGD transforms model parameters into discrete levels controlled by a configurable parameter (s), followed by encoding them using Elias Omega coding, a compression scheme that efficiently represents integer values—especially when smaller values are more common, which is often the case in gradient updates. Together, these mechanisms enable substantial communication savings with minimal impact on model accuracy. This quantization strategy was implemented within the Fleviden federated learning framework through the development of two new pods:

- The **qsgd** pod, which performs the quantization and dequantization of model parameters according to the QSGD scheme.
- The **elias** pod, which encodes and decodes the quantized values using the Elias Omega method.

These pods can be flexibly connected to other components such as the server, agents, and aggregator modules. For instance, when the server receives updates from clients, they are first passed through the **elias** pod for decoding, then through the **qsgd** pod to reconstruct the approximate floating-point values before aggregation. The reverse process is used when sending updates back to the clients.

A detailed description of the algorithms, including pseudocode, design details, and software architecture diagrams, can be found in SEDIMARK_D3.1 [104]. Interested readers are referred to that document for an in-depth technical overview of the quantized federated learning solution and its integration within Fleviden.

4.3.3 Pruning

The goal of pruning techniques is similar to the goal of quantisation and that is to produce more compact models with a reduced memory footprint. However, in this case, rather than compressing the model weights, pruning removes the redundant parameters or neurons. This happens when the weight coefficient is replicated or close to zero (0) or zero (0). The pruning techniques can be categorised in various ways [64]:

- Based on the symmetry of the pruned network into symmetric and asymmetric methods
- Based on whether the pruning is performed after or during training into static or dynamic pruning

Various elements can be pruned in deep neural networks [65] as follows:

- **Weights** - refers to removing the network weights based on some condition, such as for example weights below some pre-defined threshold. This requires traversing weights one by one, which can affect the running time.
- **Neurons** - different from removing weights, this method refers to removing the redundant neurons.
- **Filters** - when removing filters, these are first ranked according to their importance based on a pre-defined metric and then the least important filters can be removed.
- **Layers** - another pruning approach looks at removing entire layers from deep networks.

SEDIMARK will provide a suite of pruning approaches based on the user needs. For example, for users who wish to save energy during the training process, SEDIMARK will offer dynamic pruning methods. On the other hand, for users who wish to reduce the memory and inference cost of an existing model, a static pruning strategy might be more suitable. Similarly, SEDIMARK will allow the user to choose the specific model elements they wish to prune and will suggest suitable pruning approaches accordingly.

4.3.4 Knowledge distillation

Knowledge distillation consists of two models, the large original model, which is also referred to as the teacher model and a smaller representation of this larger model called the student model. The goal of this approach is for the student model to learn the generalisation of the teacher model [66]. Previous research showed that shallow student models are capable of learning complex functions of the teacher models while maintaining nearly the same accuracy. Summarising a recent survey on knowledge distillation [67], there are three main components in this approach:

- **Knowledge types**: the types of knowledge used to learn the student model can be further classified into (i) response-based, (ii) feature-based and (iii) relation based. The goal of (i) is to directly mimic the final prediction of the teacher model. This is achieved by feeding the response of the last output of the teacher model into the student model. One of the disadvantages of this approach is that it can only handle supervised learning. (ii) is an extension of (i) in so far as rather than just learning the output of the final layer, the student model also learns the outputs of the intermediate layers, also called the feature maps. Finally (iii) rather than using the outputs of some specific layers, the focus of this approach is on the relationships between the feature maps.
- **Distillation strategies**: this refers to the training approaches for teacher and student models. Based on whether the student and teacher models are updated simultaneously the distillation strategies can be divided into three categories: (i) offline, (ii) online and (iii) self-distillation. The offline distillation approach is done in two steps, first, the teacher model is built based on the training data and then the student model is built based on the

knowledge from the teacher model. On the other hand, the online distillation strategy learns both teacher and student models simultaneously. This approach often takes advantage of parallel high-performance computing. Finally, self-distillation uses the same networks for both teacher and student models. To better distinguish the three approaches a good analogy is used in [67], where in method (i) the teacher teaches the student, in method (ii) both teacher and student learn together and in the final method (iii) the student learns by themself.

- **Teacher-student architectures**: this refers to the problem of the right design of structures in teacher and student models. The general idea of knowledge distillation is to transform the wider and deeper teacher model into a smaller and shallower student model. Therefore, a student model can become one of the following options: (i) a model with fewer layers and channels, (ii) preserved structure of the teacher model in quantised version, (iii) keeping efficient basic operations, (iv) minimised model optimising global structures and (v) same as the teacher.

4.3.5 Low-rank Factorisation

In neural networks, the weights being shared among working nodes are represented as weight matrices. Hence, the size of such weight matrices can be effectively reduced by low rank factorisation methods. The goal of low rank factorisation is to represent the target matrix with two or more smaller matrices. One of the most common low rank factorisation techniques is Singular Value Decomposition (SVD). Two aspects of neural networks could be decomposed into their low rank alternatives, (i) linear layers, and (ii) embeddings [68]. Some approaches aim to decompose both aspects of neural networks. Model optimisation techniques utilising low rank factorisation can become twice as fast in the learning process as their uncompressed counterparts while showing only a 1% drop in the model accuracy [69],[70]. Low rank factorisation methods can be used to speed-up tasks such as for example text character recognition models, object detection and image retrieval [69]. SEDIMARK will provide approaches such as SVD to the end user, and SEDIMARK will allow the user to choose whether to apply the low-rank decomposition to the convolutional layer or the fully connected layer, similar to the approach studied in [71].

Another use case of low-rank factorisation is the fine-tuning process of Large Language Models (LLMs). Because LLMs often contain a very larger number of trainable parameters, fine-tuning such models can require substantial memory and computational resources. Therefore Low-Rank Adaptation of LLMs (LoRA) [80] can be incredibly useful to end users with limited access to computational resources. This is achieved by freezing the pre-trained LLM parameters and only train small representations of the pre-trained parameters in selected layers during the fine-tuning process. In SEDIMARK, the Recommender Systems module contains a retrieval algorithm which is based on neural sentence transformers, called the Bi-Encoder. We added LoRA to the Bi-Encoder model to allow for more efficient fine-tuning process of this model and we discuss the comparisons of the full model vs. the modified model using LoRA in Section 5.3.7.

4.4 Reducing energy on data sharing

Reducing energy consumption during data sharing is a crucial aspect of optimizing machine learning workflows, particularly when dealing with large-scale datasets. Data transfer across networks, especially for high-volume, high-frequency processes such as model training and inference, can be a significant contributor to energy consumption. The environmental impact of data sharing is amplified when vast amounts of data are continuously moved between centralized servers, requiring substantial computational resources and network infrastructure. This becomes particularly problematic in distributed systems, where data transfer between remote edge devices and central cloud storage increases the overall energy footprint. SEDIMARK aims to address these challenges by employing strategies that reduce the volume of data shared without compromising model performance. One such approach is the use of data compression techniques, which reduce the size of datasets before transmission. The use of advanced algorithms that compress data with minimal loss of important information, can significantly lower the energy required for storage and transmission. Furthermore, edge computing is integrated into the framework, which ensures that data processing is done closer to the source, reducing the need for large-scale data transfer to central systems. This decentralized approach not only cuts down on energy costs associated with long-distance data transfer but also helps to alleviate the strain on network infrastructures. Another technique that SEDIMARK employs to reduce energy consumption during data sharing is the application of data sparsity and pruning methods. By identifying and removing redundant or irrelevant data points, the framework minimizes the amount of data that needs to be transmitted. Additionally, data sharing can be optimized by using more energy-efficient protocols and transmission methods, such as low-power wide-area networks (LPWAN) or other technologies designed for low-energy data exchange.

4.5 Reducing energy on data storage

As the SEDIMARK framework primarily features a decentralised approach to data management at the edge, it is necessary to take into consideration energy consumption as a result of data storage, which in cases of some providers will be crucial for maintaining reasonable costs. Increasing energy efficiency in relation to data storage on the data level, mainly involves reducing the storage footprint of artefacts produced during the data processing pipeline, and the storage of the final Data Assets that will serve as part of the providers Offerings.

In addition to data reduction methods such as deduplication as defined in Section 3.7.3, compression techniques can also be used, in addition to aggregation methods where Data Assets sources are co-located. Single-instance approaches can also be used at a higher level in comparison to deduplication, whereby similar chunks within Data Assets can be removed and linked to instead. Caching techniques can also be applied to Data Assets in cases where access to them is frequent. The granularity of annotations for Data Assets has a significant impact on the storage footprint, i.e. whether annotations are done on the atomic level of data points or on the level of windows or batches of time-series data points.

In relation to AI Model Assets, pruning (Section 4.3.3) and knowledge distillation (Section 4.3.4) techniques can be applied. These methods can allow AI Model Assets to operate with fewer computations, reducing the energy required for training and inference. Studies have shown that pruning combined with distillation can significantly reduce training energy costs while maintaining accuracy.

4.5.1 Compression for Data Processing Pipeline Artefacts

For intermediate artefacts generated as a by-product of a data processing pipeline as illustrated in Section 3.3, "*downcasting*" can be used is to reduce the primitive datatypes used from one of a higher magnitude to a lower, such as *float64* to *float16*, as long as the corresponding value is within the bounds of the smaller datatype.

More efficient datatypes can be adopted for data columns with a small number of unique values, such as the case for labels. In the case of Pandas, function such as "*Categorical*" and "*to_numeric*".

For certain types of dataset serializations, several compression formats can be applied to significantly reduce the size footprint, such as in the case with pickle which supports *gzip*, *bz2*, *zip*, *xz*, and *zstd*.

4.5.2 Broker Storage Configuration

Data Brokerage implementations are normally built upon well-established open-source databases. How these databases are configured can have a significant impact on the storage footprint. Data Providers are expected to produce data assets at different velocities and volumes, and hence how their data assets are evolving need to be monitored to continuously re-configure as per need and compute resource constraints of the hosting server.

In the case of the Stellio Context Broker (Section 3.3), the PostgreSQL database serves as the underlying data store. Performance tuning tools can be used, which focus on configuration settings for a set of application scenarios, in relation to storage requirements, especially for shared_buffers which acts as the cache.

4.6 Pushing data processing to the edge

One effective strategy for reducing energy consumption in data technologies is to shift some processing closer to where the data is generated (i.e., at the edge). In many scenarios, local processing allows devices to reduce the frequency and volume of data sent to the cloud, which can significantly lower energy consumption. This may lead to them being included "on the sensor", directly on the microcontroller unit (MCU) that reads the probe and controls the sending of the data to the network. One can think of various ways to integrate such capabilities into a dynamic data platform such as SEDIMARK. The current choice is to explore the possibility of pushing processing algorithms from the platform to the MCU in the sensor by using a bytecode language (WebAssembly) via some Edge Cloud orchestration services (see SEDIMARK_D4.3 [6]). This allows the simplification of DevOPs operations by setting up a single compilation chain on the cloud side, at the price of providing an implementation of the SEDIMARK WebAssembly platform API for each MCU one wants to support. This brings the main advantage of being able to integrate any kind of sensor, by any vendor, without requiring them to set up a full compilation chain for their device on the Cloud side. The vendor will only need to work on their device, making them compatible with the SEDIMARK API and protocols.

The choice of WebAssembly bytecode is driven by its fast adoption and support by major vendors, its very compact size when compiled, the availability of bytecode interpreters and compilers on all kind of platforms (including microcontrollers), and the availability of compilation chains for various languages (including C++ or javascript for example).

5 Trade-offs between performance, communication cost and energy efficiency

5.1 Overview

Data processing may require significant computing resources to analyse the datasets, depending both on the size of the dataset and the type of processing to be done or the selection of the processing algorithm. Similarly, training machine learning models also require significant resources, both when training data locally and in a distributed way. The previous section presented techniques to reduce the energy consumption when training machine learning models or when sharing data. However, these methods normally use less energy with the cost of performance or accuracy. It is therefore critical to assess the trade-offs between energy consumption, performance and communication cost in a way that data providers of consumers can know the effects of targeting for (i) maximum energy efficiency against performance and vice versa or (ii) minimum communication cost against performance. This subsection presents the final analysis of the trade-off results, expanding the initial analysis presented in the first version of the deliverable SEDIMARK_D3.1 [74].

5.2 Communication cost analysis for distributed learning

This section investigates the trade-offs between communication efficiency and the overall model accuracy. The experiments and analysis are performed using deFLight, a tool that enables federated learning and gossip learning approaches and is already implemented in SEDIMARK and presented in more detail in SEDIMARK_D3.3 [5].

Experimental Setup:

- **Dataset**: A small subset of the MNIST dataset is used [54]. This is a labelled dataset of handwritten digits and the goal of an ML model is to correctly recognise such digits. To mimic a real-world data distribution in decentralised systems, the dataset is divided between the worker nodes in a non-IID fashion. In particular, each working node is assigned 50% of the images attached to one label and the other 50% attached to another label. A small portion of IID data distribution in the form of 100 IID images per working node is also added.
- **deFLight setup**: The nodes passively receive model weights from other participating working nodes (in Gossip learning) or from the server (in Federated learning). Communication percentage values C are set to [0.25,0.5,0.75,1.0]. In the context of Federated learning this corresponds to the percentage of worker nodes participating in the model aggregation step by the server and in the context of gossip learning this corresponds to the percentage of communication peers for each worker node. A record of the overall number of bits exchanged between participating nodes/server during the learning process is also kept. The goal is to keep high accuracy while maintaining the smallest possible communication cost (i.e. the number of bits exchanged).

5.2.1 Federated Learning

The federated learning module for each communication setup is run over 10 runs and reports the average across those runs. Figure 22 illustrates that the model has a smooth convergence curve achieving high model accuracy when the level of communication is set to 100%. On the other hand, the convergence curve is not so smooth with the communication level of 25% also resulting in a significant final performance drop of 0.23. However, note that by decreasing the level of communication from 100% to 50% the drop in final model accuracy is just 0.05, while being able to decrease the number of bits being sent around (at that point in time) by 5.8 billion which equals significant energy savings, as shown in Figure 23.

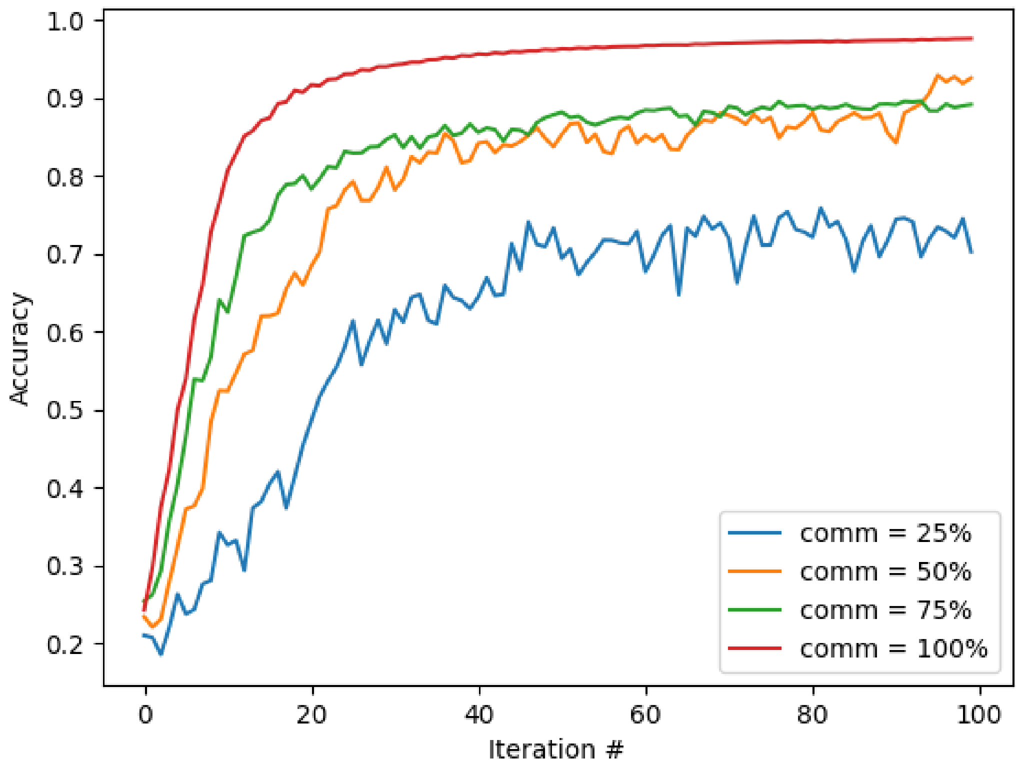


Figure 22: Accuracy for different percentages of selected nodes for communication in federated learning.

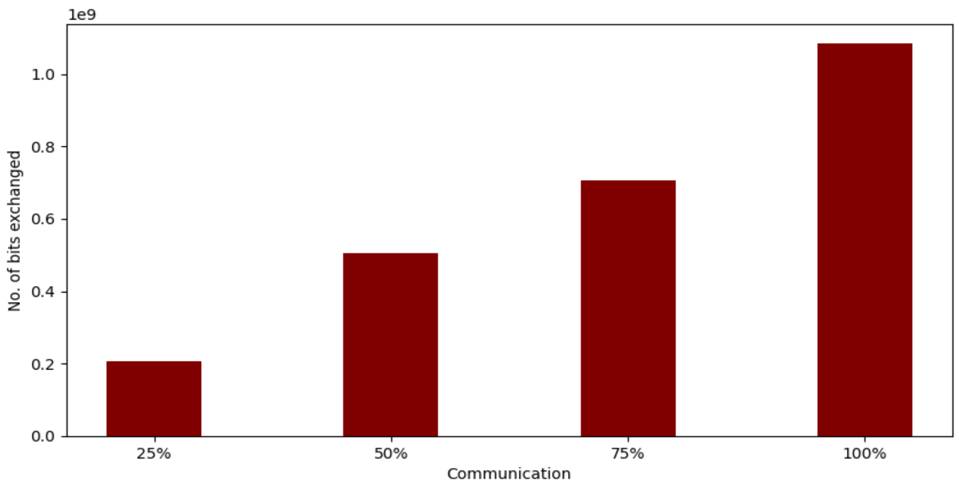


Figure 23: Communication cost (in bits) for different percentages of selected nodes for communication in federated learning.

5.2.2 Gossip Learning

This experiment is averaged over 4 runs and the results are plotted in Figure 24. As expected, the model achieves the best accuracy when the worker nodes are set to 100% communication. However, by decreasing the level of communication to 50% a drop in accuracy from 0.98 to 0.97 is noted, while decreasing the level of communication by half, which in this case equals to a whopping 276 billion bits saved from communicating (shown in Figure 25). This illustrates how massive savings in energy consumption through communication volume can be achieved with minimal impact on final model accuracy.

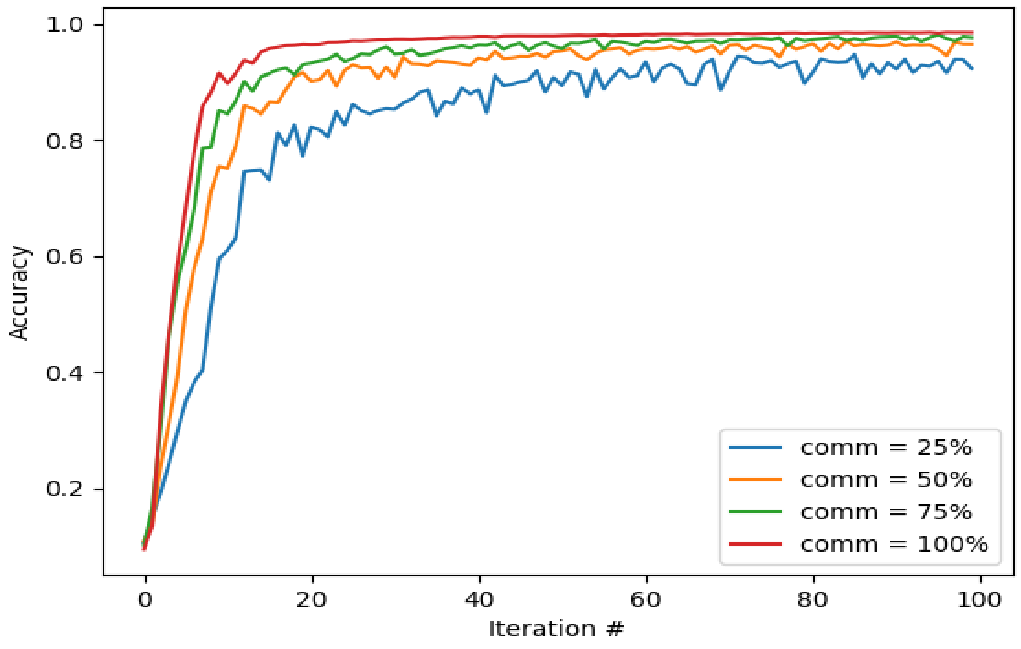


Figure 24: Accuracy for different percentages of selected nodes for communication in gossip learning.

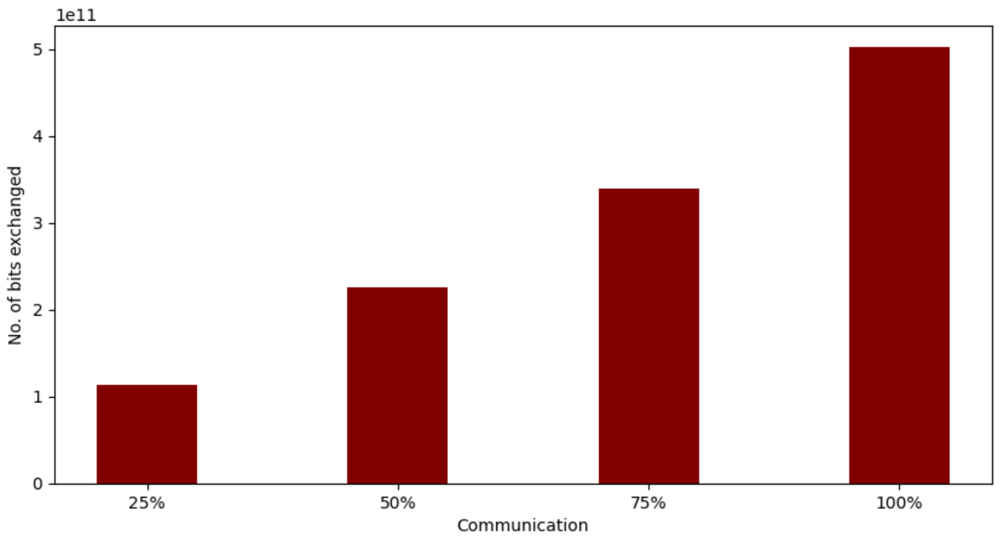


Figure 25: Communication cost (in bits) for different percentages of selected nodes for communication in gossip learning.

