SEcure Decentralised Intelligent Data MARKetplace

D3.1 Energy efficient AI-based toolset for improving data quality. First version

Document Identification	
Contractual delivery date:	31/12/2023
Actual delivery date:	19/12/2023
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

This document is issued within the frame and for the purpose of the SEDIMARK project. This project has received funding from the European Union's Horizon Europe Framework Programme under Grant Agreement No. 101070074. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

Document Information

List of Contributors	
Name	**Partner**
Elias Tragos Diarmuid O'Reilly Morgan Erika Duriakova Honghui Du Qinqin Wang Aonghus Lawlor Neil Hurley	NUID-UCD
César Caramazana Maxime Costalonga Jairo Rojas-Delgado	ATOS
Luc Gasser Léa Robert Romain Magnani Franck Le Gall Philippe Cousin Gilles Orazi	EGM
Maroua Bahri Nikolaos Georgantas	INRIA
Ioannis Tsogias Nikos Babis	MYT
Gabriel Danciu Stefan Jarcau	SIE
Tarek Elsaleh Adrian Hilton	SURREY

List of Contributors	
Pablo Sotres Laura Martín Juan Ramón Santana Jorge Lanza Luis Sánchez	UC
Panagiotis Vlacheas Panagiotis Demestichas Grigoris Koutantos Loizos Koutsantonis Konstantinos Almpanakis Dimitrios Triantafyllou	WINGS

Document History			
Version	**Date**	**Change editors**	**Change**
0.1	13/02/2023	NUID-UCD	First draft of ToC
0.11	04/10/2023	SURREY	Contribution to Section 3.3
0.12	08/10/2023	WINGS	Contribution to Section 3.7.1
0.13	15/10/2023	WINGS	Contribution to Sections 3.6, 3.8, 4.4
0.2	07/11/2023	ATOS	Contribution to Section 4.3.1
0.31	08/11/2023	INRIA	Contribution to Section 3.2
0.32	08/11/2023	SIE	Contribution to Sections 3.3, 3.4
0.33	08/11/2023	EGM	Contribution to Section 3.7.2
0.34	09/11/2023	INRIA	Contribution to Sections 3.9, 3.10
0.4	16/11/2023	NUID-UCD	Contribution to Sections 3.5, 3.6, 3.7.1, 3.7.3, 3.7.4. 3.8, 4.2, 4.3.2, 4.3.3, 4.3.4
0.41	17/11/2023	SURREY	Contribution to Section 4.5
0.42	17/11/2023	SIE	Updates to Sections 3.2, 3.3, 3.4
0.43	17/11/2023	UC	Contribution to Sections 3.5, 3.7.4
0.5	21/11/2023	EGM	Contribution to Sections 4.6, 4.7

Document History			
0.51	21/11/2023	NUID-UCD	Updates to Sections 3.5, 3.6, 3.7.1, 3.7.3, 3.7.4, 4.2
0.52	23/11/2023	NUID-UCD	Updates to Section 3.5, contribution to Section 5
0.53	23/22/2023	MYT	Review of Sections 3.5, 3.6, 3.7
0.6	27/11/2023	SURREY	Updates to Section 4.5
0.7	28/11/2023	SIE	Updates to section 3.3, 3.4
0.71	28/11/2023	WINGS	Updates to Section 3.8
0.72	28/11/2023	UC	Updates to section 3.2
0.73	28/11/2023	INRIA	Updates to Section 3.10
0.74	28/11/2023	MYT	Review of Sections 3.8. 3.9
0.8	29/11/2023	SURREY	Updates to section 4.5
0.81	29/11/2023	NUID-UCD	Contribution to Sections 1, 6.
0.82	29/11/2023	NUID-UCD	Overall editing and formatting of the deliverable.
0.9	30/11/2023	NUID-UCD	Final editing and preparing version for internal review
0.91	06/12/2023	NUID-UCD	Addressing the comments of the first internal review
0.92	11/12/2023	NUID-UCD	Addressing the comments of the second internal review and preparing version for quality review.
0.93	15/12/2023	NUID-UCD	Pre-final version addressing the quality review comments.
0.95	18/12/2923	ATOS	Quality Format Review
1.0	19/12/2023	ATOS	FINAL VERSION TO BE SUBMITTED

Quality Control		
Role	**Who (Partner short name)**	**Approval date**
Reviewer 2	Juan Ramón Santana, Pablo Sotres (UC)	07.12.2023
Reviewer 1	Arturo Medela, César Caramazana (ATOS)	05.12.2023

Quality Control		
Role	**Who (Partner short name)**	**Approval date**
Quality manager	María Guadalupe Rodríguez (ATOS)	18.12.2023
Project Coordinator	Arturo Medela (ATOS)	19.12.2023

Table of Contents

List of Figures

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
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
IoT	Internet of Things
IR	Imbalance Ratio

Abbreviation / acronym	Description
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
SGD	Stochastic Gradient Descent
SMA	Simple Moving Average

Abbreviation / acronym	Description
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
VIF	Variance Inflation Factor
WCL	Weekly Coordinated Learning
WMA	Weighted Moving Averages

Executive Summary

Data quality is considered to be 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/analysing data to extract knowledge and information for decision-making processes. Thus, one of the main work items of SEDIMARK is to develop a usable data processing pipeline that assesses and improves the quality of data generated and shared by the SEDIMARK data providers.

This deliverable presents the first version of the methods and techniques developed within SEDIMARK for processing data and improving their quality. The focus is at first to identify the key techniques that can be used for quality improvement of datasets and then to improve the techniques and adapt them to the requirements of the SEDIMARK platform so that they can all work together smoothly.

SEDIMARK considers two main types of data generated and shared within the marketplace: (i) static/offline datasets and (ii) online/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 are developed for static and streaming datasets. In this respect, techniques about outlier detection, noise removal, deduplication and imputation of missing values are considered 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 are also developed in order to enrich the datasets. Synthetic dataset creation is considered 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 that has been developed in order to orchestrate the whole functionality of the data processing pipeline using a Data Processing Orchestration component, which the users are using via a dashboard. The deliverable also presents the quality metrics that SEDIMARK has defined for assessing the quality of datasets, both per data point and as a whole.

Another important part of the deliverable 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 discussed, focusing on quantisation, pruning, low rank factorisation and knowledge distillation.

Finally, considering that minimising energy consumption can have an effect on performance or communication, the deliverable presents a first 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 first deliverable from WP3, aiming to provide a first draft of the SEDIMARK data quality pipeline. This document details how the pipeline aims 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 is actually the first version of the deliverable, showing the initial ideas and initial implementation of the respective tools and techniques. An updated version of the deliverable with the final version of the data quality pipeline will be delivered in M34 (July 2025). This means that the document should be considered a "live" document that will be continuously updated and improved as the technical development of the data quality tools evolves.

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 will be provided to them for improving the quality of their data before sharing on the data marketplace. This will help 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 USD, 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 and the initial ideas about them, so that the rest of the technical tasks can draw ideas about the integration 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

Figure 1 shows the interaction of the activities within WP3 and the relation with the rest of the work packages. This deliverable is the output of work done in the first 15 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]). 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 next version of this deliverable (SEDIMARK_D3.2, to be delivered July 2025), the refinement of the architecture in SEDIMARK_D2.3 (WP2), as well as for the integration of the overall system (integrating the components of T3.1 and T3.5) and the overall system testing and validation (in WP5).

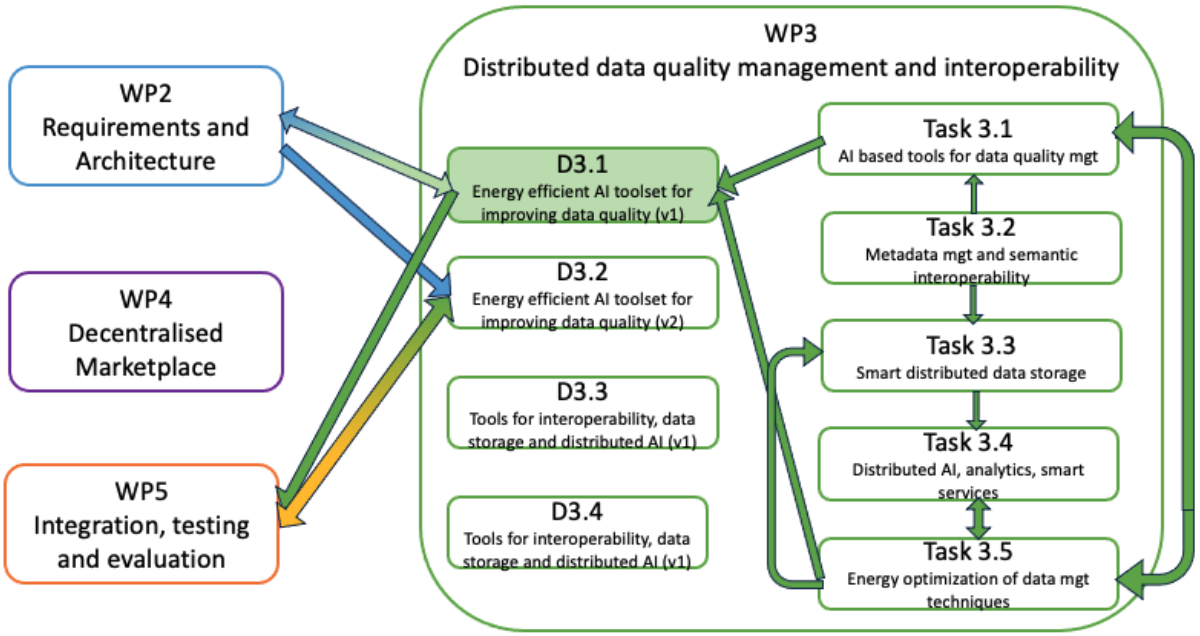

Figure 1: Relationship between SEDIMARK_D3.1 and other deliverables, tasks, and work packages.