5.2.3 Communication/Computation cost versus performance comparison with topologies in deFLight

In this section we further extend our experiments with deFlight to consider two newly introduced topologies, Split Learning (SL) [90] and Federated Distillation (FD) [89]. We again employ the MNIST dataset, setting aside 20% of the data to be randomly distributed between all nodes, and with the remaining data assigning only 2 labels to each node, thus creating a non-iid split. In the case of gossip learning, we use a fully connected graph. Both SL and FD are implemented without any server-side aggregation of model weights. We then allow the experiments to run for 2 hours each. In Figure 28 and Figure 29, we then plot a rolling mean (with window size 50) of the accuracy attained against a) the number of bits communicated and b) the number of floating point operations per second (FLOPS) in one forward pass of a model, multiplied by the total number of forward passes conducted by the clients at each point in time.

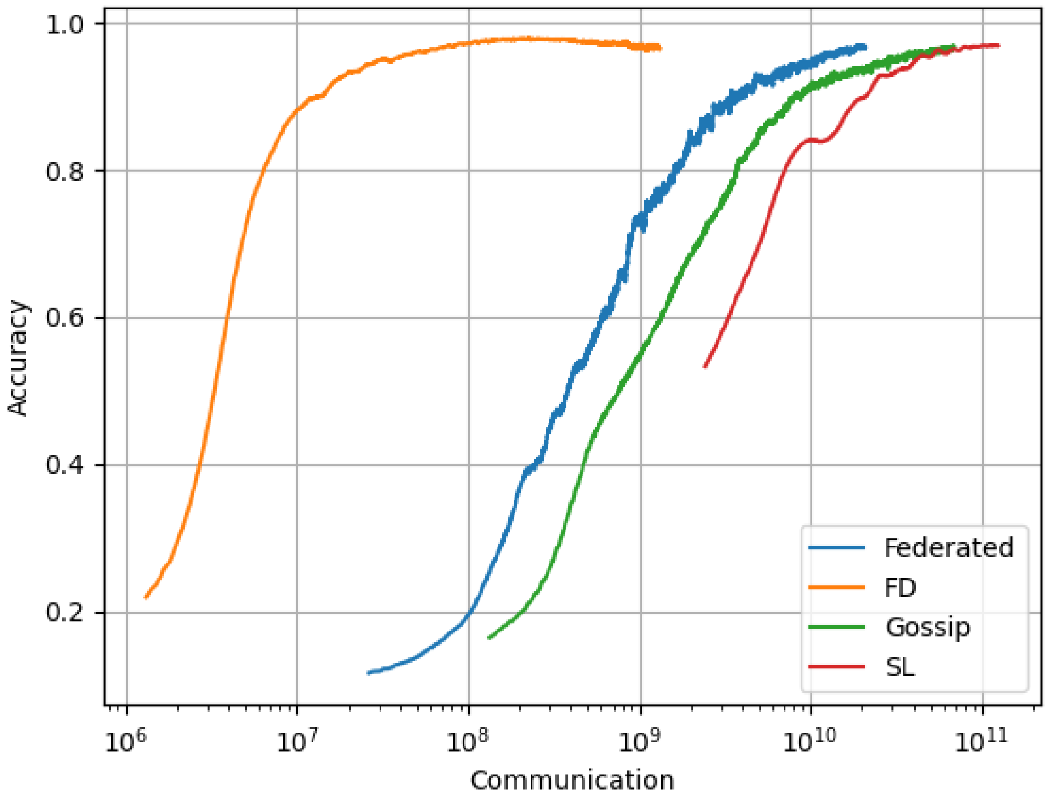


Figure 26: Communication cost (in bits) for various topologies training on the MNIST dataset within deFlight.

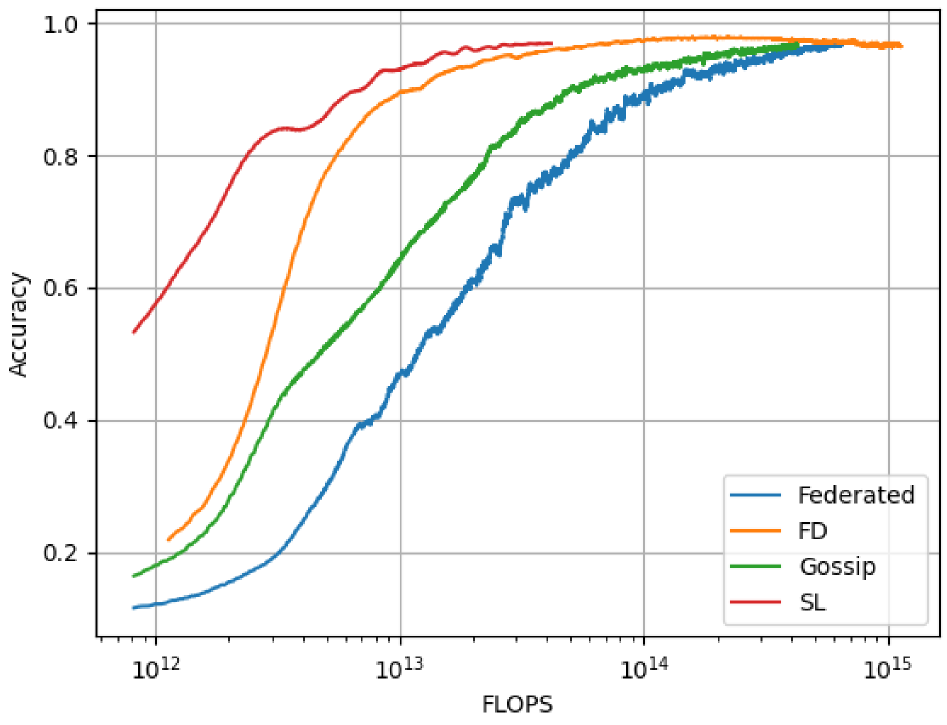


Figure 27: Computational cost in FLOPS for various topologies training on the MNIST dataset.

We observe that, while Federated is unsurprisingly more efficient in terms of communication, Gossip employs somewhat less FLOPS in order to reach a similar level of accuracy. FD shows great benefit in terms of the communication cost, and this is due to the fact that instead of communicating full model weights, it is only sending the logits for a relatively small number of datapoints. On the other hand, SL incurs a high communication cost, as it needs to communicate tensors and gradients after every batch of data that is processed. However, it is fairly efficient in terms of FLOPS, likely, because the central model is being trained asynchronously. The good performance of FD in terms of flops might be ascribed to the fact that the global portion of data used to share logits is IID.

5.2.4 Communication cost versus performance in Fleviden

In this experiment we leverage Fleviden to analyse the trade-off between accuracy and communication cost in a Federated Learning scenario. Like in the previous experiment conducted with DeFLight, we will train a Convolutional Neural Network for the classification of the MNIST dataset, measuring the differences in accuracy and loss when the number of participants N in a training round is limited to a percentage C of the whole federation –i.e., only N*C clients train and contribute to the round aggregation–. The dataset has been divided into ten subsets where, to emulate a typical Federated Learning casuistic, each subset contains

80% of the samples of one label and 20% of the rest –i.e., client zero will have most of the digit zero images, and so on. The resulting label distribution per client can be seen in Figure 28.

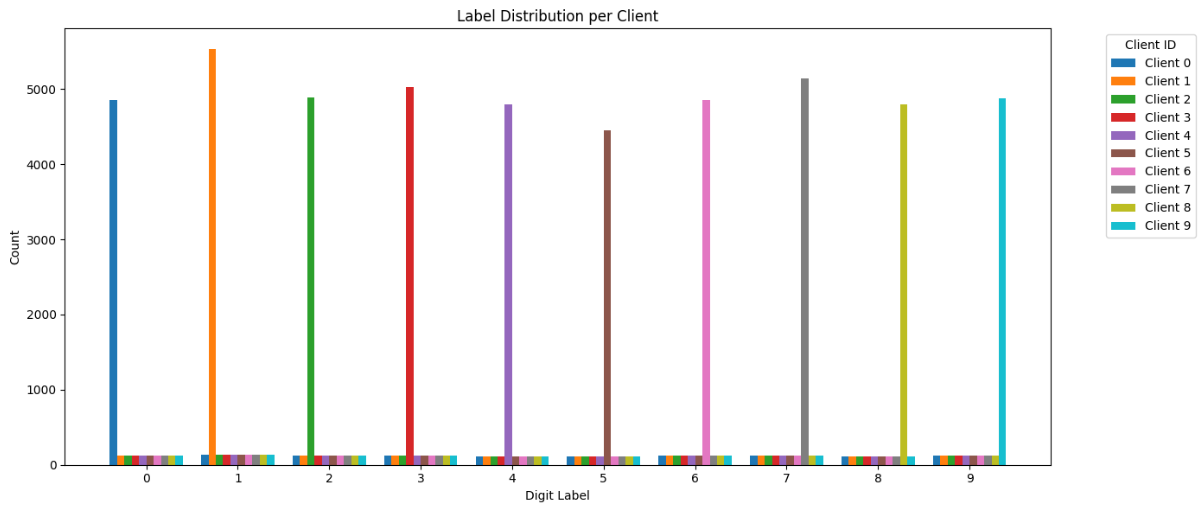


Figure 28: Label distribution per client in the experiment, where each client represents most of the samples for each digit label (highly imbalanced distribution)

The communication cost per round is measured in terms of the total number of bytes exchanged between clients and the aggregation server. The accuracy is measured over a uniformly distributed test set at the server at the end of each round. A FedAverage strategy is used for the aggregation. In total, 15 rounds are completed per experiment run. Regarding local training configuration, clients train the model for 1 epoch with a randomly selected subset of 3000 samples. The reason behind this limitation is to homogenize the number of images used in different clients, since the total number of samples varies depending on the digit.

The experiment evaluates the test accuracy for C in [0.2, 0.4, 0.6, 0.8, 1.0]. The subset of clients in a round when C<1.0 is chosen randomly. The resulting accuracy curve obtained is shown in Figure 29.

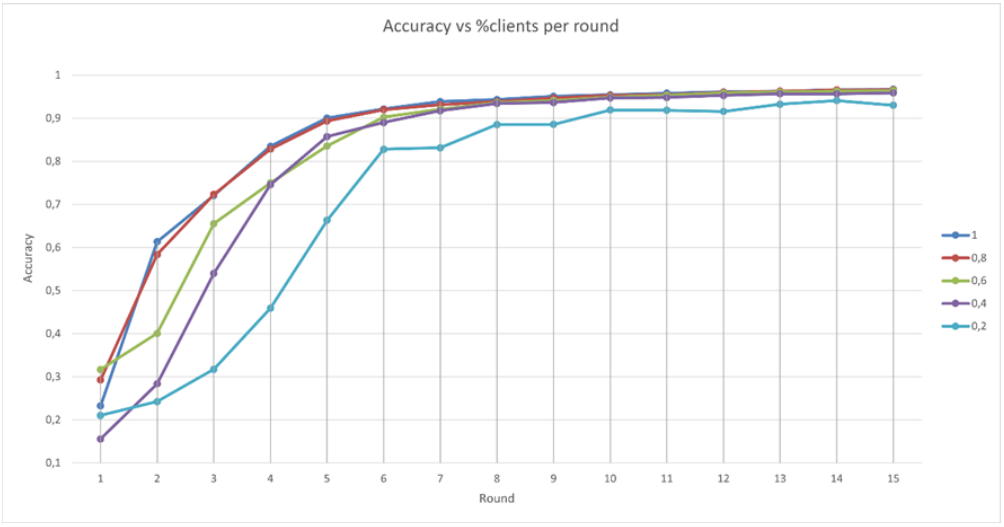


Figure 29: Accuracy versus number of clients per FL round

Due to the use of the toy dataset and the expressiveness of the CNN, all models eventually reach an accuracy around 0.98, although, as an obvious result, the higher the number of participants, the faster convergence occurs. In this case, however, differences between C=0.6, C=0.8 and C=1.0 are practically negligible. As can be seen in Figure 30 below, each client represents around 7 MB in the total sent to the server, which means that we could save almost 30 MB per round with no detrimental effects.

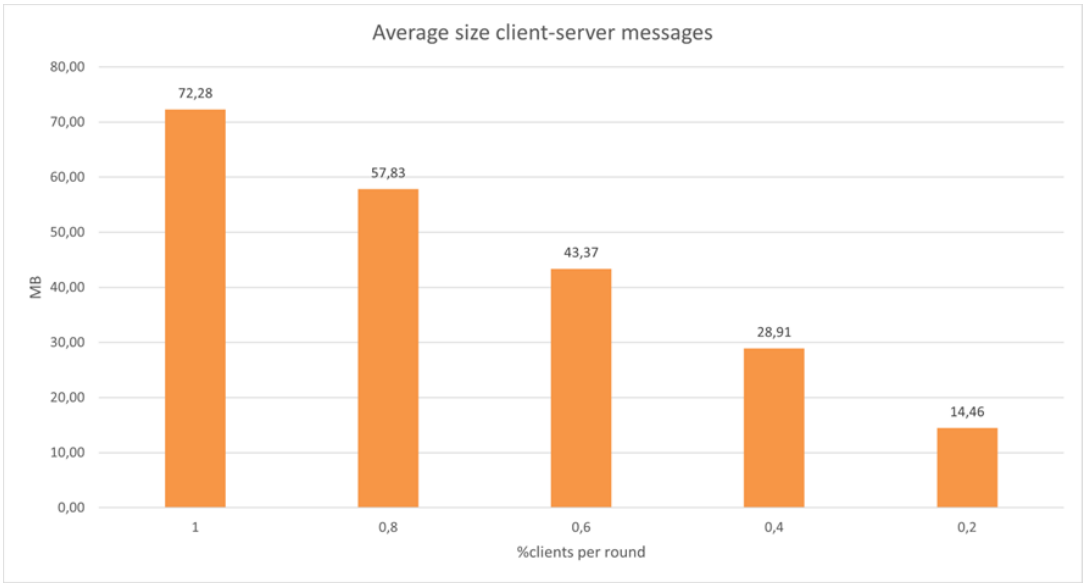


Figure 30: Average size of exchanged messages between clients and server

The results are, as expected, similar to those achieved in the deFLight experiment.

5.2.4.1 Communication cost of Fleviden's lossless compression

Even though in this toy scenario the CNN model is not too large for communication to become prohibitive, applying lossless compression in our federated learning setup can help us reduce the size of the message exchanges at a relatively small computational cost. To do so, we leverage Fleviden's compressor pod to encode and decode messages before client-server communication takes place.

The compressor pod is configured to use MessagePack as the binary serialization format, Zstandard as the compression algorithm –which prioritizes speed over compression ratio–, and base64 as the string code for JSON-serialization. Every other parameter is the same as in the previous setup. Since, by definition, lossless compression doesn't modify the values of the model's weights, the accuracy curve is practically identical to the one in Figure 29, so it is omitted here. In terms of communication cost, Figure 31 shows the average number of bytes sent to the server in a round, for C= [0.2, 0.4, 0.6, 0.8, 1], compared with the vanilla scenario with no compression.

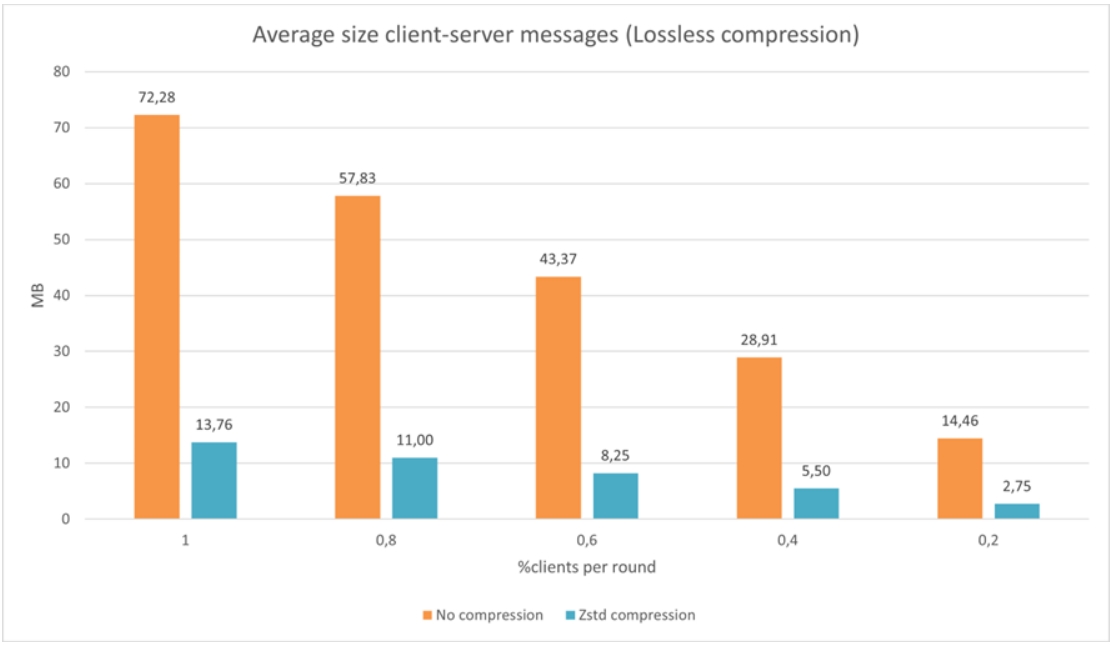


Figure 31: Average size of exchanged messages w/o lossless compression

The achieved compression ratio is around 5.4, which reduces the size obtained with C=1.0 to a lower value than the one with C=0.2 and no compression. Because Zstd is very fast, the cost in terms of time is negligible in comparison with the local training time. As such, the benefits outweigh the drawbacks; however, in lossless compression the ratio that can be obtained is always upper bounded. To further compress the size of the messages, a quantization approach must be used.

5.2.4.2 Performance vs communication cost of Fleviden's QSGD

In this experiment, we leverage Fleviden's quantization pod and compare the accuracy and compression ratio achieved for different setups of the QSGD algorithm. Namely, we vary the number of quantization levels (L) parameter in L= [175, 200, 225, 250, 500, 2000]. These values have been selected heuristically through a preliminary experiment, because QSGD is sensitive to the model size, and thus there is not a rule of thumb for the proper selection range. The higher the number of levels, the fewer the reconstruction error of the decoding, and the higher the computational cost of the encoding. All clients participate in every round (C=1.0), but the other hyperparameters remain the same.

The accuracy curves obtained are shown in Figure 32 below.

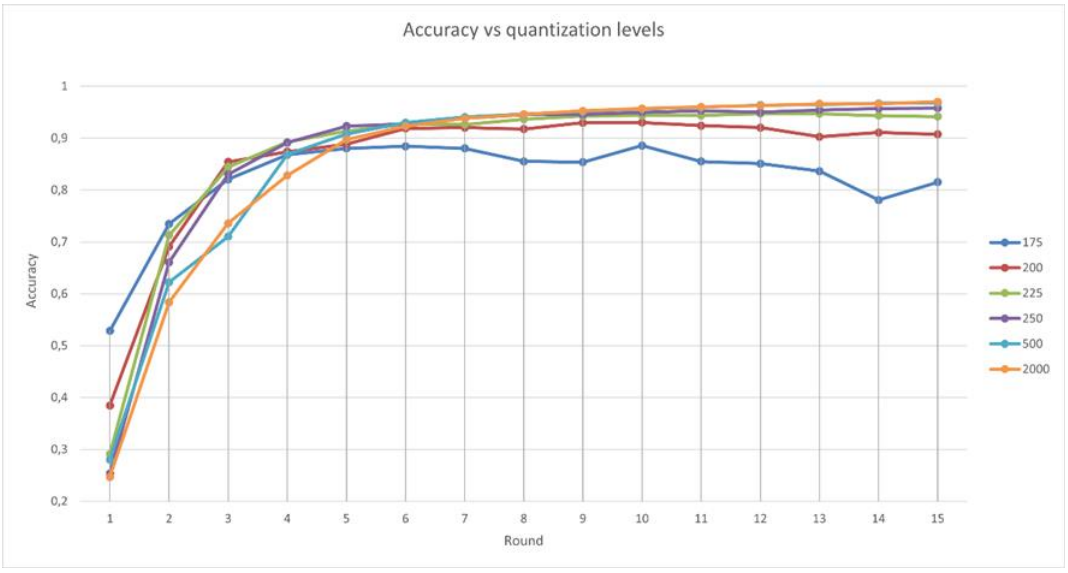


Figure 32: Accuracy versus number of quantization levels of QSGD

On the other hand, the number of megabytes exchanged for each L is shown in Figure 33. For reference, the sizes without compression and with Zstd are included in the chart.

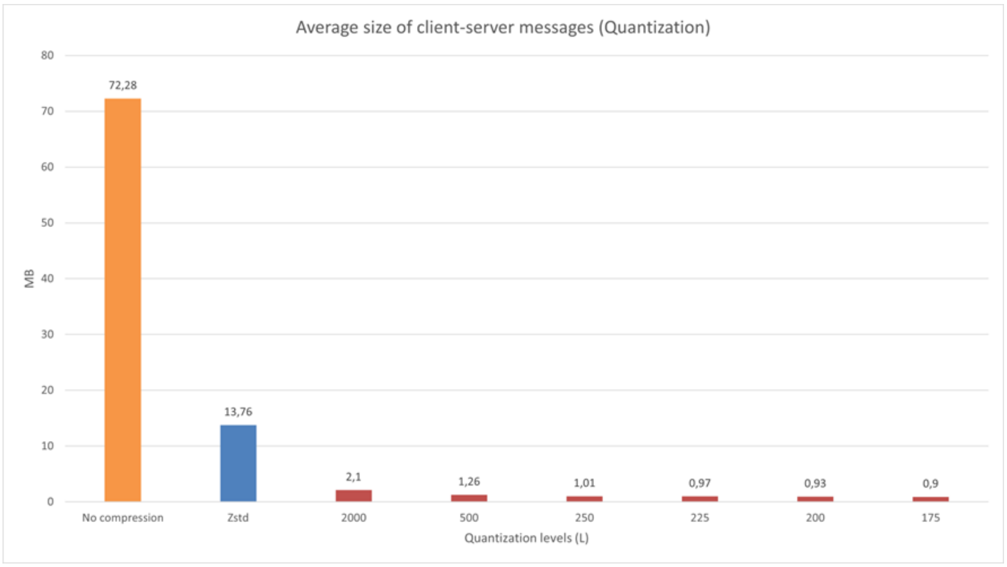


Figure 33: Average size of exchanged messages w/o QSGD quantization

Regarding accuracy: QSGD quantization step pushes values to zero and decreases the dynamic range of the input, which translates into reducing the expressiveness of the CNN and consequently worsening the performance. When L=2000 (lowest quantization error), the final accuracy reaches 0.96, which is a 0.02 loss with respect to lossless techniques. The computational cost of encoding and decoding is, however, too high, to the point that it takes several times the training time of an epoch. On the other end, L=175 impacts much more significantly on the performance, with a drop of 0.15 that can be considered too drastic to justify its use. Further lowering the number of quantization levels results in unstable training and the curves do not converge. Regarding the compression ratio: the gain is quite significant with respect to Zstd. With L=2000, the ratio is around 36, which increases up to 80 with L=175. A good balance between the two is found at 225, where the accuracy decreases only to approximately 0.94, with a ratio of 72 and acceptable encoding-decoding time. It is important to highlight that the classification problem is not very hard, and the dataset contains enough samples to make the stochastic gradient descent smooth. In other more realistic scenarios, the user should evaluate the trade-off more carefully and tune the number of quantization levels to find the optimal spot in the trade-off.

5.3 Analysis of trade-offs in performance vs computational cost

This section investigates the energy efficiency trade-offs for a number of modules that are already implemented within SEDIMARK. The performance of three modules is considered: (i) the deFLight distributed learning module (described in SEDIMARK_D3.3 [5]), (ii) the Data Deduplication and (iii) Anomaly Detection modules within the data pipeline. All three of these modules offer a number of parameters or different methods for which the trade-off between their computational efficiency and performance on a dataset can be examined. In the following experiments, the popular python package eco2ai [53] is employed to estimate the CO_2 consumption of the target algorithms.

Additionally, in this section performance evaluation of other techniques that have been integrated within the SEDIMARK Toolbox (i.e. timeseries and data streams anomaly detection, synthetic data generation, low-rank factorisation, time-series forecasting and offering generation) are also presented.

5.3.1 CO_2 consumption of Federated Learning in deFLight

This section considers the CO_2 efficiency of the federated learning scenario implemented with the deFLight distributed learning component of SEDIMARK as the amount of client-server communication is adjusted. Following the protocol described in greater detail in section 5.2, ten non-IID training subsets of the MNIST dataset are created, and small convolutional neural networks at each node are trained. Each experiment is run for 100 iterations and both the accuracy attained and the CO_2 output as measured by eco2ai are compared.

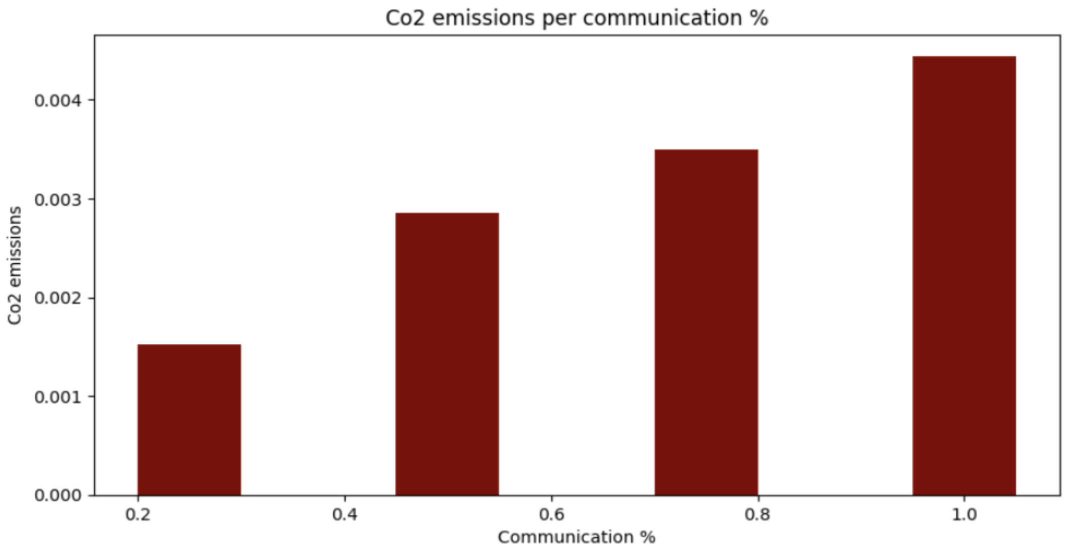


Figure 34: CO_2 cost per 100 iterations of FL given communication percentage.

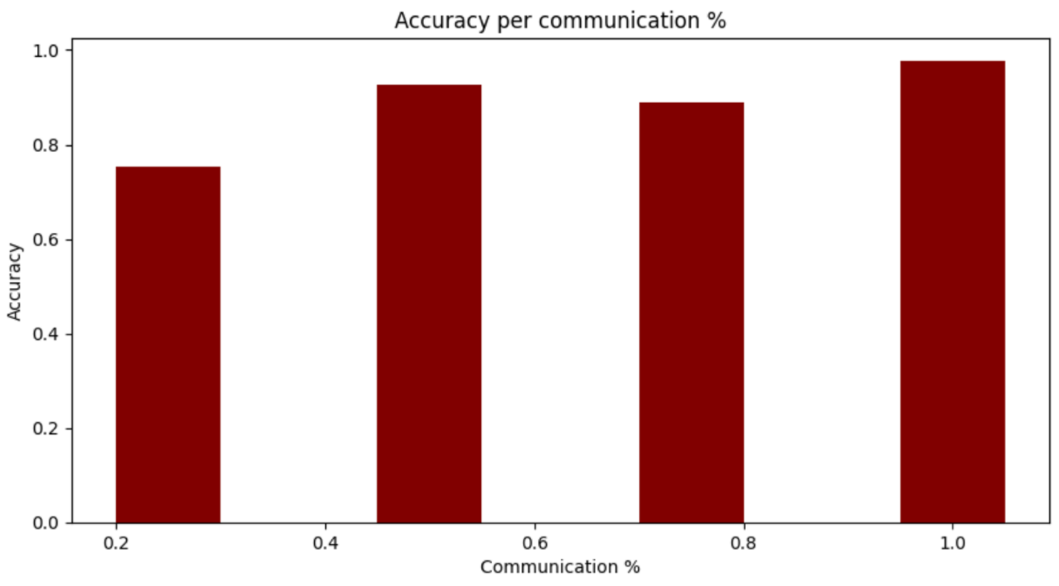


Figure 35: Final accuracy for 100 iterations of FL given communication percentage.

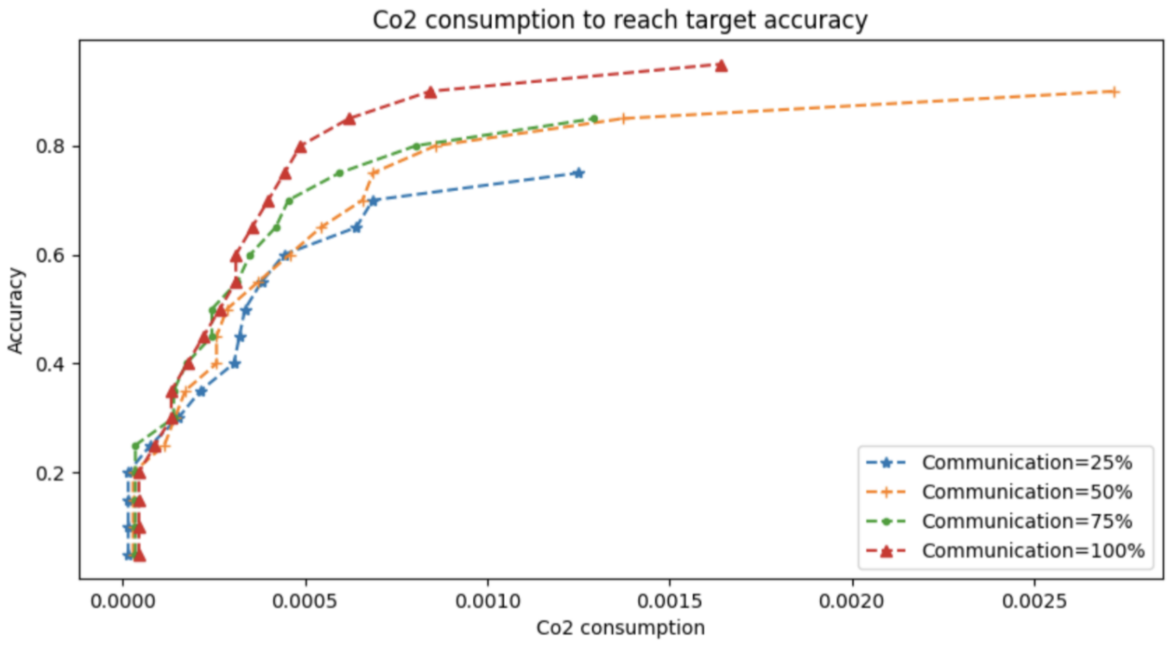


Figure 36: Comparison of CO_2 cost to reach given accuracy levels for different communication percentage in FL.

In Figure 34, the CO_2 cost of running a full 100 rounds is observed to increase with the amount of communication but in Figure 35 there is a trade-off here in the final accuracy attained. However, Figure 36 shows that for any given target accuracy, running with higher communication is actually more CO_2 efficient. With greater communication, the algorithm converges much faster, which is demonstrated further in section 5.2. However, full or even high communication can be quite unlikely in real world settings. As such, going forward the DeFLight SEDIMARK module might target a) more efficient sampling of nodes and b) non-IID robust aggregation and distributed model training techniques such as e.g. FedProx [55], in order to decrease the CO_2 consumption in low communication non-IID settings.

5.3.2 Performance of data deduplication module as the indexing method is varied.

As described in section 3.7.3, the deduplication module includes an indexing component with a significant effect on the computational cost of reaching a deduplication solution. Naively indexing the dataset results in comparisons taking place between all pairs of records and thus an $O(n^2)$ runtime complexity. However, there are two supplemental methods included - *Block* and *SortedNeighbourhood* Index, which greatly reduce the number of comparisons made. The *Block* method only compares records that agree on a specific variable - for instance in a dataset of restaurant records, it might only compare restaurants within the same city. The *SortedNeighbourhood* indexing method extends this by also including records within their neighbourhood - e.g. perhaps if there are likely to be small spelling mistakes in the city name, these will be compared to. The three methods are run on the Fodor/Zagat restaurant dataset, which has ground truth duplicate labels [72]. For the indexing methods, the 'city' feature in the dataset is used as an efficient way to draw out blocks of the data. Precision, recall and CO_2 consumption are recorded and compared for the three indexing methods.

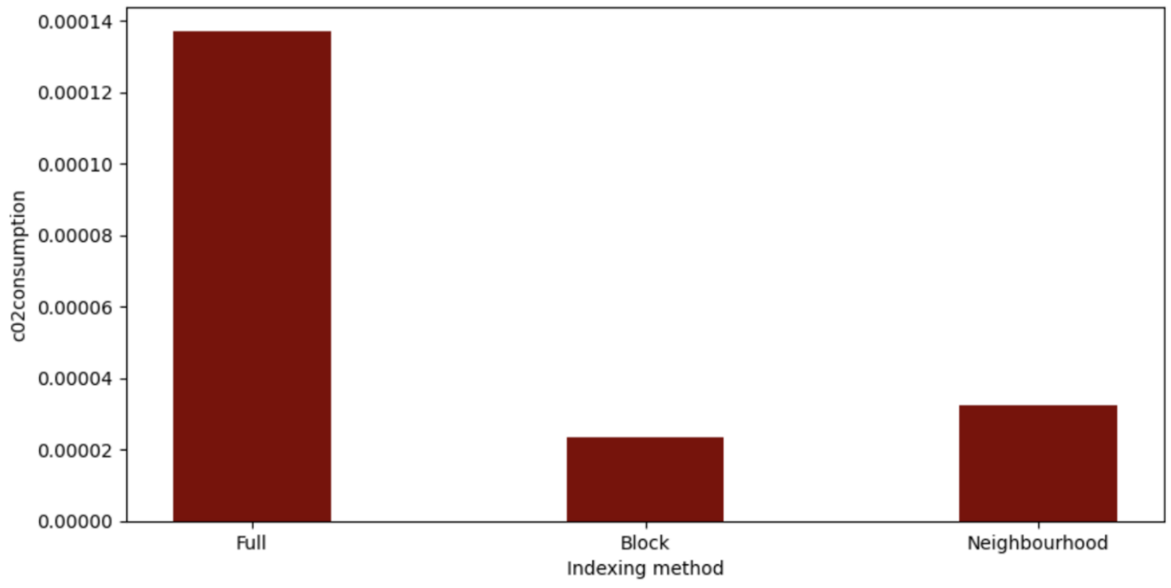


Figure 37: Comparison of CO_2 cost of different indexing methods for deduplication.

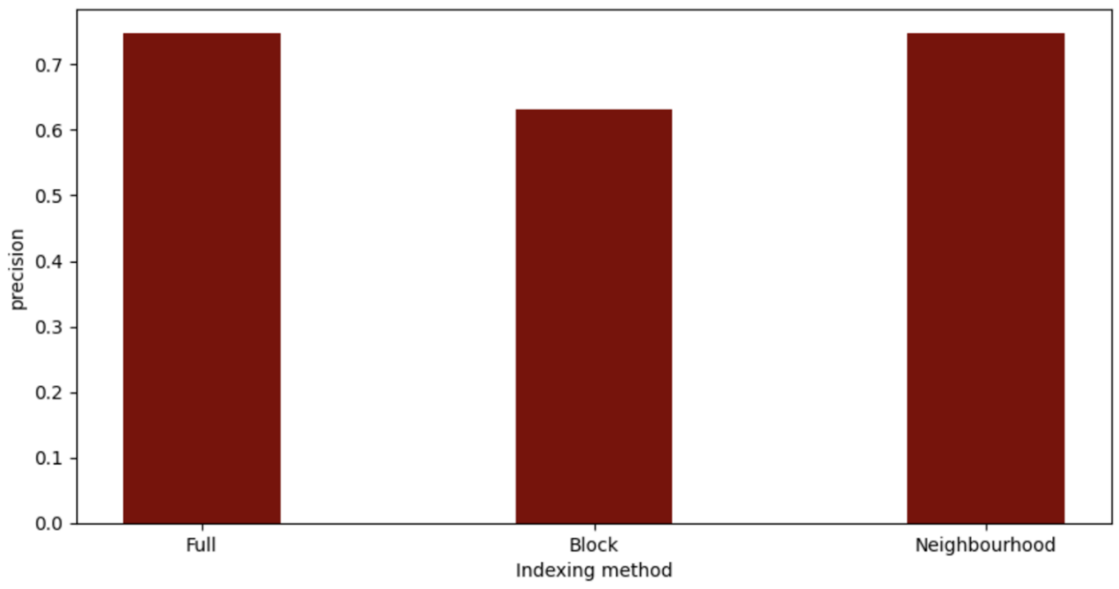


Figure 38: Comparison of precision of different indexing methods for deduplication.

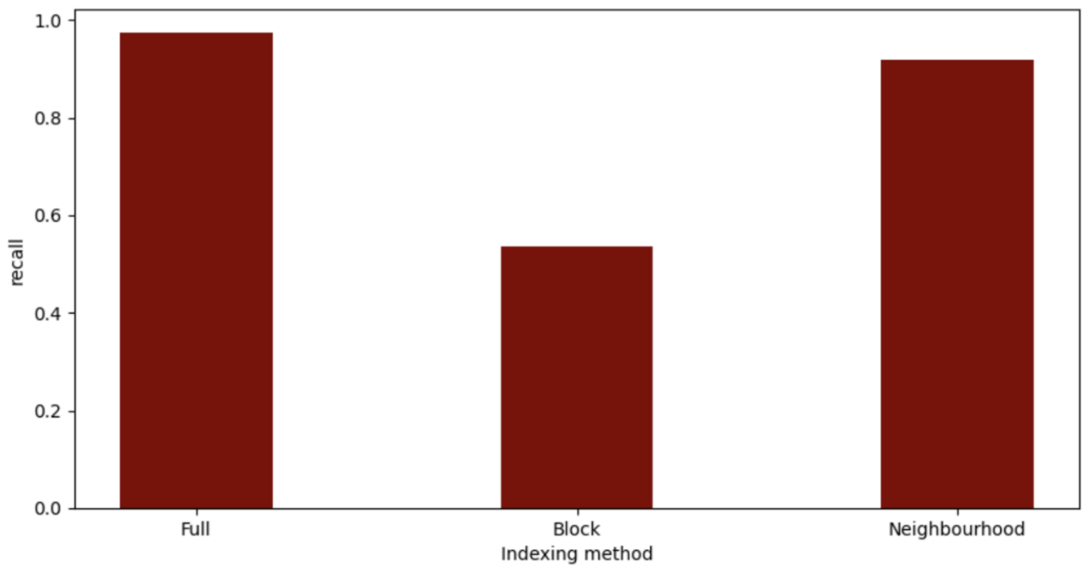


Figure 39: Comparison of recall of different indexing methods for deduplication.

In Figure 37 the high CO_2 cost of full indexing can be immediately observed. While 'Block' based indexing has a greatly reduced CO_2 cost, this increases somewhat for the Neighbourhood method which also searches through neighbouring strings. In Figure 38, negligible drops in precision for both of the block-based indexing methods are observed. On the other hand, in Figure 39 it can be observed that recall is much higher for the full indexing method, with a small drop for Neighbourhood, indicating that pure Block based indexing causes us to miss comparing many records that would have been valid duplicates. Going forward, the project's intention is to explore further indexing methods that do not necessarily sacrifice recall while still greatly reducing energy consumption compared to full indexing of the provided dataset. Smart data models might also be exploited to automatically select performant features on which to index the data.

5.3.3 Performance of Anomaly Detection module on ground truth, employing different models.

The SEDIMARK Anomaly Detection module contains a large number of different models, which might be suited to different kinds of datasets. This section compares a large number of algorithms on a generic tabular anomaly detection task. This work considers the tabular anomaly detection task for which PYOD is built and experiments are run on the thyroid dataset introduced by [56]. 11 models are considered: ABOD, AutoEncoder, Feature Bagging, HBOS, Isolation Forest, KNN, LOF, MCD, OCSVM, PCA and DeepSVDD (discussed in section 3.7.1). As the focus of this experiment is to show how a naive user might interact with these modules, they are deployed using their default parameters from the PYOD library. As a performance metric, the Receiver Operating Characteristic Area Under the Curve (ROC AUC) score is used, measured against the true labels provided in the dataset.

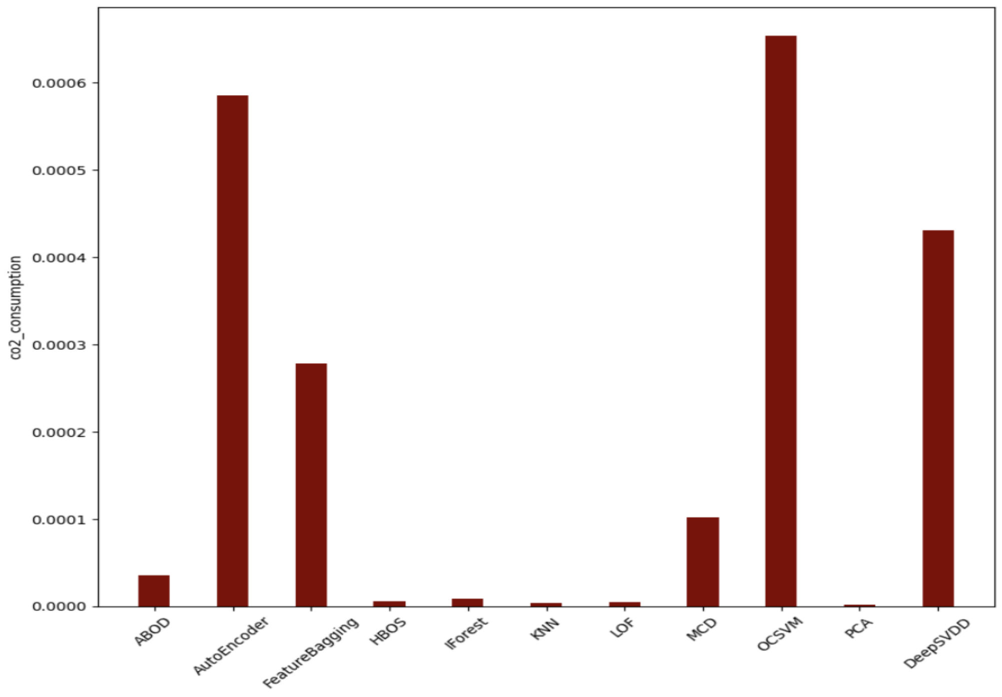


Figure 40: CO_2 cost for the various anomaly detection methods available in the SEDIMARK pipeline.