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 the future steps.

1.4 Glossary adopted in this document

The readers are referred to SEDIMARK_D2.2 which provides a complete list of the terminology used within SEDIMARK.

2 Position within the project

This deliverable presents the first 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 work 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.
- **Data Processing Dashboard**: 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 Data Processing Dashboard 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, 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 in the next versions 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.

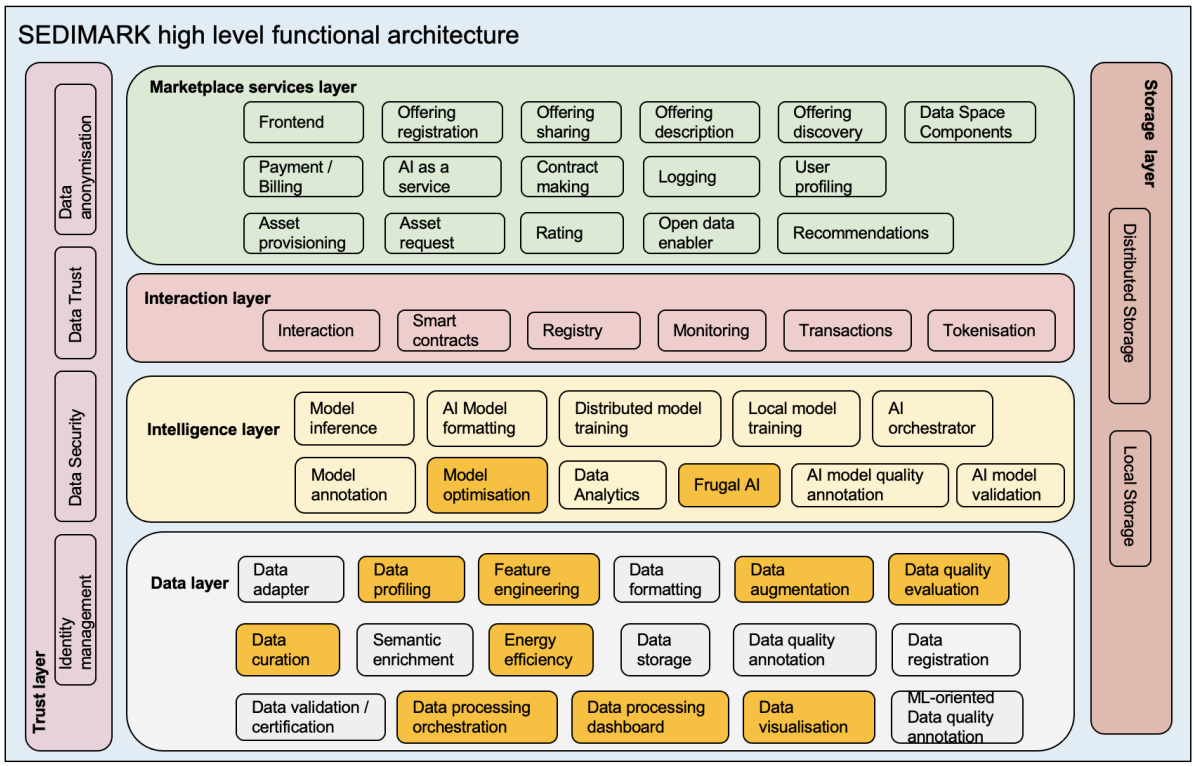

Figure 2: Mapping of SEDIMARK_D3.1 components to the SEDIMARK architecture.

As discussed in the introduction section, this is the first version of the deliverable describing the methods and tools developed within SEDIMARK related to data processing and energy efficiency. As discussed in deliverable SEDIAMRK_D2.2 in Figure 48 [3], these tools are part of the Data Processing Pipeline, which is part of the SEDIMARK toolbox. The developed tools will become public in the future, considering that most of them will be open sourced. In detail, currently there are Python implementations of:

- **Data Processing Orchestration, Dashboard** and **Visualisation**, implemented using mage.ai and Figma.
- **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.

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 future development of SEDIMARK will concern reducing the amount of human involvement required by the data cleaning process. As such, SEDIMARK will explore 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 of this information to the Data Processing Orchestration component, which is the main component that 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 [4]) are given in the below subsections.

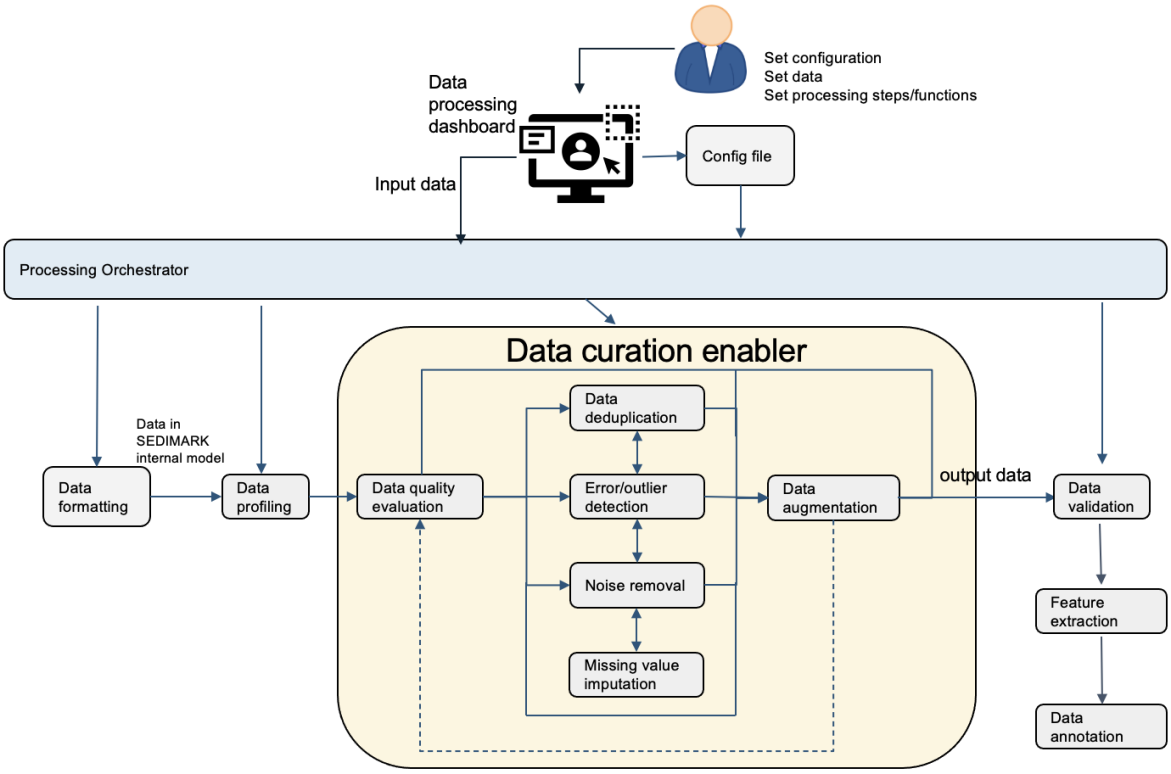

Figure 3: Overview of the data processing pipeline of SEDIMARK

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 provided 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 continuously and require immediate action, streaming is the preferred option, 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

Figure 4 represents a multi-stage data processing architecture involving various components that coordinate to handle, process, and store data. The flow begins with a Data Producer/Provider that interacts with a UI/API Server. This server forwards the data to a Pipeline Scheduler, which orchestrates the data flow through multiple processing stages. Brokers facilitate the queuing of data between the scheduler, an Object Store, and a set of processes that execute data transformation or analysis tasks using the SEDIMARK Toolkit.