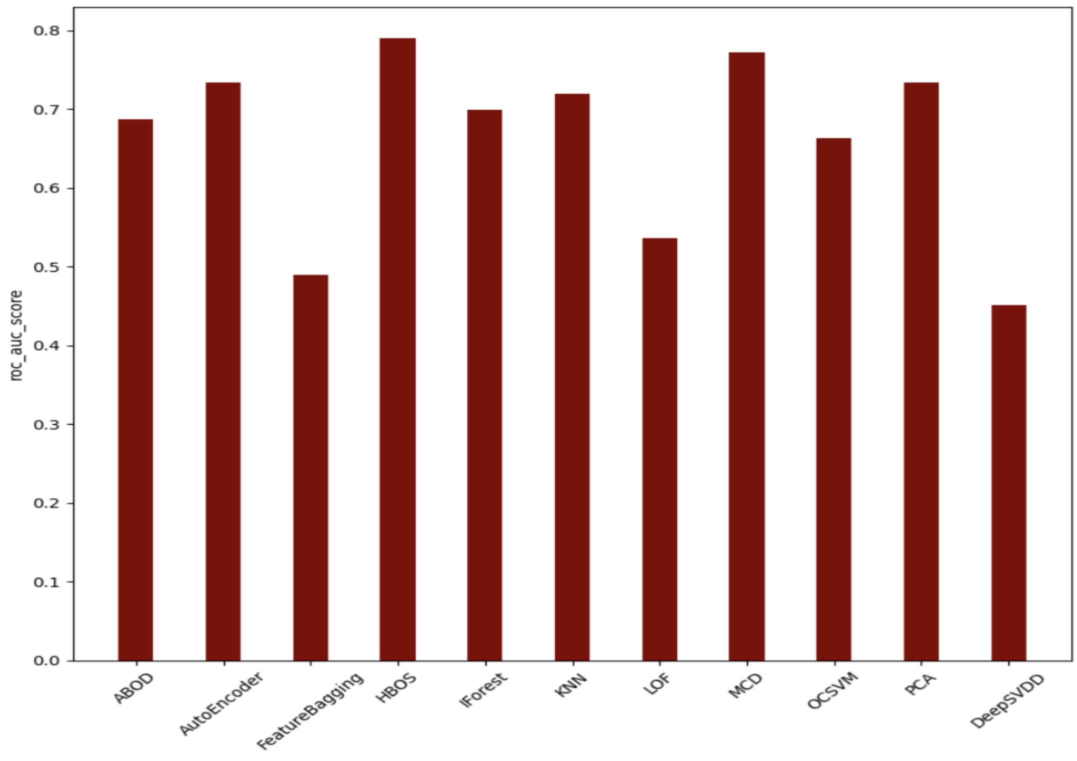


Figure 41: ROC AUC score for the various anomaly detection method available in the SEDIMARK pipeline.

Comparing the results in Figure 40 and Figure 41, one can see that unlike the algorithms in the previous sections, anomaly detection does not appear to show a clear trend between the energy consumption produced and performance on the given task, with one of the lowest cost models (HBOS) actually attaining the best performance. Furthermore, one of the most energy consuming methods (DeepSVDD) has the lowest performance overall. This signals towards a problem in the deployment of unsupervised methods - as the methods do not optimise directly against a ground truth, it is hard to know a priori that e.g. a model with more parameters might better fit the data. Indeed, this is in line with the problem of unsupervised model selection discussed in section 3.7.1. As such, going forward, SEDIMARK seeks to further understand the series of trade-offs implicit in fitting unsupervised methods in the absence of ground truth data. SEDIMARK plans to employ some of the AutoML methods referenced earlier in the deliverable (sections 3.7.1 and 3.10) to optimise both for energy consumption and accuracy.

5.3.4 Performance of Time Series Anomaly Detection module on ground truth, employing different models.

The SEDIMARK Anomaly Detection module also integrates a large number of time series anomaly detection algorithms through the TimeSeAD [85] benchmark. In this section we briefly deploy the algorithms with their default hyperparameters on a section of the SMD dataset [88]. For algorithms requiring training, we set the number of epochs to 10. We largely adopt the evaluation process from [85]. We train the models on clean training data, then evaluate them on the contaminated test data against the set of withheld anomaly labels (which aren't required for training). We measure performance using the best_f1 metric suggested by the same authors, and we record the co2 cost for our experimental runs in Table 2.

Table 2: Data Processing requirement analysis.

category	model	Best_f1_score	CO2 (grams)
baselines	eif	0.2743	2.5605
baselines	hbos	0.2929	0.1156
baselines	iforest	0.3579	0.1465
baselines	iqr_ad	0.1841	0.0707
baselines	kmeans	0.3705	0.0713
baselines	knn	0.4337	2.9676
baselines	oos_ad	0.184	0.0715
baselines	pca_ad	0.3999	0.0624
baselines	wmd_ad	0.7073	0.062
prediction	lstm_prediction_filonov	0.2216	0.1692
prediction	lstm_prediction_malhotra	0.7072	0.6427
prediction	tcn_prediction_he	0.7227	0.6114
prediction	tcn_prediction_munir	0.3198	0.5374
reconstruction	anomtransf	0.1838	5.8506
reconstruction	autoformer	0.1936	1.6414
reconstruction	dense_ae	0.3815	0.5127
reconstruction	etsformer	0.1941	2.2693
reconstruction	fedformer	0.1861	8.1379
reconstruction	genad	0.1838	1.6139
reconstruction	lstm_ae	0.5468	0.6523
reconstruction	lstm_max_ae	0.1841	0.6713
reconstruction	mscred	0.1838	12.3292
reconstruction	stgat_mad	0.4162	2.2248
reconstruction	timesnet	0.2717	22.8043
reconstruction	untrained_lstm_ae	0.3944	0.8002

category	model	Best_f1_score	CO2 (grams)
reconstruction	usad	0.1839	1.0528
other	lstm_ae_ocsvm	0.1838	1.1864
other	mtad_gat	0.4399	1.9327
other	ncad	0.1874	3.8797
other	thoc	0.2027	6.3633
generative	donut	0.599	0.7524
generative	gmm_vae	0.6841	0.9859
generative	lstm_vae_park	0.6978	1.0306
generative	lstm_vae_soelch	0.6864	1.29
generative	sis_vae	0.5387	29.7981
generative	beatgan	0.2043	23.8155
generative	lstm_vae_gan	0.2044	1.4443
generative	madgan	0.2003	9.1934
generative	tadgan	0.1845	1.6854

As with the results in the previous section, there is no clear relationship between the computational cost of the algorithms and their performance on the dataset. This signals both the necessity for human oversight in the anomaly detection process, and the problems raised about many of the benchmarks in the field, for instance the performance of extremely simple predictors on many of the field's benchmark datasets [94].

5.3.5 Performance of Datastreams Anomaly Detection module employing EWMA model.

This section details the evaluation and performance characterisation process of the proposed solution for outlier detection in data streams described in Section 3.7.1.2. In order to be able to thoroughly discuss the proposed algorithm, the same evaluations have been carried out on the algorithms available in the River framework [41]. This framework promotes online machine learning for streaming data in Python and therefore provides a suitable reference for comparison.

The experiments were carried out using a dataset from the Santander pilot case platform, more specifically from a temperature sensor. These observations are historical values from January 1st to September 16th (2021), with a periodicity of approximately 5 minutes. Therefore, in total there are about 75,000 observations. This dataset has been divided into two separate portions: 45,000 observations as a training dataset and the remaining 30,000 observations as an evaluation or test dataset. The training dataset is classified as clean, while 40 randomly assigned anomalies have been forced into the evaluation dataset, so that the efficiency of the algorithms' detections can be subsequently evaluated. Figure 42 depicts the dataset used for the experiments. The training dataset is represented by a blue line, while the evaluation dataset is plotted with a light green line. In addition, the introduced anomalies are marked with dark green dots.

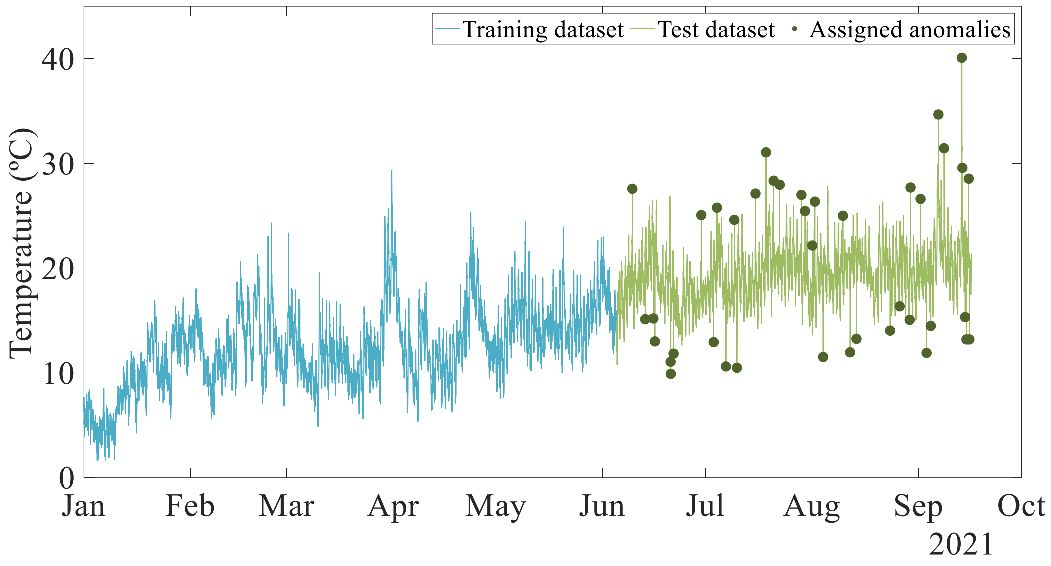


Figure 42: Santander temperature dataset divided in training and test dataset. Randomly synthetic anomalies have been introduced.

It is important to note that, since the algorithms are focused on data streams, the evaluation dataset has been injected in a way that simulates the production of a sensor device, i.e., on an observation-by-observation basis.

In order to fully characterise the algorithm proposed against the River's algorithms, two different analyses have been conducted. Firstly, we report the results of the detection capabilities of the algorithms, measured by metrics such as Precision or Recall. Secondly, we provide the results obtained in terms of the algorithms' time performance

Considering the evaluation of the detection capabilities, as mentioned above, the algorithms have been exposed to the same dataset, i.e., the same training values and the same evaluation data stream.

The following metrics are used for the assessment of the evaluated algorithms' detection capacity: Precision, Recall, F1-score and AUC-ROC. Precision quantifies the proportion of observations correctly identified as anomalies out of all the points flagged as anomalies by the model. Recall measures the fraction of anomalous observations correctly identified as such. F1-score provides the balance between Precision and Recall and is commonly employed as the preferred metric for algorithm comparison. Finally, AUC-ROC represents the likelihood of the model correctly identifying anomalies. The equations representing the first three metrics are given in (2), (3) and (4), respectively, where TP is the number of True Positives (anomalies correctly identified), FP is the number of False Positives (normal observations identified as anomalies) and FN is the number of False Negatives (anomalies not identified).

$$Precision = TP/(TP + FP) \qquad (2)$$

$$Recall = TP/(TP + FN) \qquad (3)$$

$$F1\ score = TP/(TP + (FP + FN)/2) \qquad (4)$$

Once the assessment methods are known, Table 3 and Table 4 present the numerical results obtained. The first table shows the detection results obtained for the three aforementioned variants of the proposed algorithm. In order to be able to properly illustrate and expose the different characteristics of the variants, several combinations of parameters are included in Table 3: *th* stands for threshold; and *a* is the EWMA importance factor. Alternatively, Table 4 shows the results obtained by the algorithms provided within the River framework, which will later be used for comparison.

Table 3: Detection metrics of the EWMA-based proposed algorithm.

	records	th	α	Precision	Recall	F1-score	AUC-ROC
Without update	12	1.1	0.2	0.0009	0.5	0.0017	0.3487
			0.8	0.0009	0.5	0.0017	0.3487
		1.2	0.2	0.0009	0.5	0.0017	0.3693
			0.8	0.0009	0.5	0.0018	0.3734
		1.3	0.2	0.0009	0.5	0.0019	0.3965
			0.8	0.001	0.5	0.0019	0.4028
With update		1.1	0.2	0.9524	1.0	0.9756	0.9999
			0.8	0.5132	0.975	0.6724	0.9869
		1.2	0.2	0.975	0.975	0.975	0.9875
			0.8	0.5652	0.975	0.7156	0.987
		1.3	0.2	1.0	0.95	0.9744	0.975
			0.8	0.7091	0.975	0.8211	0.9872
With smart update		1.1	0.2	0.9524	1.0	0.9756	0.9999
			0.8	0.9512	0.975	0.963	0.9875
		1.2	0.2	0.975	0.975	0.975	0.9875
			0.8	0.975	0.975	0.975	0.9875
		1.3	0.2	1.0	0.95	0.9744	0.975
			0.8	0.975	0.975	0.975	0.9875

As mentioned, Table 3 shows the Precision, Recall, F1-score and AUC-ROC results obtained for each of the combinations of the proposed algorithm presented. Previously, it was stated that the changes in the variants involved the updating of the model, which is based on EWMA. For that reason, the α parameter plays such an important role in obtaining a good detection. A higher value of α implies a higher weight or greater importance to the latest value of the time series, as was shown in Section 3.7.1.2. Taking this into account, it is evident that the performance of the second variant, "with update", is greatly affected by this parameter, since there are certain cases where the last value of the time series corresponds to an outlier, which distorts the results. In contrast, in the third variant, the model does not include these anomalies, as they have been already detected and removed, and is, therefore, relatively unaffected by the α parameter.

Table 4: Detection metrics of the River algorithms.

		Precision	Recall	F1-score	AUC-ROC
Gaussian Scorer	**wndw = 12 grc_prd = 1 th = 0.9**	0.0055	1.0	0.011	0.8797
Half Space Trees tree_ h = 4 wndw = 12	**n_trees = 10 th = 0.8**	0.4066	0.925	0.5649	0.9616
	n_trees = 10 th = 0.9	1.0	0.65	0.7879	0.9616
	n_trees = 15 th = 0.8	0.5692	0.925	0.7048	0.962
	n_trees = 15 th = 0.9	1.0	0.425	0.5965	0.7125
	n_trees = 20 th = 0.7	0.4368	0.95	0.5984	0.9742
	n_trees = 20 th = 0.8	0.9722	0.875	0.9211	0.9375
Local Outlier Factor	**n_neigh =12 th = 0.9**	0.3824	0.975	0.5493	0.9864
One Class SVM	**nu = 0.005 th = 0.9**	0.0032	0.35	0.0064	0.6033
Parameters note: *wndw* stands for window size; *grc_prd* stands for graph convolution predicted value; *th* stands for threshold; *n_trees* stands for number of trees; *tree_h* stands for tree heigh; *n_neigh* stands for number of neighbours; and *nu* is upper bound on the fraction of training errors.					

Based on the results reported, there is evidence showing that for the type of dataset used and its behaviour over time, the third variant of the proposed algorithm (i.e., the one "with smart update") shows the best performance metrics at all levels. This variant exhibit values higher than 0.95 in each metric.

Furthermore, beyond the results shown in Table 3, the best performance obtained by the "with smart update" variant is through the combination (records = 12, threshold = 1.3, α = 0.3), obtaining values of Precision = 1.0, Recall = 0.975, F1-score = 0.9873, and AUC-ROC = 0.9875. This combination is graphically depicted in Figure 43, where the test dataset is represented by a light green line, the anomalies introduced synthetically are plotted with dark green dots, and the anomalies detected by the algorithm are shown with red circles.

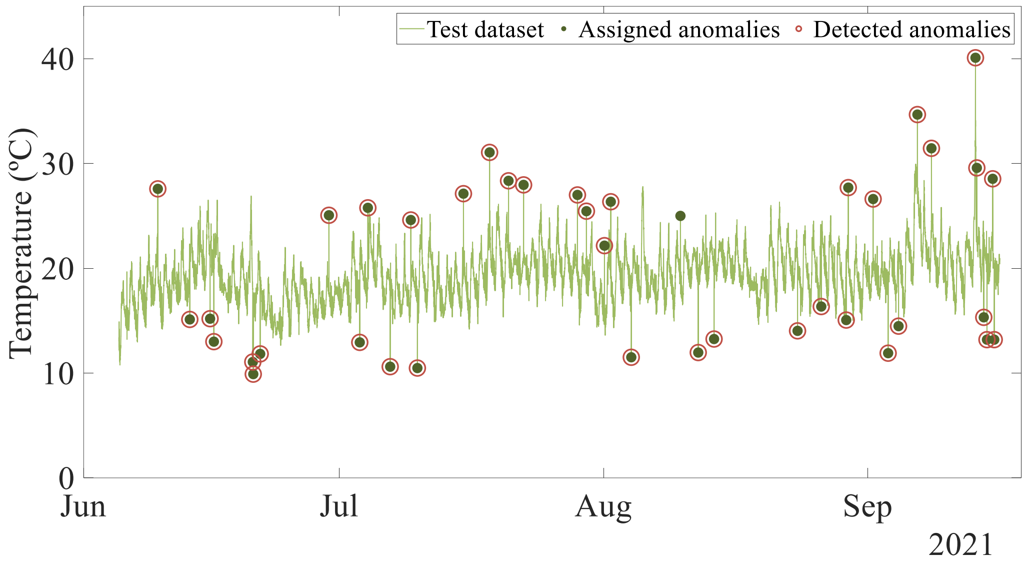


Figure 43: Anomaly detection performance of the third variant of the algorithm proposed, so-called "with smart update". Configuration parameters are: records = 12, threshold = 1.3, and α = 0.3.

Table 4 provides the performance results obtained with the algorithms available in River. Gaussian Scorer, Half Space Trees, Local Outlier Factor and One Class SVM algorithms have been used, all in their versions for data streams. The best combinations have been selected after a thorough process to analyse which parameters worked best for each algorithm. Note that the only algorithm that achieves proper detection values is Half Space Trees, with its best combination on (num_trees = 20, tree_height = 4, window_size = 12, threshold = 0.8) and metrics of Precision = 0.9722, Recall = 0.875, F1-score = 0.9211, and AUC-ROC = 0.9375. The remaining algorithms do not behave adequately with the dataset used.

Figure 44 displays the graphical results of the detection of the Half Space Trees algorithm when configured to use the parameters that showed best detection performance. As in Figure 43, the test dataset is represented by a light green line, the introduced anomalies are marked with dark green dots, and finally the anomalies detected by the algorithm appear as red circles.

Comparing both best combinations, the proposed algorithm using the "with smart update" variant and Half Space Trees, it can be seen that the former offers better and more stable detection values throughout the four metrics, as summarised in Table 3 and Table 4. Not only does this hold true in its best combination, but also over a range of them. Moreover, the difference in performances is illustrated graphically, i.e., which anomalies have been detected, which have been overlooked and which normal instances have been incorrectly identified as anomalies in Figure 43 and Figure 44.

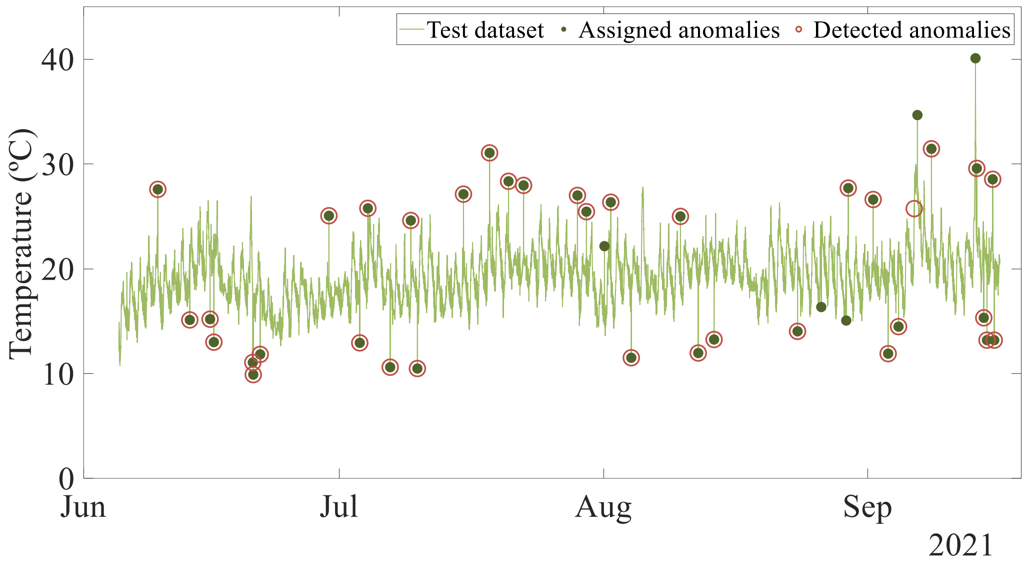


Figure 44: Anomaly detection performance of the Half Space Trees algorithm by River. Configuration parameters are: num_trees = 20, tree_height = 4, window_size = 12 and threshold = 0.8.

Concerning the second phase of the performance characterisation and comparison, it focuses on the computational time of the algorithms (i.e., the delay introduced in the detection of an outlier within the data stream). To this end, 300 Monte Carlo simulations have been performed for each of the algorithms configured with the parameters that made them have their best detection performance, monitoring both the training and assessment time of each of the incoming observations. All these tests have been carried out on an Ubuntu 20.04.6 LTS machine (2 CPU cores, 2.4 GHz clock, 16GB RAM). The resulting values are presented in Table 5. The total computation time values from the beginning of the training to the assessment of the last observation and the mean evaluation time per observation are provided. Note that the training dataset is composed of 45,000 measurements while the test dataset is composed of 30,000 records.

Table 5: Time performance results.

	Total time (s)	Mean detection time (ms)
Without update (records: 12, th: 1.6, a: 0.4)	5.9	0.197
With update (records: 12, th: 1.1, a:0.2)	127.76	4.259
With smart update (records: 12, th: 1.3, a:0.3)	133.91	4.463

	Total time (s)	Mean detection time (ms)
Gaussian scorer (wndw: 12, grc_prd: 1, th: 0.9)	5.11	0.17
Half Space Trees (n_trees: 20, tree_h: 4, wndw: 12, th: 0.8)	11.2	0.373
Local Outlier Factor (n_neigh: 12, th: 0.9)	366.09	12.2
One Class SVM (nu = 0.005)	5.46	0.182
Parameters note: *wndw* stands for window size; *grc_prd* stands for graph convolution predicted value; *th* stands for threshold; *n_trees* stands for number of trees; *tree_h* stands for tree heigh; *n_neigh* stands for number of neighbours; and *nu* is upper bound on the fraction of training errors.		

Table 5 demonstrates that, among the variants of the proposed algorithm, the computation time increases considerably in the two approaches that update the model. It is obvious as they do not keep the static photograph of the training dataset but include the new information and have to recalculate the model continuously. However, this comes as the trade-off for high precision. As for the algorithms provided by the River framework, Gaussian Scorer is the fastest, followed by One Class SVM. Nevertheless, these two algorithms together with Local Outlier Factor do not pose a strong challenge to the proposed algorithm due to their poor detection performance. In contrast, Half Space Trees has proper performance values, exhibiting an actually comparable trade-off with our proposed EWMA-based algorithm.

On the one hand, the proposed algorithm presented better detection values versus worse computation time, while Half Space Trees, the best of the River algorithms, showed lower detection delay versus poorer detection metrics. According to the needs and requirements of the use case, i.e., depending on whether the priority must lie in the speed of detection or in its accuracy, one or the other should be chosen.

5.3.6 Performance of synthetic data generation vs computational cost.

The SEDIMARK Data Augmentation module features the ability to generate purely synthetic datasets which can then be used to train downstream machine learning models, providing a useful alternative to data providers who might not wish to share their raw data within the SEDIMARK marketplace. The SEDIMARK Data Augmentation module contains a number of methods, however as these methods generally rely upon expensive neural network models as their generation backbone, the synthetic data generation incurs a large computational cost. In this section, we briefly compare the computational cost and the output performance of a downstream machine learning model on time series data. To this end, we train a machine learning model on the time series data and then compare performance when we train further machine learning models on synthetic datasets generated with increasing computational cost with the VAE component of the augmentation module. We vary the computational cost by increasing the number of training epochs for which the synthetic data generation is run. We employ the stock and energy time series datasets used by [92], and adapt the evaluation code and procedure from [93], using the S4 [91] model as the baseline for downstream performance. We then report the mean squared error of the baseline trained on both real and synthetic data as we increase the number of synthetic training epochs. Results are shown in Figure 45.

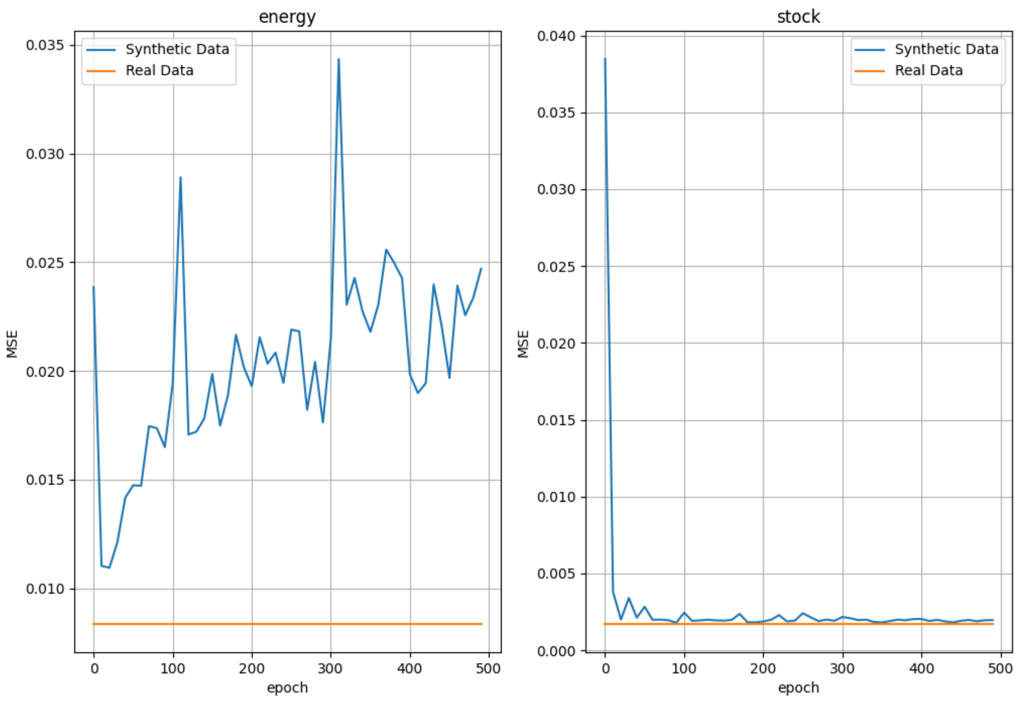


Figure 45: MSE scores for both real and synthetically trained models, when the number of epochs to train the synthetic model is increased.

On both datasets we observe that with increased computation, the predictive power of the synthetic data approaches that of the original training data. On the Energy dataset, this effect happens in only a small number of epochs, after which additional computation doesn't help to further increase its predictive power and in fact overfits the model. On the Stock dataset, the trend continues downwards. This highlights the necessity of overseeing the synthetic data generation process, perhaps by employing a validation procedure to guide early stopping.

5.3.7 Analysis of Low-rank factorisation

The goal of this section is to investigate the application of the low-rank adaptation (LoRA) in the fine-tuning process of SEDIMARK's recommender module (as described in Section 4.3.5). The goal of applying the low-rank representation is to make fine-tuning process more efficient by drastically decreasing the number of trainable parameters.

Experimental setup:

In this experiment we use the Bi-Encoder model [95], whose goal is to retrieve relevant datasets based on the given query. This is achieved using two separate sentence encoders from the transformer model. The model is then trained using a contrastive loss and hard negatives selected using BM25 retrieval [96] from the training dataset. To add LoRA to the Bi-Encoder model we use the peft (Parameter-Efficent Fine-tuning) library [97]. We compare the Vanilla (i.e. the full model) vs. LoRA approaches studying (i) the model performance and (ii) the parameter efficiency in both approaches. For (i) we report the test accuracy selected from the best accuracy reported on the validation set; and for (ii) we report the number of trainable parameters. The implementation of LoRA-based bi-encoder is available in our GitHub repository [98].

Dataset:

In this experiment we use the Datafinder dataset [99] containing train queries generated using the Galactica LLM incorporating information from abstracts of relevant research papers. The test queries were generated by human annotators. Additionally, we further divide the train set into train/validation sets at the ratio of 90/10 to fine-tune the Bi-Encoder model. Overall, this dataset contains 2687 datasets, 1024 train queries and 257 test queries.

Results:

Table 6: Results for low-rank factorization.

Model	Precision@10 (P@10)	# Trainable Params
Bi-Encoder: Full	0.168	~22M
Bi-Encoder: LoRA	0.162 ↓3.57%	~110K ↓99.59%

The results in Table 6 illustrate a massive saving in the computational resources (memory and compute cost) required for fine-tuning the Bi-Encoder model if LoRA is applied. The accuracy is measured in terms of Precision@10, which is the proportion of recommended items in the top-10 set that are relevant for the user. In particular, applying LoRA decreases the number of trainable parameters by more than 21M, which is a decrease of over 99% resulting in only slightly decreased accuracy of 3.57%.

5.3.8 Performance Evaluation of Coreset and Distillation Methods

In this section, we investigate the effectiveness of Coreset Selection and Data Distillation methods for improving efficiency in SEDIMARK. We aim to evaluate these methods in terms of accuracy, runtime, data size, and overall efficiency. The results of these experiments will provide practical guidance for deploying scalable, resource-aware machine learning workflows, directly supporting SEDIMARK’s objective of enabling high-quality and energy-efficient data marketplaces.

5.3.8.1 Experimental Setup

Dataset: In our experiments, we use two widely used public datasets to simulate different dataset sizes that might be encountered in real-world scenarios: Statlog (large) [108] and Water Potability (medium) [109].

- The Statlog (Landsat Satellite) Data Set contains multispectral values of pixels in satellite images of the Earth’s surface. The primary objective is to classify each pixel into one of six land cover types (such as soil, vegetation, or water) based on 36 spectral attributes. The dataset consists of 6,435 instances and is widely used as a benchmark for multi-class classification tasks, especially in the remote sensing and environmental monitoring domains.

- The Water Potability dataset focuses on water quality assessment. Each sample consists of nine physicochemical properties (such as pH, hardness, solids, and turbidity), and the target variable indicates whether the water is potable (safe for human consumption). The dataset contains 3,276 instances and is commonly employed for binary classification tasks, particularly in environmental data analysis and public health applications.

  These datasets enable us to systematically evaluate the performance and efficiency of the proposed methods across different domains and data scales.

Baseline Methods: For data distillation, we include:

- PCA K-means [110] Applies K-means clustering in the principal component space to capture key data variations.

For **Coreset Selection**, we include:

- Random Sampling: Randomly selects a subset of data points from the original dataset.
- K-means [111]: Clusters data using K-means and selects representative points from each cluster.
- Greedy k-Center [112]: Selects data points that maximize coverage of the dataset by iteratively choosing the farthest points.
- Facility Location [113]: Selects a subset of points that best represent the entire dataset based on facility location optimization.
- Influence [114]: Picks samples with high influence scores, aiming to retain critical points for model performance.
- Random Projection K-means [115]: Applies K-means clustering after projecting data to a lower-dimensional space.
- DBSCAN [116]: Uses density-based clustering to identify representative points.
- Gradient Coreset [117]:: Utilises gradient-based selection to retain samples that most impact model updates.
- DPP Diversity [118] Maximises diversity in the selected subset using Determinantal Point Processes.

By including these diverse algorithms, our evaluation provides a comprehensive comparison of mainstream distillation and coreset strategies, covering a broad spectrum from random and clustering-based methods to optimisation-driven and diversity-focused approaches.

To ensure the reliability and statistical significance of our results, each experiment is independently repeated 10 times. For every method and dataset combination, we report the average value of each evaluation metric across these 10 runs. For all experiments, each dataset is randomly split into training and test sets using an 80/20 ratio. Within the training set, we further apply coreset selection or data distillation to obtain reduced subsets at varying sizes: 1%, 5%, 10%, 15%, and 20% of the training data. For the downstream classification task, we employ a Support Vector Machine (SVM) as the classifier. This approach enables us to systematically evaluate the impact of subset size on model performance. By averaging over multiple runs and systematically varying the subset size, we mitigate the effects of randomness and provide a robust assessment of the comparative performance and efficiency of different methods.

Evaluation Metrics: To facilitate a fair and intuitive comparison across different subset selection and data distillation methods, we report all performance results in terms of relative metrics. These relative metrics allow us to directly assess how well each method preserves model performance or improves efficiency compared to using the full dataset.

Each relative metric is calculated by dividing the value obtained from the model trained on the coreset or distilled subset by the value obtained from the model trained on the full dataset. The result is typically expressed as a percentage. For example, relative accuracy indicates the proportion of accuracy retained when using a subset instead of the entire dataset. In this report, the same approach is applied to relative ROC-AUC, relative training time, and other metrics, providing a straightforward way to compare the effectiveness and efficiency of different methods.

5.3.8.2 Experimental Results

Figure 46 and Figure 47 show the relative classification accuracy achieved by different subset selection and distillation methods as the subset size varies on the Water Potability and Statlog datasets, respectively.

On the Water Potability dataset (Figure 46), excellent classification performance can be achieved even with very small subsets. For example, when using just 5% of the training data, which corresponds to approximately 131 samples, most methods are able to achieve nearly the same accuracy as using the full dataset. This suggests that, for this binary classification problem, both coreset and distillation methods are highly data-efficient, and that increasing the subset size beyond this point provides little additional benefit.

On the Statlog dataset (Figure 47), most selection methods continue to perform well as the subset size increases. Random sampling, k-means, and pca_kmeans maintain consistently high relative accuracy, indicating that they generalize effectively even in a more complex, multi-class scenario. However, some methods such as dbscan, gradient, and random_projection show reduced accuracy when the subset size is very small, which may be due to difficulties in adequately representing all classes under these conditions. Facility and DPP methods are not shown in the Statlog results because, at small subset sizes, they often produced subsets containing only a single class. This is a result of their diversity-based selection strategies, which can sometimes favour the dominant class or the densest regions of the feature space when the subset size is much smaller than the number of classes. As a consequence, these subsets are not suitable for multi-class model training, which limits the applicability of these methods in such scenarios.

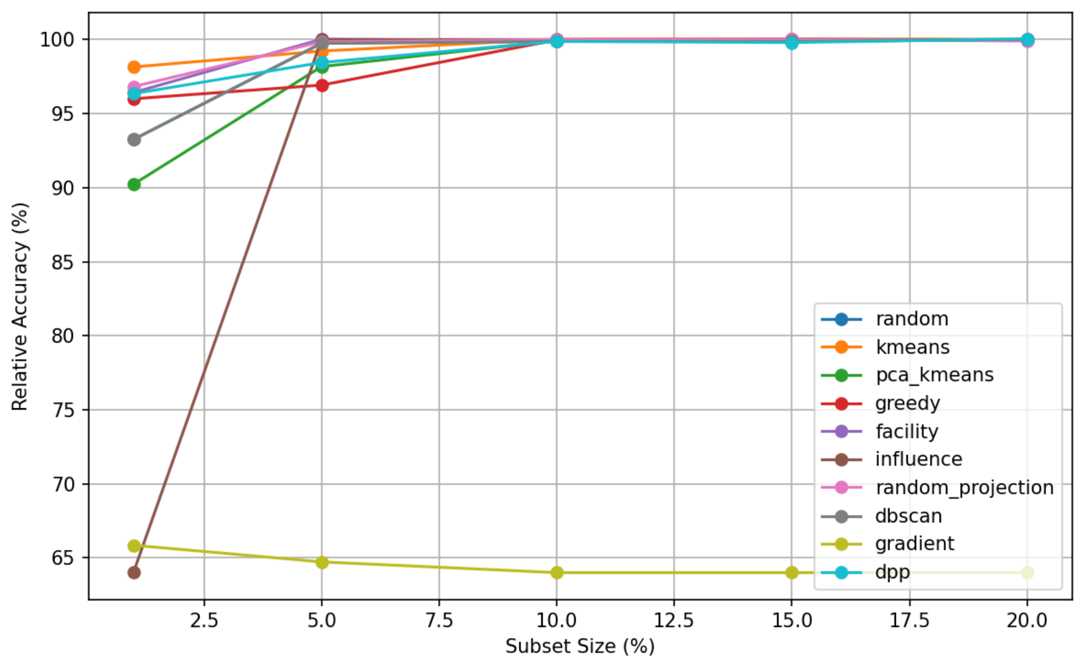


Figure 46: Relative classification accuracy (%) of models trained on coreset/distilled subsets versus the full dataset, across different subset sizes and selection methods on Water Potability.

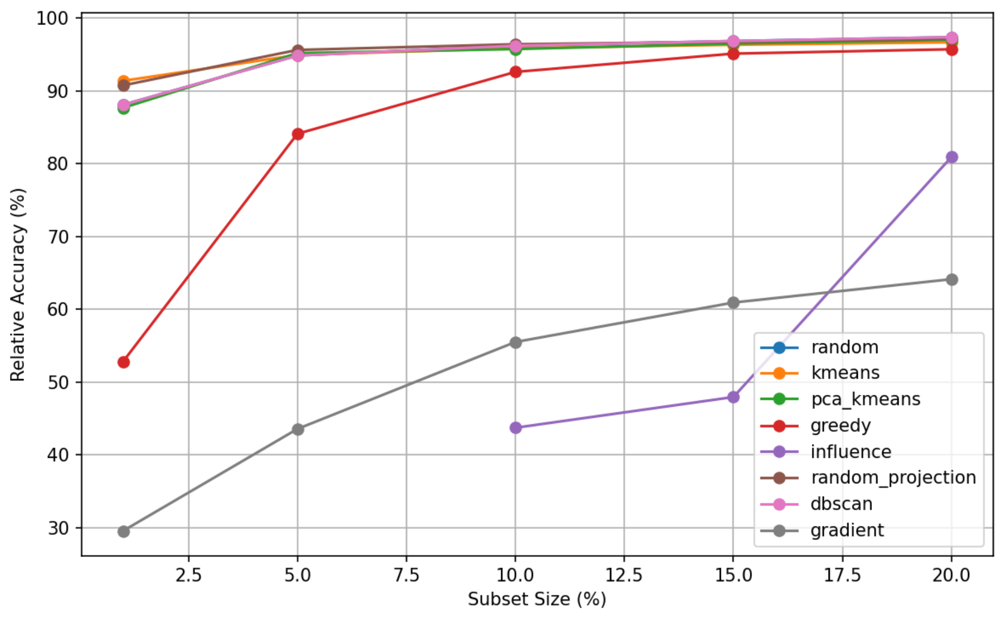


Figure 47: Relative classification accuracy (%) of models trained on coreset/distilled subsets versus the full dataset, across different subset sizes and selection methods on Statlog.

Figure 48, Figure 49 (Water Potability) and Figure 50, Figure 51 (Statlog) present the absolute and relative training time required by models trained on coreset or distilled subsets of different sizes. The results indicate a clear trend: as the subset size increases, training time increases correspondingly for all methods, which is expected. However, when compared with the accuracy results discussed previously, it becomes evident that substantial reductions in training time can be achieved without sacrificing classification performance.

For the Water Potability dataset, most methods are able to achieve nearly 100% of the full-dataset accuracy using only 5% of the training data, as shown in Figure 46. At the same time, Figure 49 shows that training with such a small subset typically requires less than 1% of the training time compared to using the full dataset. Even when increasing the subset size to 10% or 20%, the relative training time remains a small fraction of the total, yet the classification accuracy is almost indistinguishable from that obtained with the entire dataset. This demonstrates the high data efficiency of subset selection and distillation methods for this binary classification problem: excellent performance can be achieved with dramatically reduced computational cost.

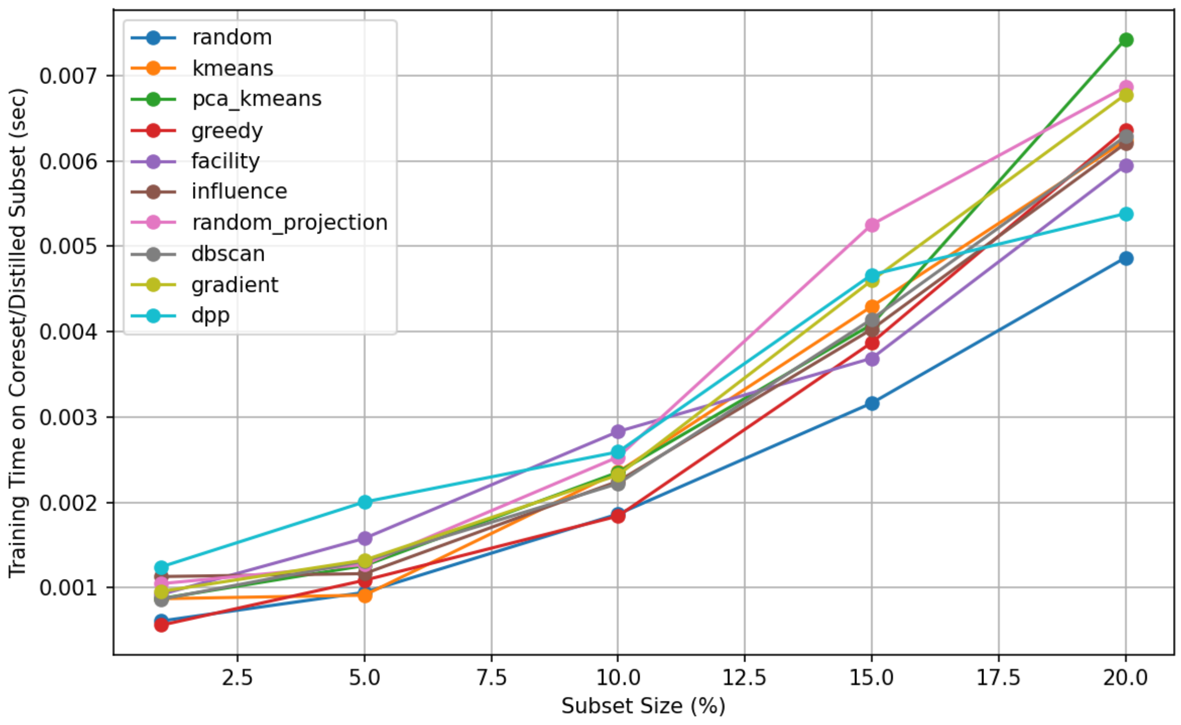


Figure 48: Absolute training time of coreset/distilled subsets on Water Potability.