The Object Store manages the storage of processed or intermediate data, while additional data sources can provide supplementary input as needed. This proposed architecture supports parallel processing and is likely designed for real-time data handling, as indicated by the presence of brokers and a pipeline scheduler. The output of the entire pipeline is represented by a Directed Acyclic Graph (DAG) depicting the workflow and the Results from the data processing.

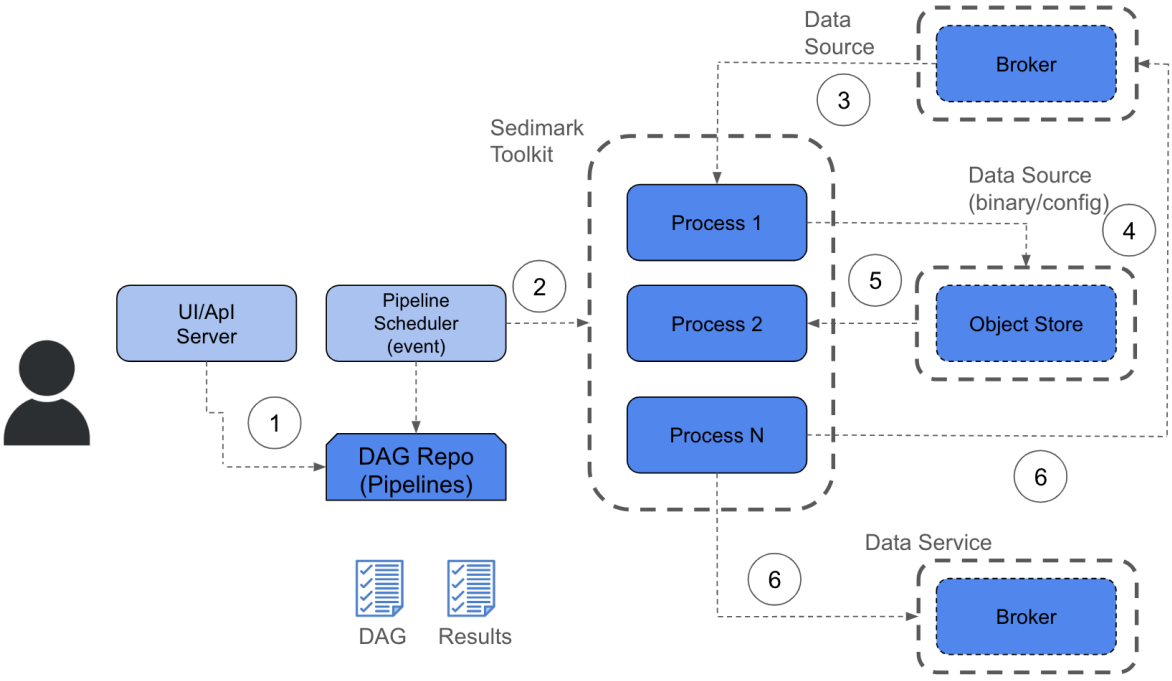

Figure 4: Architecture diagram of a Multi-Stage Data Processing and Analysis pipeline with Brokers, Object store and SEDIMARK toolkit integration.

The detailed orchestration process involves the following steps after the orchestrator is started:

1. The data provider defines the pipeline for data processing using the UI or programmatically through the corresponding API. The pipeline configuration is then submitted to the orchestrator engine.
2. Based on the pipeline triggering method, whether instant, periodic or based on an event, such as a minimal amount of new data within the original data asset, the pipeline is started.
3. The first step of the pipeline involves retrieving the original data asset from the data source. This will be provided by a data broker in the first instance.
4. The next process involves extracting the data from the original format and applying a data processing technique on the raw data. The result could involve the creation of intermediate artefacts that would be needed for the next process.
5. The next process takes the output of the first process and applies the next (valid) data processing technique according to the pipeline DAG.
6. The final data asset output is then pushed to the NGSI-LD Broker and the Offering Sharing component that will exchange the asset with Data Consumers.

The above-described steps are presented mainly for streaming datasets, but a similar process can be adapted for static datasets as well.

Mage.ai [66] is a platform that helps at streamlining data preparation, automating key tasks such as cleaning, transforming, and feature engineering with machine learning algorithms. It offers a user-friendly interface that caters to both technical and non-technical users, simplifying data handling without the need for extensive coding. The platform also encourages team collaboration, enabling multiple users to work together seamlessly on data projects.

Additionally, Mage.ai's flexible integration capabilities allow it to connect with various data sources and machine learning tools. This flexibility is embodied in its modular approach to data processing, which divides tasks into separate, customizable modules, enhancing the efficiency and adaptability of the data preparation process.

The overall orchestration follows these steps:

- Define and submit the data processing pipeline.
- Trigger the pipeline based on specified conditions (instant, periodic, event-based).
- Retrieve the original data asset and process it through various stages (anomaly detection, etc.).
- Export the final processed data for sharing with data consumers.

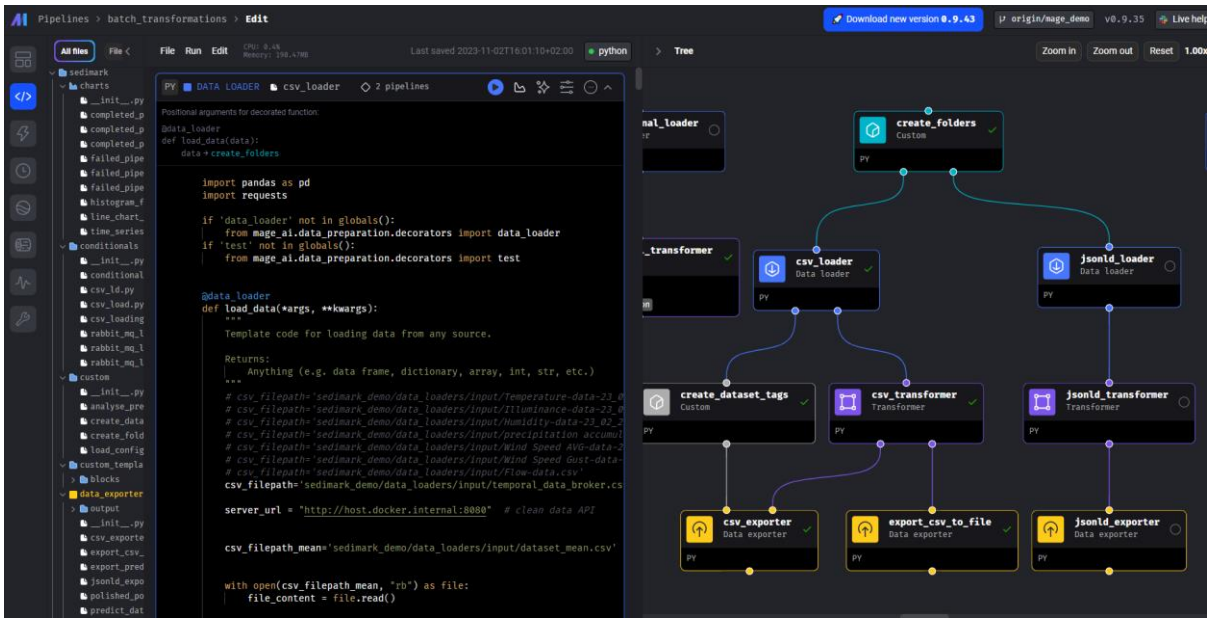

Figure 5: Mage.ai interface with sample demo flow for processing SEDIMARK data.

The pipeline presented in Figure 5 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. The figure currently shows two loaders, one using csv and the other one using NGSI-LD.