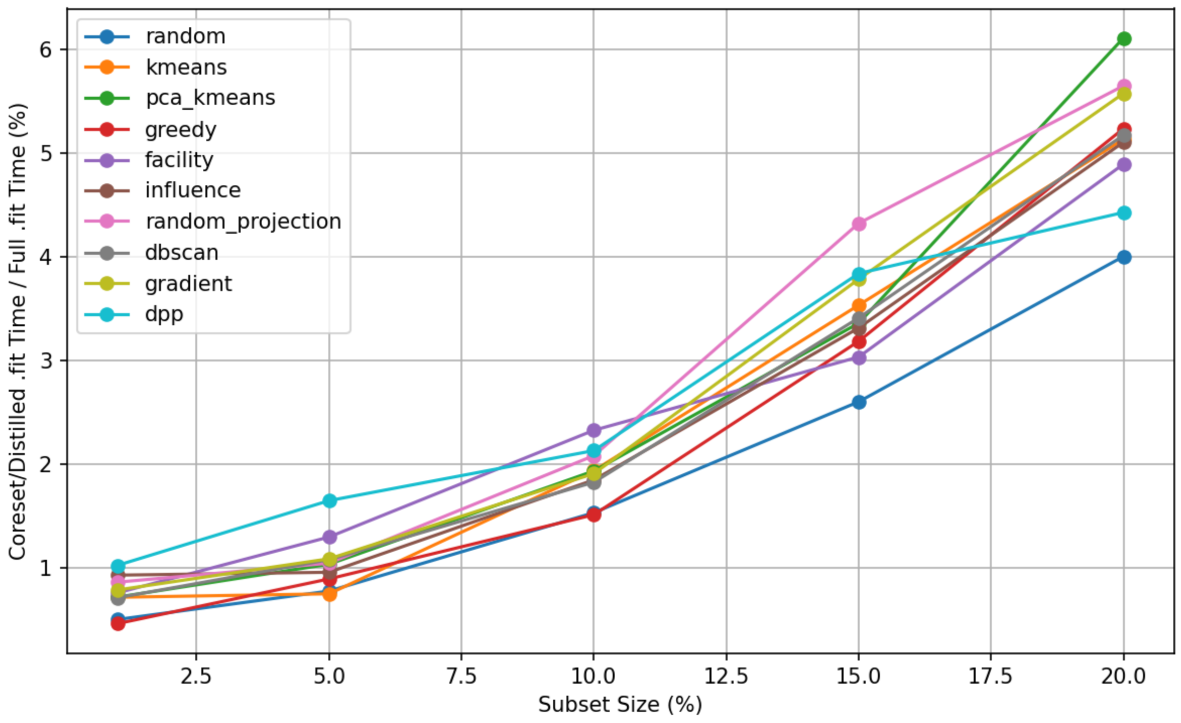


Figure 49: Relative training time (%) of coreset/distilled subset versus full dataset training time on Water Potability.

Similar trends are observed for the Statlog dataset, although the relationship between subset size, accuracy, and training time can be influenced by the multi-class nature of the problem. As seen in Figure 47, most methods (such as random, k-means, and pca_kmeans) are able to retain high relative accuracy when the subset size reaches 15–20%. According to Figure 51, at these subset sizes, the relative training time is only about 4–5% of that needed for the full dataset. This means that it is possible to reach almost full classification performance while using just a small fraction of the total training time. For smaller subset sizes, some methods show lower accuracy, indicating that sufficient class diversity is important in multi-class tasks, but the computational savings remain substantial.

Overall, these results show that by using only a small proportion of the training data, it is possible to achieve both significant computational efficiency and high model performance. This is particularly valuable in practical applications where training time or computational resources are limited.

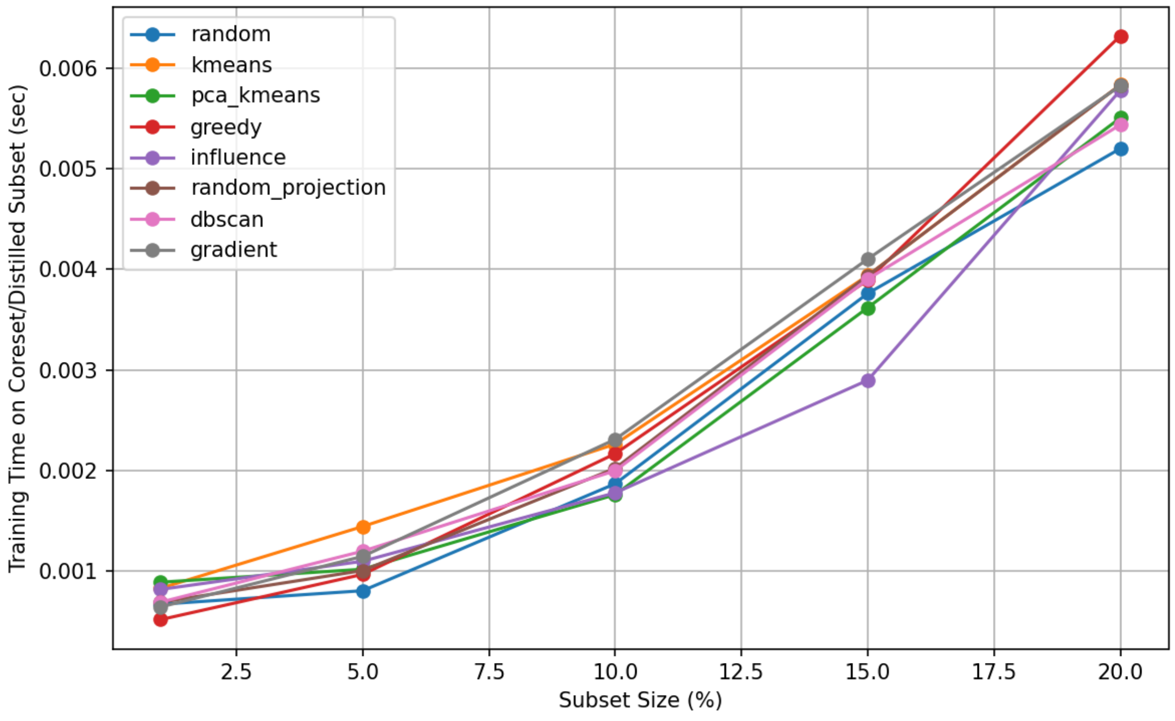


Figure 50: Absolute training time of coreset/distilled subsets on Statlog.

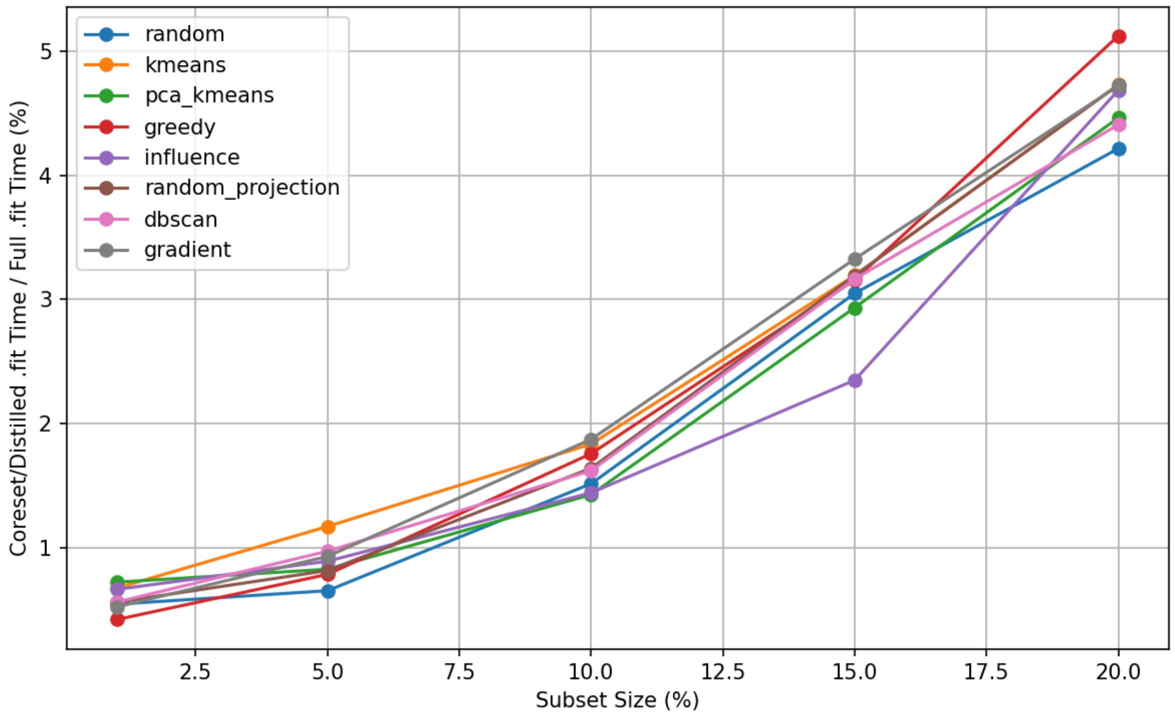


Figure 51: Relative training time (%) of coreset/distilled subset versus full dataset training time on Statlog.

5.3.9 Performance of Time Series Forecasting with Optimisation Techniques

The goal of this section is to analyse the performance of time-series multivariate forecasting approach, CrossFormer, with diverse optimisation techniques. The performance evaluation uses the evaluation feature designed within the CrossFormer component, including MAE, MAPE, MSE, MSPE, RMSE and SCORE.

Performance of Forecasting with Quantization (FLOPS)

As tensor calculations cost a significant amount of computational resources in deep learning based time series forecasting approaches, different float precision settings of tensor may lead to varied performance. Therefore, the model has been tested on the weather use case with several float precision settings, comparing their forecasting performance. The result is given in Table 7.

Table 7: Comparisons of performance over diverse float precision settings.

Float Precision	MAE	MSE	RMSE
Float 64	3.2644	20.2673	4.2437
Float 32	3.9656	27.0471	4.8866
Float 16	5.3205	99.1768	6.9796

Forecasting with Pruning Techniques:

Building on the pruning techniques introduced in Section 4.3.3, we present the results of applying pruning techniques to the CrossFormer model. The primary objective was to evaluate how channel pruning affects the model's size, the number of parameters, floating-point operations per second (FLOPs), memory usage, and performance metrics.

Structured (Channel) Pruning:

The results in Table 8 demonstrate that channel pruning is an effective method for reducing the computational complexity of the CrossFormer model while also achieving substantial memory savings. By targeting entire channels for pruning, this approach results in a model that is not only smaller and faster but also uses significantly less memory, making it highly appropriate for deployment in environments with limited resources.

Table 8: Impact of Channel Pruning on CrossFormer Model.

	Parameters	Size (MB)	FLOPS (GFLOPS)	Score
Original Model	6.35 M	24.24	0.382	0.0932
Channel Pruned Model	3.04 M	11.62	0.181	0.0925

Unstructured Pruning:

The findings from layer specific pruning in Table 9 highlight the effectiveness of this method in maintaining a balance between reducing the computational load and preserving model performance. Unlike uniform or unstructured pruning, where a fixed ratio or individual weights are pruned across all layers, layer specific pruning allows for more nuanced and strategic reduction of parameters. This is evident in the relatively small performance degradation observed even after reducing the model's size and parameter count by more than half.

By leveraging the Coefficient of Variation (CoV) and other weight statistics to determine layer specific pruning ratios, this approach ensures that critical layers are preserved while less important ones are pruned more aggressively. The success of this strategy is reflected in the high scores, which are only slightly lower than those of the unpruned model. By selectively pruning layers based on their importance, this method allows for substantial reductions in model size without sacrificing the accuracy needed for effective video quality assessment

Table 9: Impact of Unstructured Pruning on CrossFormer Model.

	Parameters	Size (MB)	FLOPS (GFLOPS)	Score
Original Model	6.35 M	24.24	0.382	0.0932
Unstructured Pruned Model	2.98 M	11.37	0.383	0.0962

Hybrid pruning:

The results in Table 10 demonstrate the effectiveness of combining Layer specific and Channel Pruning as a strategy for optimizing neural networks. This approach allows for a more targeted reduction of model complexity by applying different pruning techniques that complement each other. The significant reduction in model size, parameter count, and FLOPs, combined with the retention of high scores, highlights the potential of this method for creating efficient, high-performance models.

Table 10: Impact of Hybrid Pruning on CrossFormer Model.

	Parameters	Size (MB)	FLOPS (GFLOPS)	Score
Original Model	6.35 M	24.24	0.382	0.0932
Hybrid Pruned Model	1.45 M	5.53	0.181	0.0927

Key Observations:

- Channel Pruning reduces model size by ~48% (24.24MB → 11.62MB) and FLOPs by ~53% (0.382 → 0.181 GFLOPs) while maintaining competitive accuracy (6.526 MAE vs. original 6.630). Ideal for edge deployment due to dense weights.
- Unstructured Pruning achieves ~47% size reduction (24.24MB → 11.37MB) but no FLOPs reduction (0.382 GFLOPs retained). Improves MSE (128.03 vs. 132.62) but requires sparse-aware hardware for acceleration.
- Combined Pruning combined with Unstructured Pruning delivers the highest compression (78% fewer parameters, ~77% smaller size) and 53% fewer FLOPs, with the best MAE (6.517) but slightly worse MSE (139.10). Balances sparsity and FLOPs reduction.

5.3.10 Performance of Offering Generation with Optimisation Techniques

The Offering Generator implements knowledge distillation to address efficiency challenges by enabling smaller, more efficient models to learn from the capabilities of much larger teacher models. This approach achieves comparable performance with significantly reduced computational requirements. A compute-efficient model such as Qwen 2.5-3B serves as the smaller student model, trained through knowledge distillation from a larger counterpart.

To assess performance and efficiency, five optimisation techniques as discussed in Section 4.3 were applied to both Qwen 2.5-3B and the larger Qwen 2.5-7B variant:

- **Baseline (FP16):** Standard 16-bit floating-point precision model
- **8-bit Quantization:** Reduced precision model using 8-bit integers
- **4-bit Quantization:** Further reduced precision using Normalized Float 4 (NF4) format
- **LoRA:** Low-Rank Adaptation for parameter-efficient fine-tuning
- **QLoRA:** Combination of 4-bit quantization and LoRA for maximum efficiency

To illustrate the impact of these techniques, comparative results for both model sizes Qwen 2.5-3B and Qwen 2.5-7B are presented in Table 11 and Table 12, respectively. These results allow for analysis of how optimisation effectiveness scales with model size while maintaining a consistent architecture.

5.3.10.1 Memory Efficiency and Parameter Reduction During Training

The optimisation techniques yielded significant improvements in memory efficiency and trainable parameter counts. 8-bit quantization achieved moderate memory reduction (42–45%) while maintaining the full parameter count of the baseline model, thereby retaining model capacity with lower memory usage. In contrast, 4-bit quantization resulted in dramatic memory savings (66–67%) along with a reduction in effective parameter count, making it the most compact configuration. LoRA preserved the overall model size but drastically reduced the number of trainable parameters to just 0.1–0.2% of the total, allowing for highly efficient fine-tuning. QLoRA combined the benefits of 4-bit quantization and LoRA, reducing memory usage by 46–55% while keeping the number of trainable parameters minimal, offering an optimal setup for efficient training.

5.3.10.2 Model Size Scaling

The comparison between the 3B and 7B variants revealed key scaling behaviours across all optimisation techniques. Memory efficiency remained consistent across model sizes, with 8-bit and 4-bit quantization achieving similar percentage reductions approximately 42–45% and 66–67%, respectively. Parameter efficiency improved with scale, as evidenced by LoRA’s trainable parameter share dropping from 0.2% in the 3B model to 0.1% in the 7B model, highlighting increased relative benefits for larger models. Relative size reductions were also stable across both variants, with roughly 7× reduction achieved through 4-bit quantization and 2× through 8-bit. Importantly, absolute memory savings were greater in the larger model, with the 7B configuration saving 9.1 GB using 4-bit quantization compared to 4.0 GB in the 3B model.

5.3.10.3 Context Curriculum Learning

To enhance robustness under variable context availability common in real-world deployment scenarios a context curriculum learning strategy was applied. Training began with full context access and progressively reduced it in successive stages, using context dropout rates of 30%, 60%, and 90%. This gradual reduction improved the model's ability to generate structurally valid JSON-LD even when context was limited. Notably, the 7B model maintained superior structural coherence under high dropout conditions, indicating stronger contextual reasoning capabilities despite optimisation

5.3.10.4 Results

Selecting appropriate optimisation techniques in this context enables a reduction in computational and environmental costs while preserving the semantic richness and structural integrity of marketplace offerings.

Table 11: Optimisation Results for Qwen 2.5-7B Model.

Qwen 2.5-7B Model					
Technique	Parameters	Trainable Parameters	Model Size (MB)	Memory Usage (MB)	Size Reduction
Original Model	6.35 M	24.24	0.382	0.0932	-
Baseline	7,615,616,512	7,615,616,512 (100.0%)	0.382	14,587.4	1.00×
8-bit	7,615,616,512	1,090,199,040 (14.3%)	2,943.0	8,418.3	2.00×
4-bit	4,352,972,288	1,090,199,040 (25.0%)	810.0	5,473.6	7.00×
LoRA	7,625,709,056	10,092,544 (0.1%)	5,900.0	14,625.9	1.00×
QLoRA	4,363,064,832	10,092,544 (0.2%)	3,254.0	7,851.3	1.75×

Table 12: Optimisation Results for Qwen 2.5-3B Model

Qwen 2.5-3B Model					
Technique	Parameters	Trainable Parameters	Model Size (MB)	Memory Usage (MB)	Size Reduction
Baseline	3,085,938,688	3,085,938,688 (100.0%)	5,886.0	6,007.1	1.00×
8-bit	3,085,938,688	311,314,432 (10.1%)	2,943.0	3,085,938,688	2.00×

Qwen 2.5-3B Model					
Technique	Parameters	Trainable Parameters	Model Size (MB)	Memory Usage (MB)	Size Reduction
4-bit	1,698,672,640	311,314,432 (18.3%)	810.0	2,021.7	7.27×
LoRA	3,093,311,488	7,372,800 (0.2%)	5,900.0	6,035.2	1.00×
QLoRA	1,706,045,440	7,372,800 (0.4%)	3,254.0	2,730.0	1.81x

5.3.11 Performance of AutoML for data streams

We use model execution time as a proxy for energy consumption: on a CPU, power usage is proportional to the time to finish the task.

We compare classification algorithms on several classic datasets, both real-world (Covertype [131], Electricity [132], Sensors [133]) and synthetic (Banana, LED, RBF, SEA). LED, RBF, and SEA also had variations with induced concept drift, both slow and fast.

We compare the models OnlineSMAC (ours), OnlineSMAC Ens (ours), Automated Streaming Machine Learning [134], Auto Class [135], Adaptative Random Forest [136], Streaming Random Patches [137], and Leveraging Bagging [138] (respectively abbreviated as OSMAC, OSMAC Ens, ASML, AC, ARF, SRP, and LB). OSMAC, ASML, and AC are AutoML models, the others are traditional ensemble models.

We measured the overall accuracy of the models in test-and-train mode and the time they needed to process the datasets.

Figure 52 and Figure 53 report the average of the metrics over the datasets.

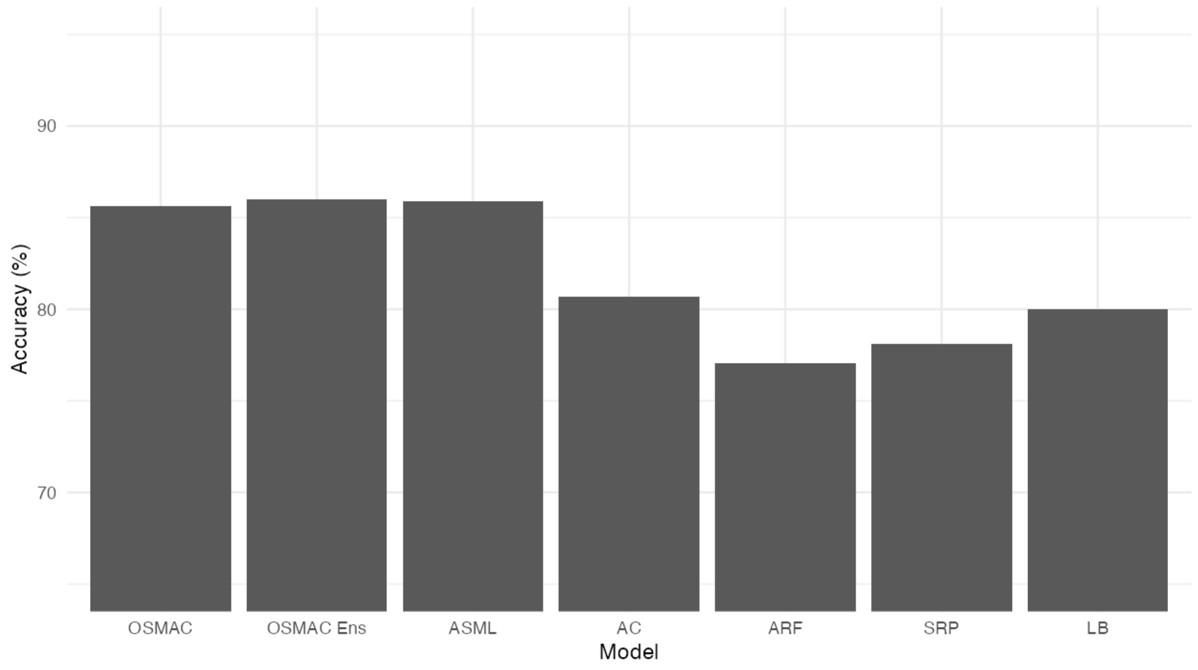


Figure 52: Average accuracy of the models over the test datasets (note that the Y axis is truncated for legibility). Higher is better.

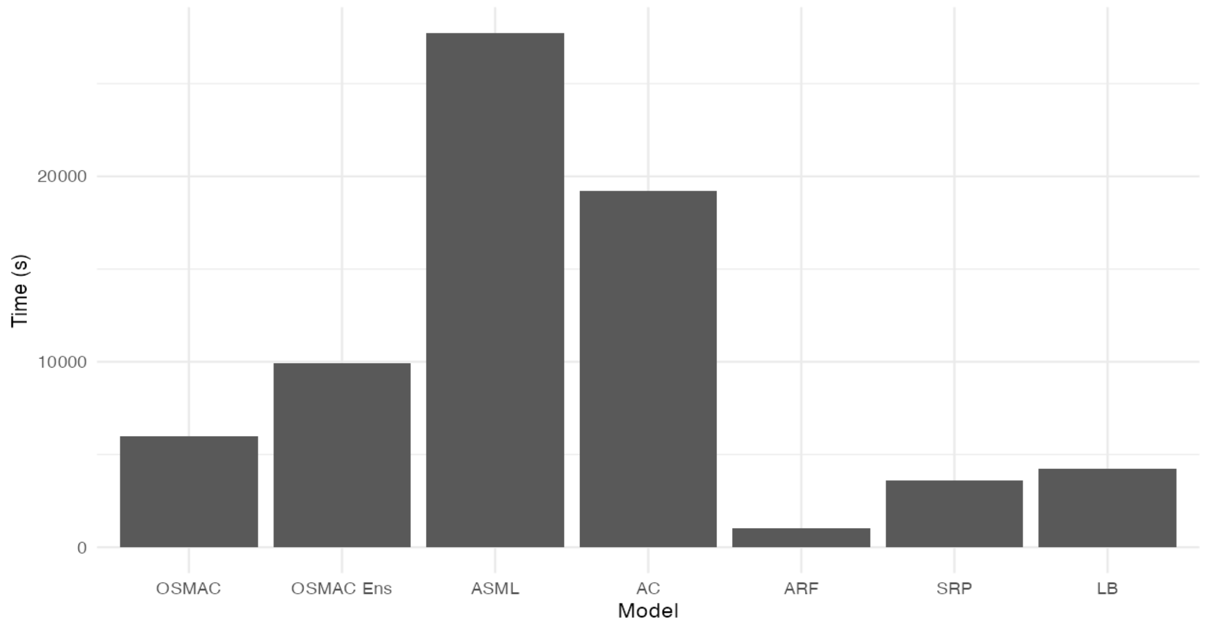


Figure 53: Average processing time (in seconds) of the models over the test datasets. Lower is better.

For the accuracy, OSMAC, OSMAC Ens and ASML are equivalent. OSMAC Ens is the best by an unsignificant margin.

For the processing time, ARF, SRP, and LB are the best-performing models in this order thanks to their relative simplicity. ASML and AC are the worst models in this comparison, AC needing almost twice as long as OSMAC Ens, and ASML two and a half.

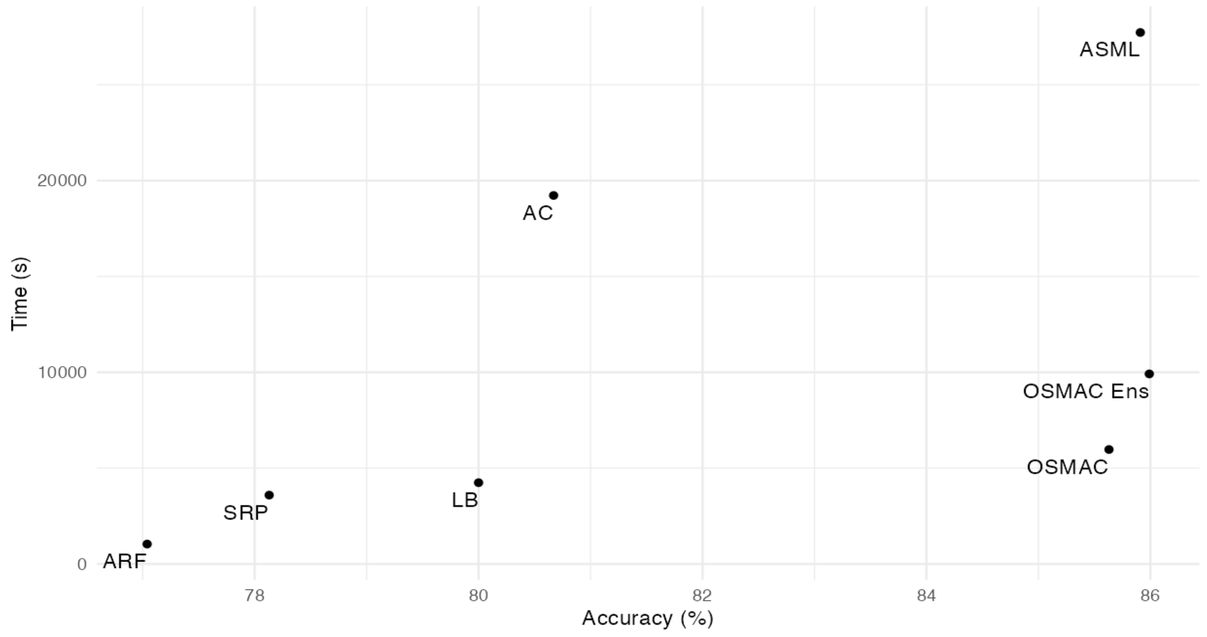


Figure 54: Trade-off between accuracy and processing time. Models on the lower right are better.

Figure 54 shows the trade-off between accuracy and runtime. The models OSMAC and OSMAC Ens are on the optimality frontier, together with ARF.

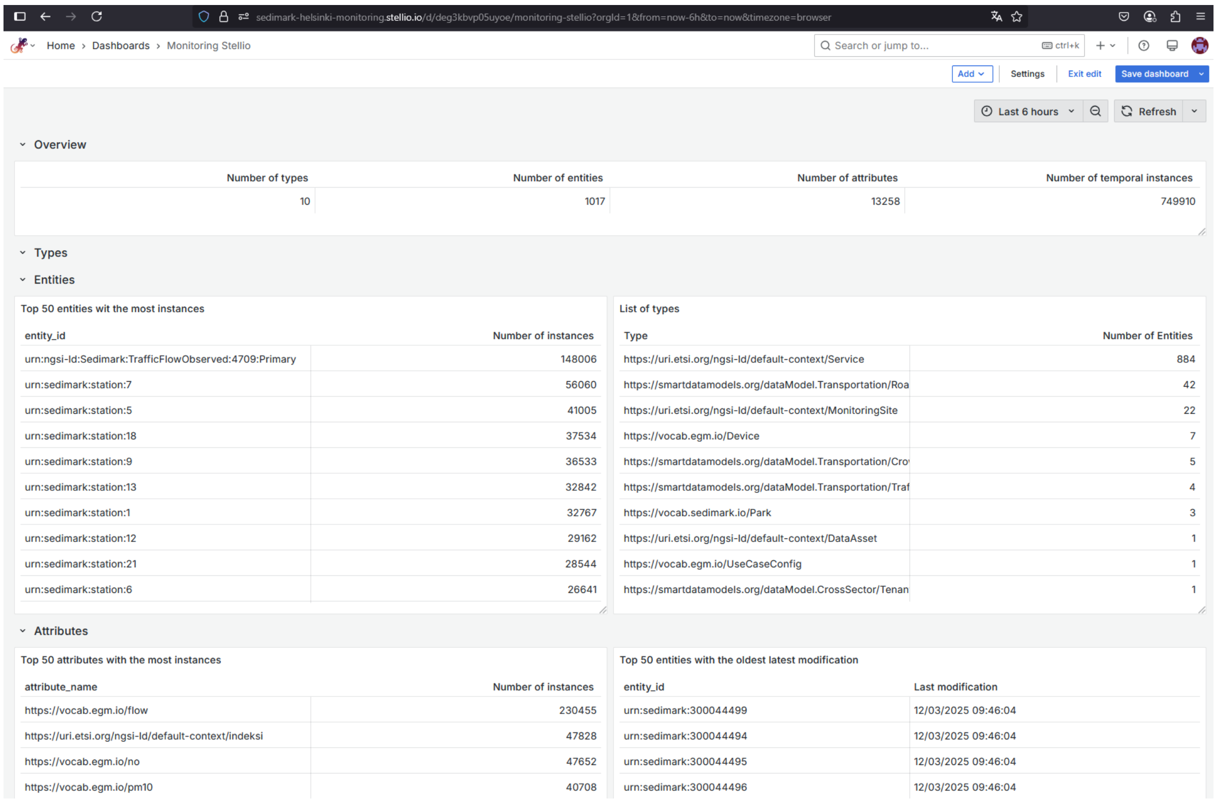
The ensemble, non-adaptative methods are faster and use less energy, but they have a lower overall accuracy. In contrast, ASML is very accurate but also slow.

5.4 Data storage efficiency analysis

This section investigates the different method to increase data storage efficiency. The experiments and analysis are performed using Stellio, a NGSI-LD broker that store data and enable easy interoperability as described in SEDIMARK_3.3 section 6.2.3.

5.4.1 Current data storage evaluation cost

In NGSI-LD the data is stored in entities. Each entity contains a list of attributes which can have temporal evolution. To know what data is stored on the broker and detect inconsistent behaviour we have developed a dashboard that let us follow in real time the number of entity, attribute and temporal instances (each temporal instance represents a point in time of an attribute) in the broker.



sedimark-helsinki-monitoring.stellio.io/d/deg3kbvp05uyoe/monitoring-stellio?orgId=1&from=now-6h&to=now&timezone=browser
Home › Dashboards › Monitoring Stellio
Search or jump to...
ctrl+k
Add
Settings
Exit edit
Save dashboard
Last 6 hours
Refresh

Overview

Number of types	Number of entities	Number of attributes	Number of temporal instances
10	1017	13258	749910

Types

Entities

Top 50 entities wit the most instances

entity_id	Number of instances
urn:ngsi-ld:Sedimark:TrafficFlowObserved:4709:Primary	148006
urn:sedimark:station:7	56060
urn:sedimark:station:5	41005
urn:sedimark:station:18	37534
urn:sedimark:station:9	36533
urn:sedimark:station:13	32842
urn:sedimark:station:1	32767
urn:sedimark:station:12	29162
urn:sedimark:station:21	28544
urn:sedimark:station:6	26641

List of types

Type	Number of Entities
https://uri.etsi.org/ngsi-ld/default-context/Service	884
https://smartdatamodels.org/dataModel.Transportation/Roa	42
https://uri.etsi.org/ngsi-ld/default-context/MonitoringSite	22
https://vocab.egm.io/Device	7
https://smartdatamodels.org/dataModel.Transportation/Cro	5
https://smartdatamodels.org/dataModel.Transportation/Traf	4
https://vocab.sedimark.io/Park	3
https://uri.etsi.org/ngsi-ld/default-context/DataAsset	1
https://vocab.egm.io/UseCaseConfig	1
https://smartdatamodels.org/dataModel.CrossSector/Tenan	1

Attributes

Top 50 attributes with the most instances

attribute_name	Number of instances
https://vocab.egm.io/flow	230455
https://uri.etsi.org/ngsi-ld/default-context/indeksi	47828
https://vocab.egm.io/no	47652
https://vocab.egm.io/pm10	40708

Top 50 entities with the oldest latest modification

entity_id	Last modification
urn:sedimark:300044499	12/03/2025 09:46:04
urn:sedimark:300044494	12/03/2025 09:46:04
urn:sedimark:300044495	12/03/2025 09:46:04
urn:sedimark:300044496	12/03/2025 09:46:04

Figure 55: Dashboard following the data deployed for Helsinki datasets

By directly looking at the database storage we have been able to estimate the following storing costs

- **an entity:** 2500 bytes (1017 entity for a total of 2.5MB)
- **an attribute:** 1100 bytes (13258 attributes for a total of 14.6MB)
- **a temporal instance:** 650 bytes (749910 instances for a total of 487.4MB)

Some temporal representations are measured every 15 minutes for multiple consecutive years. This explains the huge number of temporal instances present and consequently the fact that most of the storage cost is taken by the temporal representation of attributes. Since most of the cost of storage is taken by the temporal representation we will focus on optimizing it in the next points.

5.4.2 Increasing data storage efficiency by down sampling

Down sampling is a technique used to reduce the number of temporal instances by decreasing the frequency of recorded values. Since most of the storage cost is due to temporal data, down sampling offers a promising solution to significantly reduce the amount of data stored while keeping the general trend of the attribute evolution.

Several strategies can be considered:

- **uniform down sampling:** storing one value every n interval (e.g., every hour instead of every 15 minutes).
- **aggregation-based down sampling:** storing aggregated values such as the average over a fixed time window.

These strategies allow for a trade-off between data volume and temporal precision. For example, reducing the sampling rate from 15 minutes to 1-hour results in a 75% storage gain. Aggregation-based down sampling needs some additional calculation but may preserve more meaningful statistics.

In some use cases, fine-grained temporal data is essential and cannot be reduced without losing important information. In such cases, down sampling is not applicable. However, when high precision is not required, down sampling can lead to significant storage savings by reducing the number of stored temporal instances.

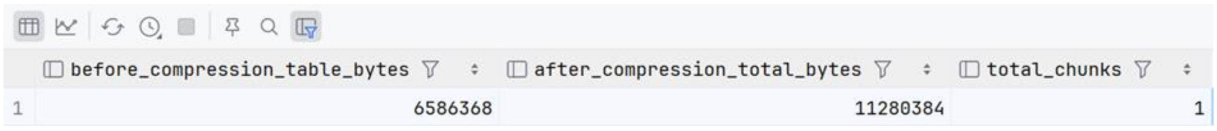
Stellio, as a generic tool, must handle down sampling carefully to avoid unintended effects. Introducing configurable down sampling mechanisms at the attribute level could provide greater flexibility and efficiency. To ensure this remains fully interoperable, these mechanisms should be discussed within the NGSI-LD specification, paving the way for broader adoption and consistent implementation.

5.4.3 Increasing data storage by database compression

Database compression is another way to reduce storage size without modifying the data itself. TimescaleDB, the database used in our setup, offers native compression for time-series data. The trade-off is you have to decompressed data before accessing it. This decompression will impact the request performance and may cause some errors if the compression is not configured perfectly.

5.4.3.1 Compression data storage reduction

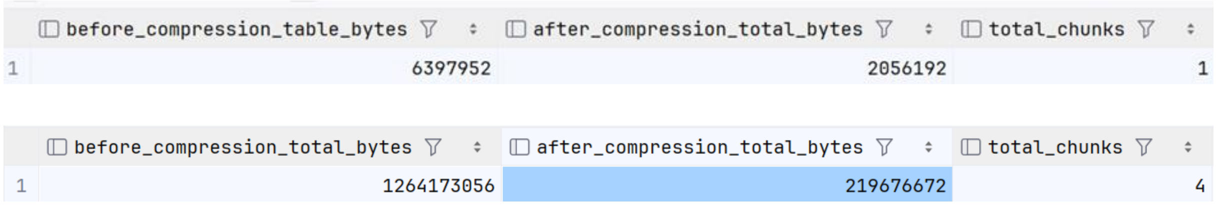
We experimented with the compression mechanism included in TimescaleDB, the tool that we use for time-series data. One of the challenges of compression is to find good configuration. For example, the first experiments gave a bigger data size after compression.



	before_compression_table_bytes	after_compression_total_bytes	total_chunks
1	6586368	11280384	1

Figure 56: Compression result segmented by id

After further investigation we found the segment by parameter that described how the elements are grouped in the compression. A good practice is to find a parameter that is a natural way of separating the table data. We tried segmenting the table by attribute and found way better results. With a reduced size of 68% for 12000 data points and 82% for a million datapoints.



	before_compression_table_bytes	after_compression_total_bytes	total_chunks
1	6397952	2056192	1

	before_compression_total_bytes	after_compression_total_bytes	total_chunks
1	1264173056	219676672	4

Figure 57: Compression result segmented by attribute

5.4.3.2 Impact on request performances

Another important configuration is the interval after which data is compressed. Since recent data is the most likely to be accessed and modified, setting a compression interval of one month ensures that requests targeting recent data remain unaffected.

For requests targeting data outside of this interval, compression can have a performance impact. The observed results are as follows:

Consumption Requests:

We observed an average performance degradation of 13–20 ms, corresponding to a 3% slowdown for long-running queries and up to 15% for very fast requests.

Table 13: Compression performance impact on consumption request.

Requests	Iterations	Duration	All tests	Avg. Resp.
Request1 (Long) Not compressed	500	9m 16s	0	1036 ms
Request1 (Long) Not compressed	500	9m 22s	0	1051 ms
Request2 (Small) Not compressed	100	26s 342ms	0	201 ms
Request2 (Small) Not compressed	100	28s 452ms	0	214 ms

Modification Requests:

The performance impact is more pronounced for modification operations, with delays of up to 1.2 seconds. This is due to the need to decompress and recompress the entire chunk of data affected by the modification. In our tests, the experimental setup created an unusually large chunk (over one million columns), which do not reflect typical real-world scenarios.

Fortunately, chunk size is configurable, and by splitting the chunk we were able to reduce the impact to 0.2 seconds. Further reductions in chunk size could yield even better performance. However, it's important to note that smaller chunk sizes decrease compression efficiency, resulting in less effective storage savings. Therefore, a careful trade-off between query performance and storage optimization must be considered.

Table 14: Compression performance impact on modification request.

Requests	Iterations	Duration	All tests	Avg. Resp.
Not compressed	500	1m	0	23 ms
Compressed	500	11m 24s	0	1276 ms
Compressed with reduced chunk	100	2m 50s	0	238 ms

6 Conclusions

One of the main goals of SEDIMARK is to provide easy to use tools for developers to understand their datasets, process them and improve their quality before sharing them to the marketplace, aiming to increase their value. As described in this series of documents, SEDIMARK made significant steps towards the twin aims of improving data quality while managing trade-offs with energy efficiency and environmental impact.

In Section 3, the current work on the SEDIMARK data pipeline was outlined, showing how a set of data quality metrics has been implemented, alongside a number of data cleaning tools and modules for orchestrating the pipeline and visualising its results. Work on implementing the data pipeline has concluded successfully, with modules for data profiling, data cleaning (anomaly detection, missing value imputation, deduplication), and data orchestration, as well as data augmentation (data synthesis) and feature engineering (dimension reduction and feature selection).

These modules were carefully designed and implemented based on the requirements elicitation process of the SEDIMARK WP2, aiming to address the most important functionalities that data providers need for assessing and improving the quality of their data, before these are shared through the marketplace. The data quality pipeline as developed by SEDIMARK balances efficiency and user friendliness and caters for users of different technical expertise. Advanced users can exploit the maximum capabilities of the pipeline and its modules, configuring themselves all the parameters of the models and designing new models using the modules through the command line interface. However, novice users can benefit from SEDIMARK's user interface through Mage.ai, where with minimum user intervention they can take advantage of the full set of the SEDIMARK pipeline tools, being able to just set the modules they want to execute and using default parameters. The SEDIMARK pipeline modules are open sourced and uploaded on GitHub. This was agreed by the partners aiming to allow the research community to improve and extend the modules and maximise impact.

In Section 4, the work for reducing the environmental impact of the data processing and AI tools of SEDIMARK was outlined, with approaches discussed for carbon friendly model training, inference, as well as reducing the impact of data sharing, communication and data storage. These approaches included dimension reduction, data distillation and coreset selection for reducing the model training cost. Quantisation, model pruning, model distillation and low rank factorization have also been investigated for further optimising machine learning models. Furthermore, it was analysed that there are viable methods for optimising communication cost, as well as that of data sharing and storage.

In the final section, an extended number of experiments showed the trade-offs between accuracy and environmental cost that are implicit in naively running the tools in the SEDIMARK toolbox. These experiments emphasised that often the results can be counter intuitive, for instance with full communication federated learning being more carbon efficient than low communication, or worse performance coming from highly parameterised anomaly detection algorithms. Additionally, it was shown that with Fleviden's compression and quantisation algorithms, the communication cost of distributed learning can be decreased significantly, without a major impact on performance. Energy consumption can also be decreased in other modules of the data processing pipeline using respective methods and parameters. For example, in the deduplication module, by using different indexing methods the energy consumption can be reduced significantly without an effect on precision. Similarly, in the anomaly detection module, the type of algorithm selected for detection can have a severe effect on energy consumption, with some algorithms being very resource heavy without having any benefit in terms of performance. Users of the module can then select themselves which algorithm they want to use based on their target objectives. Regarding synthetic data generation, it was shown that there is a trade-off between energy consumption in terms of computation and how close to the real data the synthetic ones are.

Techniques for minimising energy consumption in model training were also shown to have good results. For example, with low rank factorisation, data distillation and coreset selection methods, the computation cost of model training can be significantly reduced, without changing too much the model performance. In time series forecasting it was shown that model size and FLOPS can be reduced almost by half with a minimum reduction in performance. For techniques for offering generation, a reduction in computational and environmental cost can be achieved while preserving the semantic richness and structural integrity of marketplace offerings. It was also shown how AutoML can provide assistance to users to select models with high performance. As such, SEDIMARK should emphasise the need for user guidance, and automation, in their use of the pipeline.

To this end one conclusion is that SEDIMARK has managed to provide a set of tools that work together to automate the data cleaning process, while emphasising energy efficiency in the choices that it makes. SEDIMARK’s toolbox is adaptive and user friendly, catering for the diverse types of users that span from novice (using the user-friendly UIs) to experts (using command line configurations). Both carbon and human hours are rapidly expended via naïve approaches to data cleaning, with results that might still be suboptimal. Thus, going forward, a key aim of the final step of the integration work in WP5 is to further automate the pipeline, so as to reduce both the human and energy costs associated with it.

7 References

[1] https://www.marklogic.com/blog/the-staggering-impact-of-dirty-data/

[2] SEDIMARK Deliverable D2.1 Use Cases Definition and Initial Requirement Analysis, June 2023

[3] SEDIMARK Deliverable D2.2, SEDIMARK Architecture and Interfaces – First Version, September 2023

[4] SEDIMARK Deliverable D2.3, SEDIMARK Architecture and Interfaces – Final Version, September 2024

[5] SEDIMARK Deliverable D3.3, Enabling tools for data interoperability, distributed data storage and training distributed AI models. First version, December 2023.