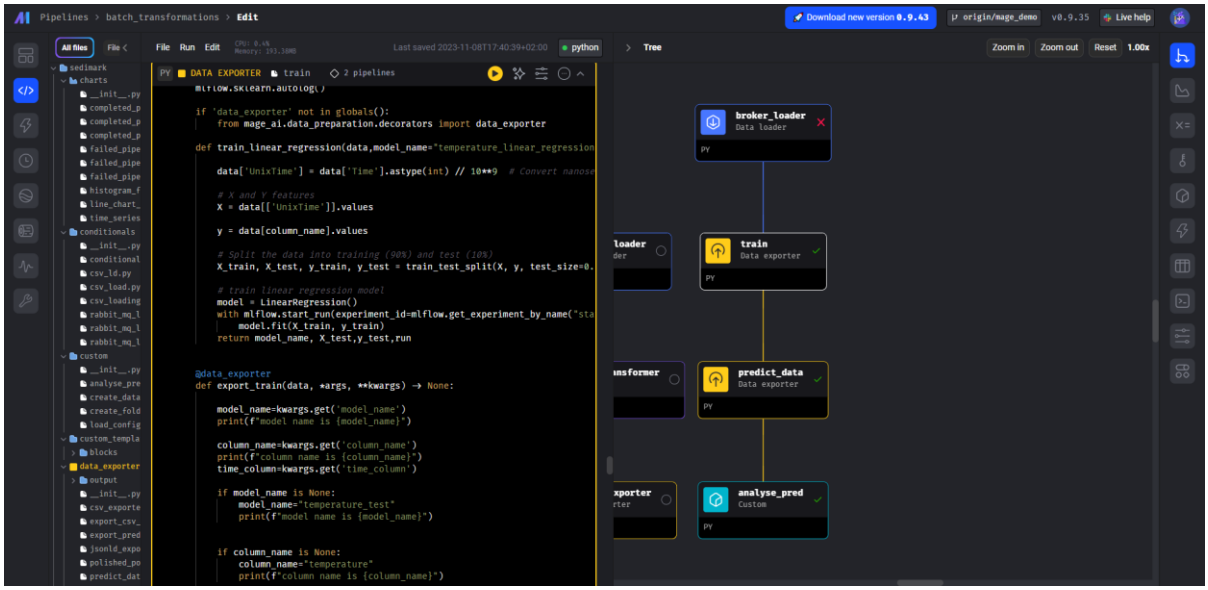

Figure 6: Mage.ai orchestration for a simple training flow using SEDIMARK data.

Figure 6 shows how a simple flow for analysing a dataset can be configured, leveraging Mage.ai's capabilities for streamlined data handling. 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, i.e. the Stellio context broker [12]. This data is then processed and formatted, preparing it for the subsequent stages of the pipeline, which can be either data prediction or any of the modules of the data processing pipeline, i.e. anomaly detection, deduplication or value imputation. Mage.ai uses a common way to launch all underlying components.

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

3.4 Data Processing Dashboard

Creating a Data Visualization dashboard that simplifies the complex data flows of the AI training and prediction pipelines involves several key steps. The goal is to make the interface intuitive and informative for users, regardless of their technical background.

The key, when designing this GUI is to highlight crucial information such as data sources, model performance metrics, or transaction flows in an intuitive manner. This involves integrating charts and graphs for visual representation of data like transactions, service results like forecasts, alongside real-time updates to reflect the most current predictions.

The dashboard's design prioritizes user-friendliness, with a clean layout that groups related information together and employs colour coding and icons for quick recognition. Interactive elements like filters and drill-down capabilities are essential for enhancing user engagement, allowing users to delve deeper into specific data points or adjust views according to their needs.

Responsiveness is another crucial aspect of the dashboard design, ensuring accessibility across various devices and screen sizes. Additionally, incorporating a feedback mechanism allows for gathering user insights, which is vital for continuous improvement of the interface.

Behind the scenes, the dashboard is powered by web technologies and visualization libraries, all integrated through APIs that fetch data dynamically from the pipeline. This technical setup ensures that the dashboard is not only informative but also interactive and up to date.

Overall, the dashboard stands as a user-centric interface that distils intricate data processes into a format that is accessible and actionable for a wide range of users, fostering better understanding and decision-making based on the AI pipeline's outputs. Figure 7 presents the view of the wrappers over Mage.ai functionalities using Figma, which is a web application for interface design [13].

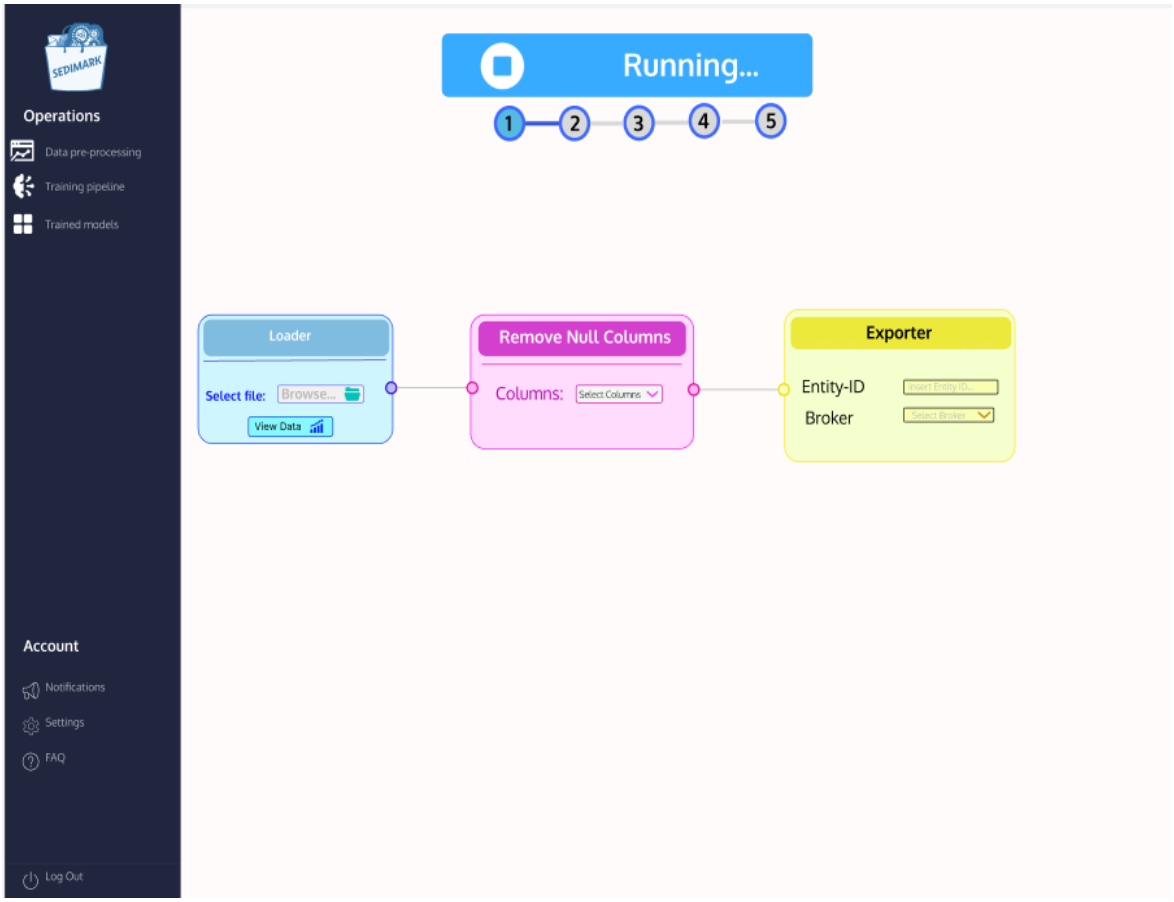

Figure 7: Figma mock for the Data Processing Dashboard.

This is the initial view of how the user will be able to customize the data processing pipelines by:

- The possibility to select the file/source of the data set. This would be a loader control custom designed so that the user can view a preview of the data before continuing with the process. The visualization is in Figure 8.
- The possibility to clean incomplete or redundant columns. This is depicted in Figure 7 with the magenta coloured block. Of course, many other actions can be implemented in

the data processing pipeline like outlier removal, redundant columns removal or any other data cleaning algorithms.

- The yellow coloured block signifies the exporter of the data pipeline processing results and this can be customized to either export the data to a Broker or another source or a further pipeline.