[6] SEDIMARK Deliverable D4.3, Edge data processing and service certification. First version, December 2023

[7] Ilyas, I. F., & Chu, X. (2019). Data cleaning. Morgan & Claypool.

[8] *Chu, X., Ilyas, I. F., Krishnan, S., & Wang, J. (2016, June). Data cleaning: Overview and emerging challenges. In Proceedings of the 2016 international conference on management of data (pp. 2201-2206).*

[9] Batini, C., Cappiello, C., Francalanci, C., & Maurino, A. (2009). Methodologies for data quality assessment and improvement. ACM computing surveys (CSUR), 41(3), 1-52.

[10] Zha, D., Bhat, Z. P., Lai, K. H., Yang, F., Jiang, Z., Zhong, S., & Hu, X. (2023). Data-centric artificial intelligence: A survey. arXiv preprint arXiv:2303.10158.

[11] Ng, A. Data-centric ai resource hub. Snorkel AI. Available online: https://snorkel.ai/(accessed on 8 February 2023) (2021).

[12] https://stellio.readthedocs.io/en/latest/

[13] Panahy, P. H. S., Sidi, F., Affendey, L. S., Jabar, M. A., Ibrahim, H., & Mustapha, A. (2013). A framework to construct data quality dimensions relationships. Indian Journal of Science and Technology, 6(5), 4422-4431.

[14] Ortigosa-Hernández, J., Inza, I., & Lozano, J. A. (2017). Measuring the class-imbalance extent of multi-class problems. Pattern Recognition Letters, 98, 32-38.

[15] Kuemper, D., Iggena, T., Toenjes, R., & Pulvermueller, E. (2018, June). Valid. IoT: A framework for sensor data quality analysis and interpolation. In Proceedings of the 9th ACM Multimedia Systems Conference (pp. 294-303).

[16] Liu, X. (2015). Chapter 3 - Linear mixed-effects models. Methods and applications of longitudinal data analysis. Elsevier.

[17] Rostaghi, M., & Azami, H. (2016). Dispersion entropy: A measure for time-series analysis. IEEE Signal Processing Letters, 23(5), 610-614.

[18] Kalousis, A., Gama, J., & Hilario, M. (2004). On data and algorithms: Understanding inductive performance. Machine learning, 54, 275-312.

[19] Lorena, A. C., Costa, I. G., Spolaôr, N., & De Souto, M. C. (2012). Analysis of complexity indices for classification problems: Cancer gene expression data. Neurocomputing, 75(1), 33-42.

[20] Zhu, R., Wang, Z., Ma, Z., Wang, G., & Xue, J. H. (2018). LRID: A new metric of multi-class imbalance degree based on likelihood-ratio test. Pattern Recognition Letters, 116, 36-42.

[21] Lorena, A. C., Garcia, L. P., Lehmann, J., Souto, M. C., & Ho, T. K. (2019). How complex is your classification problem? a survey on measuring classification complexity. ACM Computing Surveys (CSUR), 52(5), 1-34.

[22] Orriols-Puig, A., Macià, N., Bernadó-Mansilla, E., & Ho, T. K. (2009). Documentation for the data complexity library in C++ (Tech. Rep. 2009001). La Salle, Universitat Ramon Llull.

[23] Agencia Estatal de Meteorología www.aemet.es

[24] Kader, G. D., & Perry, M. (2007). Variability for categorical variables. Journal of statistics education, 15(2).

[25] ydata-profiling (https://docs.profiling.ydata.ai/4.6/)

[26] IBM data quality API https://www.ibm.com/products/dqaiapi

[27] Samariya, D., & Thakkar, A. (2023). A comprehensive survey of anomaly detection algorithms. Annals of Data Science, 10(3), 829-850.

[28] Alrawashdeh, M. J. (2021). An adjusted Grubbs' and generalized extreme studentized deviation. Demonstratio Mathematica, 54(1), 548-557.

[29] Chen, Y., Miao, D., & Zhang, H. (2010). Neighborhood outlier detection. Expert Systems with Applications, 37(12), 8745-8749.

[30] Amer, M., Goldstein, M., & Abdennadher, S. (2013, August). Enhancing one-class support vector machines for unsupervised anomaly detection. In Proceedings of the ACM SIGKDD workshop on outlier detection and description (pp. 8-15).

[31] Louni, H. (2008). Outlier detection in ARMA models. Journal of time series analysis, 29(6), 1057-1065.

[32] Zhu, G., Zhao, H., Liu, H., & Sun, H. (2019, October). A novel LSTM-GAN algorithm for time series anomaly detection. In 2019 prognostics and system health management conference (PHM-Qingdao) (pp. 1-6). IEEE.

[33] An, J., & Cho, S. (2015). Variational autoencoder based anomaly detection using reconstruction probability. Special lecture on IE, 2(1), 1-18.

[34] Perini, L., Bürkner, P. C., & Klami, A. (2023, July). Estimating the contamination factor's distribution in unsupervised anomaly detection. In International Conference on Machine Learning (pp. 27668-27679). PMLR.

[35] Ruff, L., Vandermeulen, R., Goernitz, N., Deecke, L., Siddiqui, S. A., Binder, A., ... & Kloft, M. (2018, July). Deep one-class classification. In International conference on machine learning (pp. 4393-4402). PMLR.

[36] Ren, K., Yang, H., Zhao, Y., Chen, W., Xue, M., Miao, H., ... & Liu, J. (2018). A robust AUC maximization framework with simultaneous outlier detection and feature selection for positive-unlabeled classification. IEEE transactions on neural networks and learning systems, 30(10), 3072-3083.

[37] Blázquez-García, A., Conde, A., Mori, U., & Lozano, J. A. (2021). A review on outlier/anomaly detection in time series data. ACM Computing Surveys (CSUR), 54(3), 1-33.

[38] Zha, D., Bhat, Z. P., Lai, K. H., Yang, F., Jiang, Z., Zhong, S., & Hu, X. (2023). Data-centric artificial intelligence: A survey. arXiv preprint arXiv:2303.10158.

[39] Kulik, D., Schmidl, S. & Perini, L. (2023). KulikDM/pythresh: v0.3.5 (v0.3.5). Zenodo. https://doi.org/10.5281/zenodo.10072727

[40] Zhao, Y., Nasrullah, Z., & Li, Z. (2019). Pyod: A python toolbox for scalable outlier detection. arXiv preprint arXiv:1901.01588.

[41] Montiel, J., Halford, M., Mastelini, S. M., Bolmier, G., Sourty, R., Vaysse, R., ... & Bifet, A. (2021). River: machine learning for streaming data in python. The Journal of Machine Learning Research, 22(1), 4945-4952.

[42] Mehmood, H. S., Ahmad, R. Z., & Yousuf, M. J. (2019, July). A comprehensive review of adaptive noise cancellation techniques in the internet of things. In Proceedings of the 3rd international conference on future networks and distributed systems (pp. 1-8).

[43] Taneja, T., Jatain, A., & Bajaj, S. B. (2017, May). Predictive analytics on IoT. In 2017 International Conference on Computing, Communication and Automation (ICCCA) (pp. 1312-1317). IEEE.

[44] Krishnan, S., Wang, J., Wu, E., Franklin, M. J., & Goldberg, K. (2016). Activeclean: Interactive data cleaning for statistical modelling. *Proceedings of the VLDB Endowment*, *9*(12), 948-959.

[45] De Bruin, J. (2019). Python Record Linkage Toolkit: A toolkit for record linkage and duplicate detection in Python. *Zenodo DOI*, *10*.

[46] Lin, W. C., & Tsai, C. F. (2020). Missing value imputation: a review and analysis of the literature (2006–2017). *Artificial Intelligence Review*, *53*, 1487-1509.

[47] Jarrett, D., Cebere, B. C., Liu, T., Curth, A., & van der Schaar, M. (2022, June). Hyperimpute: Generalized iterative imputation with automatic model selection. In *International Conference on Machine Learning* (pp. 9916-9937). PMLR.

[48] Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[49] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, *63*(11), 139-144.

[50] Xu, L., Skoularidou, M., Cuesta-Infante, A., & Veeramachaneni, K. (2019). Modeling tabular data using conditional gan. *Advances in neural information processing systems*, *32*.

[51] Rajabi, A., & Garibay, O. O. (2022). Tabfairgan: Fair tabular data generation with generative adversarial networks. *Machine Learning and Knowledge Extraction*, *4*(2), 488-501.

[52] Open-source data pipeline tool for transforming and integrating data. https://www.mage.ai/

[53] Budennyy, S. A., Lazarev, V. D., Zakharenko, N. N., Korovin, A. N., Plosskaya, O. A., Dimitrov, D. V. E., ... & Zhukov, L. E. E. (2022, December). Eco2ai: carbon emissions tracking of machine learning models as the first step towards sustainable ai. In *Doklady Mathematics* (Vol. 106, No. Suppl 1, pp. S118-S128). Moscow: Pleiades Publishing.

[54] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine, 29(6), 141–142.

[55] Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., & Smith, V. (2020). Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, *2*, 429-450.

[56] Pang, G., Shen, C., & van den Hengel, A. (2019, July). Deep anomaly detection with deviation networks. In Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining (pp. 353-362).

[57] Feldman, D. "Core-sets: Updated survey." *Sampling techniques for supervised or unsupervised tasks* (2020): 23-44.

[58] Jubran, I., Maalouf, A. and Feldman, D.. "Introduction to coresets: Accurate coresets." *arXiv preprint arXiv:1910.08707* (2019).

[59] Sachdeva, N., and McAuley, J. "Data distillation: A survey." *arXiv preprint arXiv:2301.04272* (2023).

[60] Wang, T., et al. "Dataset distillation." arXiv preprint arXiv:1811.10959 (2018).

[61] Duriakova, E., Tragos, E., Lawlor, A., Smyth, B. and Hurley, N., "Boosting the Training Time of Weakly Coordinated Distributed Machine Learning," in IEEE International Conference on Big Data, 2021

[62] Gholami, A., "A survey of quantization methods for efficient neural network inference," in Low-Power Computer Vision, Chapman and Hall/CRC, 2022, pp. 291-326.

[63] Cheng, Y., Wang, D., Zhou, P., and Zhang, T. "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," IEEE Signal Processing Magazine, vol. 35, no. 1, pp. 126-136, 2018

[64] Liang, T., Glossner, J., Wang, L., Shi, S. and Zhang, X.,, "Pruning and quantization for deep neural network acceleration: A survey," Neurocomputing, no. 461, pp. 370-403, 2021.

[65] Vadera S., and Ameen, S., "Methods for pruning deep neural networks," IEEE Access,, vol. 10, pp. 63280-63300, 2022.

[66] Choudhary, T., Mishra, V., Goswami, A., and Sarangapani, J., "A comprehensive survey on model compression and acceleration," Artificial Intelligence Review, vol. 53, pp. 5113-5155, 2020.

[67] Gou, J., Yu, B., Maybank, S. J., and Tao, D., "Knowledge distillation: A survey," International Journal of Computer Vision, no. 129, pp. 1789-1819, 2021.

[68] Xu, C., and McAuley, J. "A survery on model compression and acceleration for pretrained language models.," in AAAI, 2023.

[69] Swaminathan, S., Garg, D., Kannan, R., and Andres, F., "Sparse low rank factorization for deep neural network compression.," no. 398, pp. 185-196, 2020.

[70] Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R., "Exploiting linear structure within convolutional networks for efficient evaluation.," in Advances in neural information processing systems, 2014.

[71] Yu, X., Liu, T., Wang, X., and Tao, D., "On compressing deep models by low rank and sparse decomposition.," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017.

[72] https://www.cs.utexas.edu/users/ml/riddle/data.html

[73] SEDIMARK GitHub, https://github.com/Sedimark

[74] SEDIMARK Deliverable D3.1 Energy efficient AI-based toolset for improving data quality. First version, December 2023

[75] Mushtaq, Rizwan. "Augmented dickey fuller test." (2011).

[76] Kwiatkowski, D., Phillips, P.C.B., Schmidt, P., & Shin, Y. (1992). Testing the null hypothesis of stationarity against the alternative of a unit root. Journal of Econometrics, 54: 159-178.

[77] https://www.statsmodels.org

[78] Shannon, Claude E. "A mathematical theory of communication." The Bell system technical journal 27.3 (1948): 379-423.

[79] Wetschoreck F., Krabel T., Krishnamurthy S. 8080Labs/ppscore: zenodo release (2020), 10.5281/ZENODO.4091345 URL: https://zenodo.org/record/4091345

[80] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685, 2021.

[81] Peeters, R., & Bizer, C. (2023, August). Using chatgpt for entity matching. In *European Conference on Advances in Databases and Information Systems* (pp. 221-230). Cham: Springer Nature Switzerland.

[82] Narayan, A., Chami, I., Orr, L., Arora, S., & Ré, C. (2022). Can foundation models wrangle your data?. *arXiv preprint arXiv:2205.09911*.

[83] Gregg, F., & Eder, D. (2022). dedupe (Version 2.0.11) [Computer software]. https://github.com/dedupeio/dedupe

[84] Nikitin, A., Iannucci, L., & Kaski, S. (2024). Tsgm: A flexible framework for generative modeling of synthetic time series. *Advances in Neural Information Processing Systems*, *37*, 129042-129061.

[85] Wagner, D., Michels, T., Schulz, F. C., Nair, A., Rudolph, M., & Kloft, M. (2023). Timesead: Benchmarking deep multivariate time-series anomaly detection. *Transactions on Machine Learning Research*.

[86] Shen, L., Li, Z., & Kwok, J. (2020). Timeseries anomaly detection using temporal hierarchical one-class network. *Advances in neural information processing systems*, *33*, 13016-13026.

[87] Carmona, C. U., Aubet, F. X., Flunkert, V., & Gasthaus, J. (2021). Neural contextual anomaly detection for time series. *arXiv preprint arXiv:2107.07702*.

[88] Su, Y., Zhao, Y., Niu, C., Liu, R., Sun, W., & Pei, D. (2019, July). Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 2828-2837).

[89] Li, D., & Wang, J. (2019). Fedmd: Heterogenous federated learning via model distillation. *arXiv preprint arXiv:1910.03581*.

[90] Thapa, C., Arachchige, P. C. M., Camtepe, S., & Sun, L. (2022, June). Splitfed: When federated learning meets split learning. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 36, No. 8, pp. 8485-8493).

[91] Gu, A., Goel, K., & Ré, C. (2021). Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*.

[92] Yoon, J., Jarrett, D., & Van der Schaar, M. (2019). Time-series generative adversarial networks. *Advances in neural information processing systems*, *32*.

[93] Zhou, L., Poli, M., Xu, W., Massaroli, S., & Ermon, S. (2023, July). Deep latent state space models for time-series generation. In *International Conference on Machine Learning* (pp. 42625-42643). PMLR.

[94] Ryu, S., Yu, Y., & Seo, H. (2024). Can untrained neural networks detect anomalies?. *IEEE Transactions on Industrial Informatics*, *20*(4), 6477-6488.

[95] Viswanathan, Vijay, et al. "DataFinder: Scientific dataset recommendation from natural language descriptions." *arXiv preprint arXiv:2305.16636* (2023).

[96] S. Robertson and S. Walker. Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. In ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pages 345–354, 1994.

[97] https://huggingface.co/docs/peft/en/index

[98] https://github.com/Sedimark/Recommender/tree/main/sedirec/training

[99] https://github.com/viswavi/datafinder/tree/main

[100] Salomon, D., Motta, G., Salomon, D., & Motta, G. (2010). Dictionary methods. Handbook of Data Compression, 329-441.

[101] Huffman, David (1952). "A Method for the Construction of Minimum-Redundancy Codes". Proceedings of the IRE. 40 (9). Institute of Electrical and Electronics Engineers (IEEE): 1098–1101. doi:10.1109/jrproc.1952.273898. ISSN 0096-8390

[102] Robinson, A. H., & Cherry, C. (2005). Results of a prototype television bandwidth compression scheme. Proceedings of the IEEE, 55(3), 356-364.

[103] Ziv, Jacob; Lempel, Abraham (May 1977). "A Universal Algorithm for Sequential Data Compression". *IEEE Transactions on Information Theory*. 23 (3): 337–343. CiteSeerX 10.1.1.118.8921. doi:10.1109/TIT.1977.1055714. S2CID 9267632

[104] Deutsch, L. Peter (May 1996). *DEFLATE Compressed Data Format Specification version 1.3*. IETF. p. 1. sec. Abstract. doi:10.17487/RFC1951. RFC 1951

[105] J. Duda, *Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding*, arXiv:1311.2540, 2013

[106] Burrows, Michael; Wheeler, David J. (May 10, 1994), *A block sorting lossless data compression algorithm*, Technical Report 124, Digital Equipment Corporation, archived from the original on January 5, 2003

[107] Hutter, F., Hoos H. H., and Leyton-Brown K. (2011), "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization*.

[108] https://www.kaggle.com/datasets/venisatewu/statlog-landsat-satellite-data-set

[109] https://www.kaggle.com/code/nimapourmoradi/water-potability/notebook

[110] MacQueen, J. (1967, January). Some methods for classification and analysis of multivariate observations. In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics(Vol. 5, pp. 281-298). University of California press.

[111] Hacohen, G., & Weinshall, D. (2019, May). On the power of curriculum learning in training deep networks. In International conference on machine learning (pp. 2535-2544). PMLR.

[112] Gonzalez, T. F. (1985). Clustering to minimize the maximum intercluster distance. Theoretical computer science, 38, 293-306.

[113] Lin, H., & Bilmes, J. (2012, August). Learning mixtures of submodular shells with application to document summarization. In Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence (pp. 479-490).

[114] Koh, P. W., & Liang, P. (2017, July). Understanding black-box predictions via influence functions. In International conference on machine learning (pp. 1885-1894). PMLR.

[115] Bingham, E., & Mannila, H. (2001, August). Random projection in dimensionality reduction: applications to image and text data. In Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 245-250).

[116] Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996, August). A density-based algorithm for discovering clusters in large spatial databases with noise. In kdd (Vol. 96, No. 34, pp. 226-231).

[117] Mirzasoleiman, B., Bilmes, J., & Leskovec, J. (2020, November). Coresets for data-efficient training of machine learning models. In International Conference on Machine Learning (pp. 6950-6960). PMLR.

[118] Kulesza, A., & Taskar, B. (2012). Determinantal point processes for machine learning. Foundations and Trends® in Machine Learning, 5(2–3), 123-286.

[119] https://github.com/Sedimark/noise_cancellation

[120] Hunter, J. Stuart. "The exponentially weighted moving average." Journal of quality technology 18.4 (1986): 203-210.

[121] Farooq, Younus, and Serkan Savaş. "Noise removal from the image using convolutional neural networks-based denoising auto encoder." Journal of Emerging Computer Technologies 3.1 (2024): 21-28.

[122] Perry, Marcus B. "The weighted moving average technique." Wiley Encyclopedia of Operations Research and Management Science (2010).

[123] Schafer, Ronald W. "What is a savitzky-golay filter?[lecture notes]." IEEE Signal processing magazine 28.4 (2011): 111-117.

[124] Zhongshen, Li. "Design and analysis of improved Butterworth low pass filter." 2007 8th International Conference on Electronic Measurement and Instruments. IEEE, 2007.

[125] M. A. Jaro, "Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida," Journal of the American Statistical Association, vol. 84, no. 406, pp. 414–420, 1989.

[126] W. E. Winkler, "String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage." 1990.

[127] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," (in Russian), Doklady Akademii Nauk, vol. 163, no. 4, pp. 845–848, 196

[128] Bormann, Carsten, and Paul Hoffman. Concise binary object representation (cbor). No. rfc7049. 2013.

[129] S. Furuhashi. MessagePack. https://msgpack.org//, 2018.

[130] Alistarh, Dan, et al. "QSGD: Communication-efficient SGD via gradient quantization and encoding." Advances in neural information processing systems 30 (2017).

[131] J. Blackard and D. Dean, 'Covertype'. UCI Machine Learning Repository, Aug. 1998. doi: 10.24432/C50K5N.

[132] H. Harries and J. Gama, 'Electricity'. 2004. [Online]. Available: https://web.archive.org/web/20180305095759/https://www.inescporto.pt/~jgama/ales/ales_5.html

[133] S. Madden, 'Intel Lab Data'. June 2004. Accessed: June 05, 2024. [Online]. Available: https://db.csail.mit.edu/labdata/labdata.html

[134] N. Verma, A. Bifet, B. Pfahringer, and M. Bahri, 'ASML: a scalable and efficient AutoML solution for data streams', in *Proceedings of the Third International Conference on Automated Machine Learning*, Paris, France: PMLR, Oct. 2024, p. 11/1-26. Accessed: Mar. 19, 2024. [Online]. Available: https://proceedings.mlr.press/v256/verma24a.html

[135] M. Bahri and N. Georgantas, 'AutoClass: AutoML for data stream classification', in *2023 IEEE International Conference on Big Data (BigData)*, Sorrento, Italy: IEEE, Dec. 2023, pp. 2658–2666. doi: 10.1109/BigData59044.2023.10386362.

[136] H. M. Gomes *et al.*, 'Adaptive random forests for evolving data stream classification', *Mach. Learn.*, vol. 106, no. 9, pp. 1469–1495, Oct. 2017, doi: 10.1007/s10994-017-5642-8.

[137] H. M. Gomes, J. Read, and A. Bifet, 'Streaming random patches for evolving data stream classification', in *2019 IEEE International Conference on Data Mining*, Beijing, China: IEEE, Nov. 2019, pp. 240–249. doi: 10.1109/ICDM.2019.00034.

[138] A. Bifet, G. Holmes, and B. Pfahringer, ‘Leveraging bagging for evolving data streams’, in Machine Learning and Knowledge Discovery in Databases, J. L. Balcázar, F. Bonchi, A. Gionis, and M. Sebag, Eds, Barcelona, Spain: Springer, 2010, pp. 135–150. doi: 10.1007/978-3-642-15880-3_15.