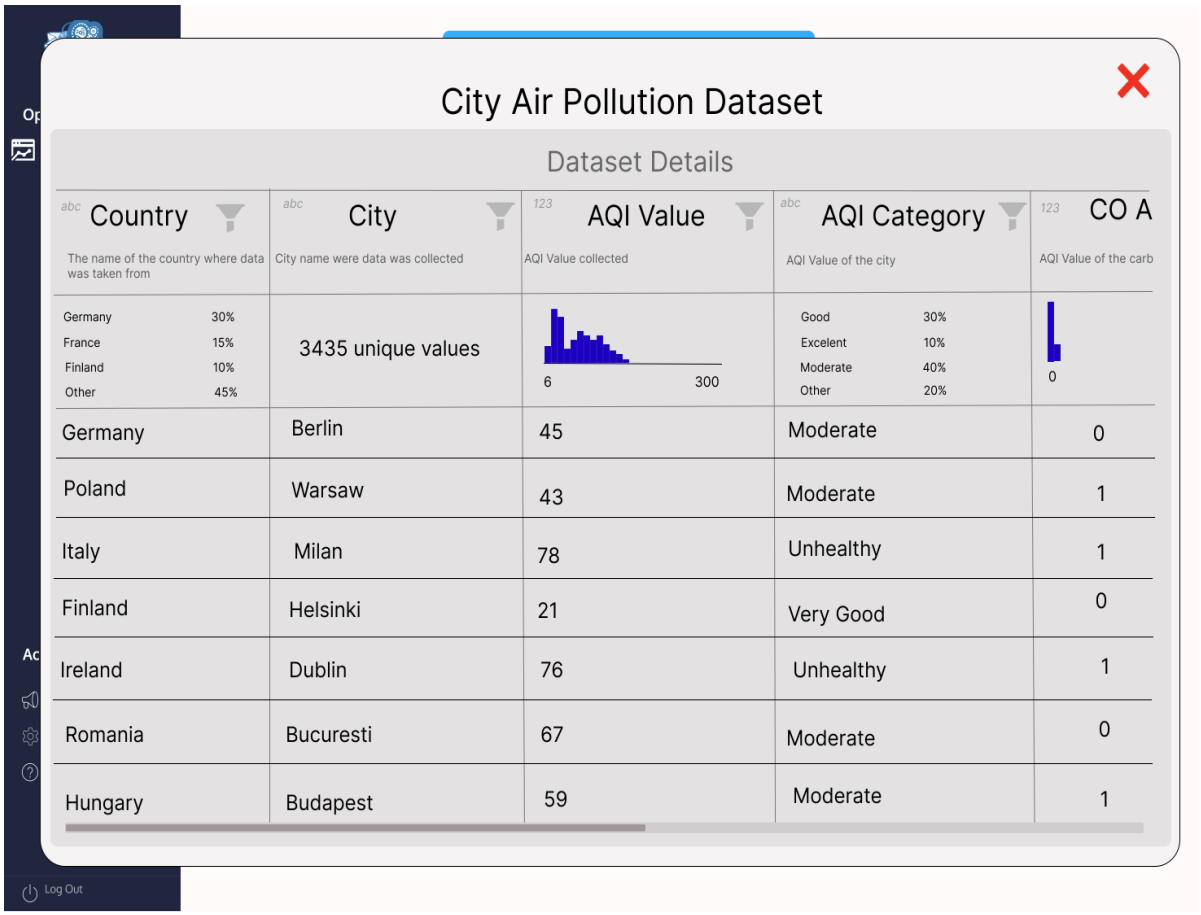

Figure 8: Figma mock for the dataset visualisation.

Figure 8 shows what the user will see when clicking on the "preview" button of the Loader block.

Other such visualizations can be created to help the user understand the changes caused by each of the intermediate step of the data pipeline. More details about the visualizations of the data processing or other service visualizations, will be described in the deliverable SEDIMARK_D4.5.

3.5 Data quality metrics

As previously stated, improved data quality is of the 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],[14] 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 the fact 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" [15].

- **Normalised cross entropy** [22],[23], 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)** [24], 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's 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 [18] 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 [25],[26] due to its simplicity. More advanced metrics will be considered for the next version of the tool.

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 [28] 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 [16], 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.

3.5.3 Future contributions

In future versions of the SEDIMARK Data Quality Evaluation modules, more metrics will be considered and developed, i.e.:

- **Stationarity**: Using the Augmented Dickey-Fuller Test [17], this metric tests the hypothesis that a unit root is present in the data. A lower metric score suggests that the series is more likely to be stationary.
- **Decomposition**: For univariate time series, the function decomposes data into its trend, seasonal, and residual components, offering deeper insights into its underlying patterns [19].
- **Entropy**: Represents the randomness or unpredictability in the series. A higher entropy value suggests more unpredictability [20].

- **Variance Inflation Factor (VIF)**: Relevant for multivariate time series, VIF [21] measures multicollinearity, indicating 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.

This assortment of metrics gives a thorough profile of time series data, aiding in understanding its behaviour, patterns, and characteristics. For unidimensional signals a frequency analysis can be used. This is useful particularly for categorical data and involves counting the number of occurrences of each category or class. It helps in understanding the distribution of categories within the data. Another technique well fitted to understanding the underlying structure is Principal Component Analysis (PCA). This is used to reduce the dimensionality of large datasets, increase interpretability, and minimize information loss. Metrics like algorithmic complexity can also provide insights into the structural complexity of the dataset. Beyond basic decomposition, more complex methods like STL (Seasonal and Trend decomposition using Loess) provide a more robust understanding of time series data. An alternative to correlation, PPS (Predictive Power Score) can capture non-linear relationships between variables and is useful in understanding the predictive capability of each feature.

Additionally, future versions of the Data Quality Evaluation modules will also include visualisations for these metrics, i.e.:

- Heatmap to visualize the correlation matrix between variables.
- Bar chart, ideal for categorical data, showing the frequency of each category.
- Scatter plot for visualizing potential outliers in a continuous dataset.
- Histogram or line plot to visualize algorithmic complexity or other complexity metrics.
- Violin plot that combines aspects of box plots and density plots, ideal for visualizing the distribution and its quantiles.

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 [29]: 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 [30]: 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_D2.2 [3] 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 [27] 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 9 shows a representative scheme of the module depicting the input needed and the outputs expected.

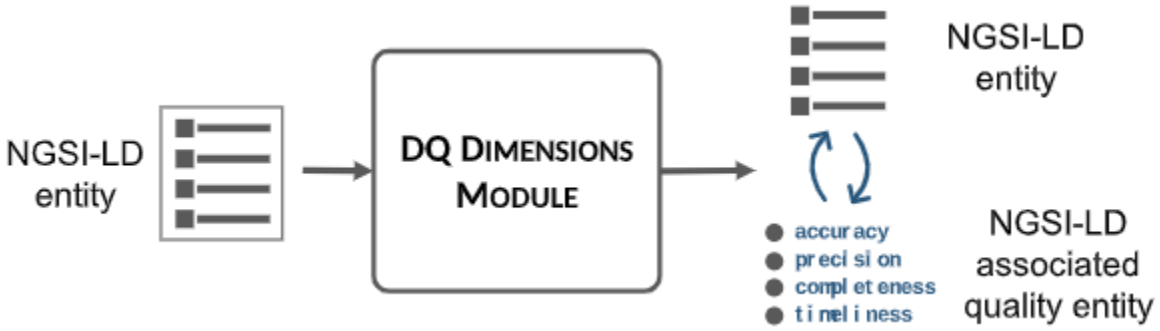

Figure 9: DQ Dimensions Module for streaming environments

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 the next Figure 10. 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.

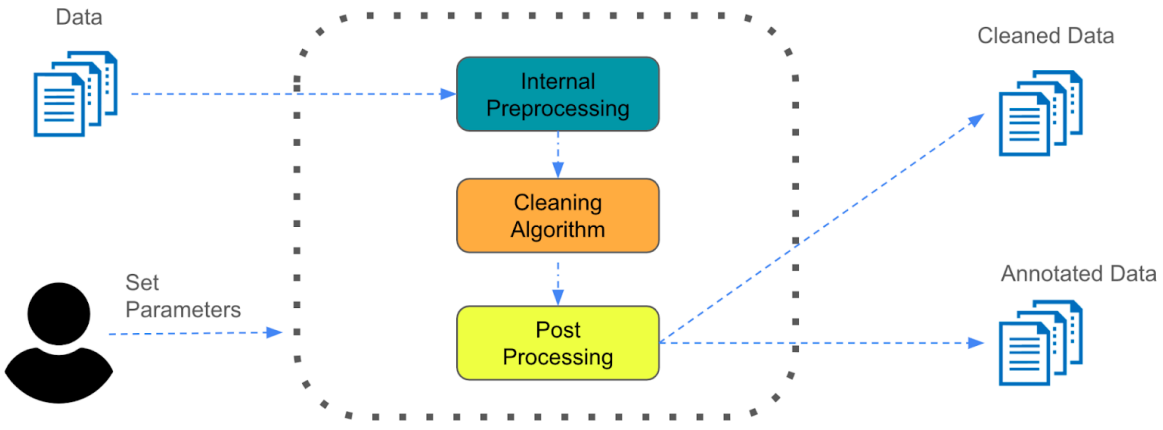

Figure 10: A generic SEDIMARK data cleaning module.

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 [44] 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) [40] or classifying anomalies per their autoencoder reconstruction error [37]. 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 [31]. 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 [42]. 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) [41], gamma Gaussian Mixture Models (gammaGMM) [39] and Generalised Extreme Studentized Deviation ESD (GESD) [32] exist to automatically determine this threshold automatically by processing the raw anomaly scores.

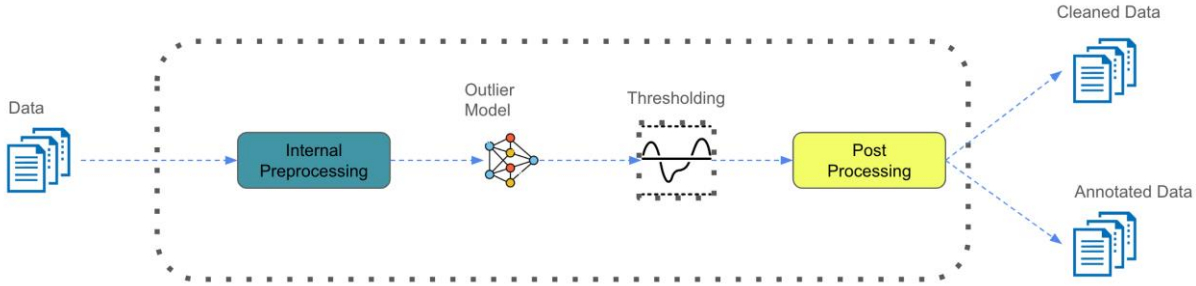

Figure 11: SEDIMARK outlier detection module.

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 [49] for the case of tabular data, and TODS [38] for time series data. Both PYOD and TODS include a large number of unsupervised OD approaches, such as traditional machine learning methods like KNN based methods [33], one class Support Vector Machines (SVMs) [34], or simple autoregression models [35] 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 [36]. In addition, SEDIMARK makes use of the PyThresh library [48] 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 11 where input data first undergo pre-processing to be PYOD/TODS 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.

At present, SEDIMARK includes the following algorithms from TODS:

- **MultiAutoRegOD**: A simple autoregressive model, classifying data points according to their error.
- **LSTMOD**: A recurrent neural network outlier detection model.

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.

To this end, SEDIMARK aims to build upon a number of recently emerging techniques for automating the model selection process in unsupervised outlier detection. On the one hand, a small number of papers [43],[45] have investigated the possibility of applying meta-learning to the model selection process. In these approaches, generally a large number of differently parameterised models are trained and evaluated on a large number of publicly available datasets that themselves have ground truth anomaly labels available. Then datasets are mined for statistical characteristics that can be utilised as descriptive features. A meta-learner, such as a decision tree or neural net is then trained to predict the performance of each model class, given these features as input.

On the other hand, another line of work investigates extracting features that can better serve as proxies for anomaly performance. Putina et. al. [47] employ three loose proxies for anomaly performance and use these to guide the AutoML process. Goswami et. al. [46] employ a more advanced approach, whereby they identify a broader class of surrogate anomaly performance metrics and combine them with a rank aggregation approach to specifically target anomaly detection for time series data. Going forward, SEDIMARK will seek to integrate these approaches into the OD component of the data pipeline, thus progressively automating the need for human involvement in OD model selection.

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 [50], 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.

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 removal

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 [51].

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 [52].

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 [51]. 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 [51].

There are two categorical approaches to increase the SNR [53]: real-time (or online / streaming / incremental) and offline (bulk) processing. Concerning offline processing, by far the most popular approach is Principal Component Analysis (PCA) or Independent Component Analysis (ICA), however, offline methods require that the SNR be constant over time (stationarity), suppose Gaussian systems, and demands high computational power. Real-time algorithms, on the other hand, filter the signals as they stream but still require prior knowledge of the noise to tune the filter parameters [53], and are poorly suited for non-stationary, non-Gaussian, complex signal and noise systems; adapt poorly to abrupt changes in dynamics or outliers, introduce a lag in the processed signal, and produce loss of fine details and high-frequency information [53].

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. Traditional noise removal methods include: (i) smoothing moving averages (Simple Moving Average (SMA), Exponential Moving Averages (EMA), Weighted Moving Averages (WMA)), (ii) filters (Kalman, Wiener, Total Variation Denoising), (iii) statistical analysis models (PCA, ICA), (iv) spectral analysis models (Wavelet Transform, Fourier Transform (FFT), Singular Spectrum Analysis (SSA)), (v) Autoregressive models (Autoregressive Integrated moving averaged (ARIMA) or Seasonal Autoregressive Integrated moving averaged (SARIMA)), and (vi) neural networks (NN) (Autoencoders such as Variational Autoencoders (VAEs) and Denoising Autoencoders, Recurrent NN (RNNs) such as Long Short-term Memory (LSTM) and Gates Recurrent Unit (GRU), Convolutional NN (CNNs), Generative Adversarial Networks (GANs), Support Vector Machines (SVMs)).

In practice, several of these (base) models are combined using ensemble methods (ex. Gradient Boosting, Adaptive Boosting, Random Forest) to reduce more effectively the noise in time series. These combine predictions from several base models (also called weak learners) to generate a stronger more accurate final model.

An example of a combination of base models was shown by [54] who proposed to combine K-Nearest Neighbours (KNN) algorithm, to classify the error/noise in data and compute the transition matrix, and the Kalman Filter (KF) algorithm to predict the final value using the transition matrix. This method was compared with state-of-the-art "base" algorithms and outperformed by maximum 30% their accuracy.

Such offline batch methods have the disadvantage of being computationally intensive and built on suppositions that parameters such as noise characteristics are constant [53]. To overcome these limitations, the best way to effectively eliminate background noise from streaming real-time data is to use adaptive noise cancellation technology [51],[54]. Adaptive noise cancellation filters are widely used in communication and signal processing systems. They offer reduced computation, robust implementation, and can automatically adapt to changes in the external environment and the various noise profiles, without knowing the statistical characteristics of the input signal to achieve the best filtering effect. They are particularly effective in instances where the noise profiles are non-stationary or unknown in advance.

A comparison of three popular adaptive filtering algorithms used for real-time noise reduction is given in [51]: (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.

In conclusion, the most efficient and precise results for noise removal of streaming data within a dynamic and ever-changing environment is to perform online adaptive processing, with the possibility of using combinations of base algorithms in ensemble learning methods. The best choice depends on the specific characteristics of the time series data, the current noise profile, the underlying signal patterns, the levels of quality and adaptability required.

Examples of Python libraries that perform adaptive noise cancellation or mitigation and that will be tested are:

- padasip [96]: Performs filtering, predictive and reconstruction for adaptive real-time signal processing.
- adaptfilt [97]: Adaptive filtering for LMS, NLMS, RLS and more.
- MetaAF: Control adaptive filters with neural networks [98]: Adaptive filtering for LMS, NLMS, RLS, GRU and more.
- Noisereduce [99]: Adaptive filtering using spectral gating.
- PyWavelets [100]: Wavelet filtering of 1-D and 2-D signal.

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 user 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 [55] 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 [56], which relies upon user specified rules for data deduplication. These include e.g., setting specific metrics for string similarity (e.g., Jaro-Winkler or Levenshtein distance) 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.

At this stage, it is unclear how to progress towards an automated data pipeline, as for instance many of the available machine learning data deduplication methods require either labelled data, or in the case of dedupe [70] (Active Learning), a human evaluator must assist the algorithm in classifying low certainty pairs. In line with the work on AutoML below, SEDIMARK intends to work on further automating this process. This might be done in conjunction with information available from the NGSI-LD data models used in the project - perhaps allowing it to be inferred as to which fields would serve as the best candidates for deduplication matching.

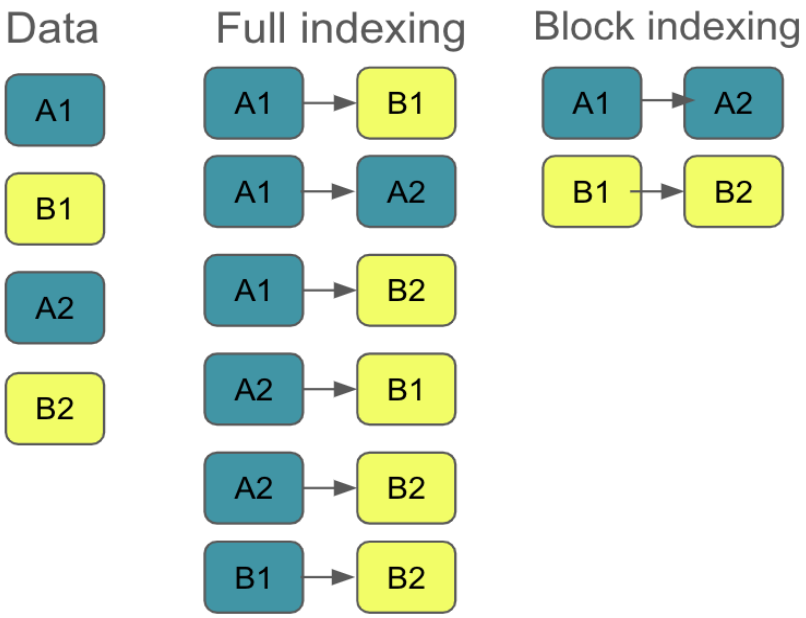

Figure 12: Comparison of full and block indexing methods.

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 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 12. 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.

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 [57]. 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 [57]. 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 13, where the module fills the gaps that it identifies in the dataset.

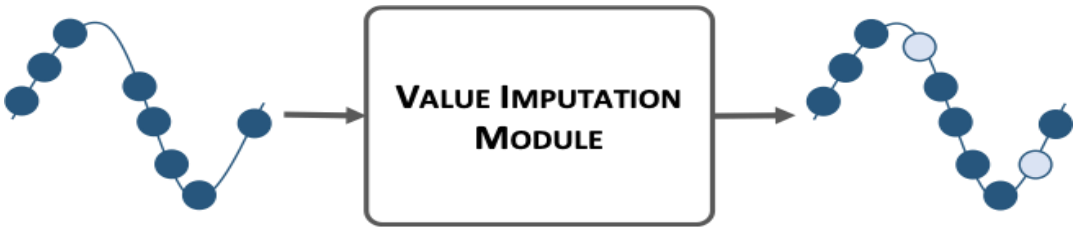

Figure 13: Module for imputing missing values.

3.7.4.1 Offline

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 [58]. 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.

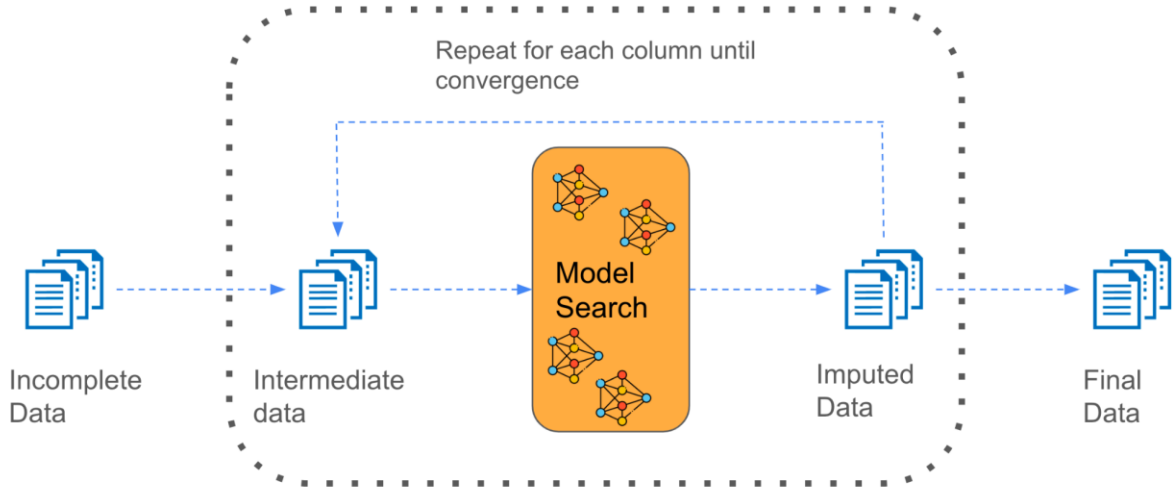

Figure 14: Iterative imputation (HyperImpute).

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 [58] (Figure 14), 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

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.

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

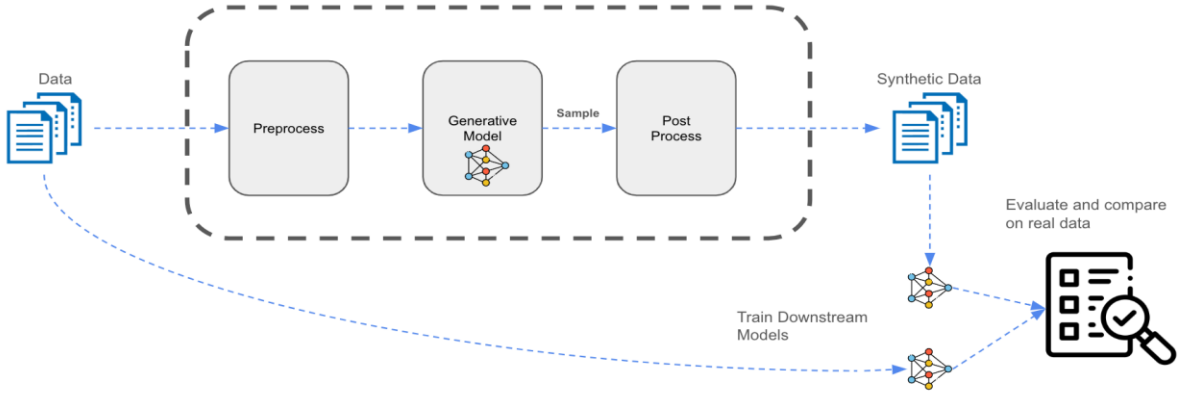

Figure 15: Synthetic data generation in SEDIMARK

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 [60] allowing for the drawing of samples from a learned gaussian latent space. In the case of GANs [61], 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 15.

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 [62] 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 actually 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. A future version of the SEDIMARK toolbox will build upon existing tools such as SynthCity [63] and TimeGan [59] in order to generate synthetic time series data.

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. In a future version of the tools, SEDIMARK will explore more fairness specific approaches such as Tabfairgan [64] and Fairgan+ [65] for either generating debiased, or balanced synthetic datasets from scratch, or using generated samples to augment and thus reweight original datasets.

3.8.4 Data Augmentation for Time series

In a future version of SEDIMARK, a time-series tool for data augmentation will provide an innovative approach to enhance the richness of time series datasets, catering to the need for more diverse data in training machine learning models. With the power of two distinct techniques – Jittering and Synthetic Minority Over-sampling Technique (SMOTE) [101] – it will address various augmentation needs. The 'Jittering' method will infuse minor random fluctuations or 'noise' into the original data, enhancing model robustness against natural variances without distorting the inherent patterns. On the other hand, SMOTE, will generate synthetic time series samples by interpolating between existing ones. This is particularly useful in addressing class imbalance problems or when the dataset is sparse. By offering both methods in one tool, users will be equipped with a versatile augmentation solution, enabling them to train more resilient and generalized models.

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 16 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.

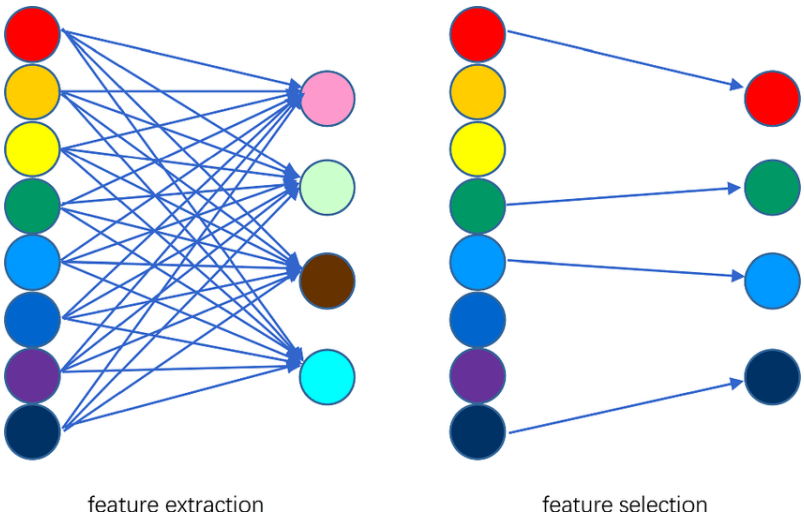

Figure 16: Feature extraction vs. feature selection

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, SEDIMARK uses 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 also 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 will be extended and could be 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.

For the second version of the SEDIMARK data processing pipeline, the aim is to 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 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 ensures 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 is also 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 is also a benefit of this choice. Through these 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

Recent years have seen a growing focus on the environmental impact caused by computing technology. This is especially evident in the fields of big data and machine learning, where computationally intensive processes can often consume large amounts of electricity, thus producing high amounts of carbon emissions, while the amount of data that is recorded and stored increases every year. As illustrated in [95], over the past five years, state of the art machine learning models have grown exponentially in their parameter counts. As greater computational resources become available, and machine learning models grow significantly in size, the climate impact of machine learning increases concomitantly, while the storage and transfer of the data needed to train them also creates a significant environmental effect. At the same time, much research has looked at ways to efficiently scale deep learning models, and to reduce their environmental impact, both during training and inference. This chapter briefly looks at some of the ways in which SEDIMARK plans to address its environmental impact. Section 4.2 discusses ways to decrease energy consumption during training ML models, via the use of smart sampling, data distillation and dimension reduction. Section 4.3 looks at further ways to optimise models for reduced energy consumption, via quantization, pruning, model distillation and low rank factorization. Section 4.4 further looks at how dimension reduction might be used to reduce the environmental cost incurred in sharing data. Section 4.5 discusses ways to reduce the cost of storing data. Finally, Section 4.6 discusses plans to implement a light version of the SEDIMARK toolbox that will be operable on edge devices. Section 4.7 discusses some first thoughts about addressing grey energy.

4.2 Reducing energy consumption during ML training

4.2.1 Optimising Data Efficiency in ML Training

As machine learning models become more complex and data-intensive, the computational resources required for their training have increased significantly. Advanced algorithms often involve processing vast datasets multiple times, leading to prolonged use of high-powered computing systems.

This escalation in energy use has several impacts: it can lead to increased operational costs, making ML less accessible to smaller organisations. Environmentally, it contributes to higher carbon emissions, especially if the energy is sourced from fossil fuels. Additionally, it can strain energy resources, challenging the sustainability of broader technological growth.

In SEDIMARK, improving data efficiency is a key strategy in managing the considerable energy needs associated with machine learning training. The goal of optimising data efficiency is to reduce the quantity of data needed to train models without compromising the performance, thus saving on computational resources and energy, which are both critical factors in reducing the environmental impact of ML training. This subsection explores three pivotal methods for improving data efficiency: Coreset Selection, Data Distillation and Dimension Reduction.

4.2.1.1 Coreset Selection

Coreset selection [72],[73] 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 [74],[75] 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.

4.2.1.3 Dimension Reduction

Dimension reduction techniques [76], [77], i.e., feature selection and extraction, play a pivotal role in addressing the imperative need to reduce energy consumption in various applications.

By identifying and retaining only the most relevant features from the dataset, these techniques effectively trim down the data's dimensionality. This process not only streamlines computational complexity but also contributes significantly to minimizing energy requirements. Through methods like PCA or correlation, redundant or less informative features are identified and removed, ensuring that subsequent computations are more energy efficient. Moreover, the reduction in data dimensionality facilitates the use of lighter models, which inherently demand less energy during both training and inference phases. So, employing dimension reduction techniques emerges as a strategic approach to reducing energy consumption, providing a sustainable solution while maintaining or even enhancing predictive performance.

Both coreset selection, data distillation and dimension reduction offer potent avenues for SEDIMARK to enhance data efficiency, significantly reducing the energy consumption associated with ML training. These techniques not only streamline the learning process but also fortify the platform's commitment to privacy and scalability. By implementing these methods, SEDIMARK stands to lower its environmental impact and operational costs, fostering a sustainable and resilient infrastructure for data technologies.

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 [78], 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 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 [78], 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.

With this in mind, SEDIMARK will provide a hybrid ML approach, whose aim is to boost the model initialisation process. SCL will be used to initialise the model which is then refined by WCL. The result is a considerably decreased training time of the algorithm compared to standard WCL approaches. Due to dense communication overhead, SCL has the ability to distribute the model data across the network very early in the training process. This hybrid approach starts with a SCL approach for a small number of epochs with a view to navigating the latent space more efficiently at this critical stage of the algorithm. Then the model parameters learned during this time are stored and input into the WCL algorithm. The algorithm is then let to fine-tune the global model with minimal communication overhead.

The main challenge of this approach is to effectively balance the increased cost of communication while at the same time improving the global training time of the model. As demonstrated in [78], a good trade-off between communication cost and training time can be well estimated during the parameter tuning process. Moreover, it was shown that the communication overhead of such a hybrid algorithm is relatively small while at the same time drastically improving the overall learning time of the global algorithm.

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 [79], SEDIMARK focuses on reducing the memory footprint and computational cost of the machine learning models and hence focuses on four main approaches for model compression:

- **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.

Different model compression techniques can be applied to different ML models. For instance, Deep neural networks can be compressed using all four 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 [80]. 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 Quantisation in federated learning

4.3.1.1 Summary

Quantized federated learning is a method that combines federated learning with quantisation techniques to reduce the communication overhead by transmitting compressed model updates during distributed training across multiple devices or nodes. This work implements several quantisation approaches for federated learning and evaluates them in terms of accuracy, convergence speed and communication cost. The results provide quantitative details on the trade-offs between the accuracy, convergence speed and communication between the agents (used as a proxy of energy consumption) in a benchmark set of forecasting problems.

4.3.1.2 Introduction

Training machine learning models using private data from several agents is a problem of pressing importance. Federated Learning (FL) is an increasingly broad topic that has been presented as the fundamental framework in which a set of agents can jointly train machine learning models without sharing their sensitive data.

In the general FL protocol, as illustrated in Figure 17, an algorithm or model is trained across multiple decentralized edge devices that hold local and sensitive data samples and exchange only the local models with a global server that is in charge of aggregating the parameters and sharing them back to the clients. There are, however, many variations of this protocol. For example, the agents may collaborate with each other to aggregate the models without relying on a global server (Gossip Learning), or they may opt to use some privacy preserving technique before the aggregation, like Differential Privacy. In all cases, the need to share and send through the network a local model (represented by the empty blocks with arrows in all directions) remains.

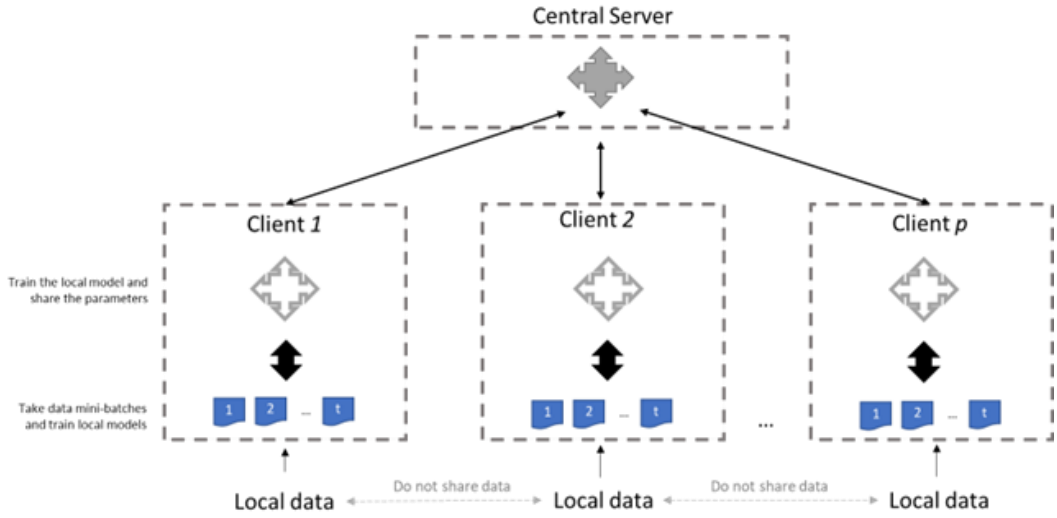

Figure 17: General federated learning protocol, taken from the Fleviden platform documentation.

The communication and exchange of local models through the network represents one of the major bottlenecks and energy consumption issues in federated learning. This is especially true in the case of the general FL protocol because the central server is responsible for single-handedly aggregating the models received from all agents. This work explores the use of quantisation approaches to address this problem while minimizing the impact on the accuracy of the global model and the convergence speed of the entire training process.

The rest of this section is organized as follows. Section 4.3.1.3 reviews related work, and several quantisation approaches available for federated learning training. In Section 4.3.1.4 the Quantized Stochastic Gradient Descent (QSGD) algorithm is described as a general parametrizable lossy-compression scheme for federated learning. Section 4.3.1.5 provides additional details about the implementation in the federated learning solution, specifying logical, component and deployment views of the overall software architecture.

4.3.1.3 Related work

In this section some relevant quantisation approaches for Federated Learning are discussed. As the complexity of the machine learning models has drastically increased over the past few years, so has the need to find solutions that aim to reduce energy consumption and storage space [80],[81],[82] .One of the most studied strategies, widely spread in the field of telecommunications, is quantization, understood as the process of mapping a continuous

space of values into a discrete set. As such, its application in the field of machine learning has also become matter of interest.

Alistarh et al. [83] introduce a general parametrizable lossy-compression scheme for multi-processor training as a two-step algorithm that 1) first performs a quantisation of the gradients given a certain number of levels, and 2) then employs an efficient encoding strategy using the Elias-omega code [84] .In their experiments they demonstrate that the reduced communication cost between processors comes at no significant accuracy loss after model convergence.

On the other hand, Reisizadeh et al. [85] propose an algorithm that aims to address the communication bottleneck by spatially and temporally subsampling the number of nodes that send their local updates at each iteration. On top of that, a low precision quantizer is used to compress the difference between the received parameters and the updated ones.

Hönig et al. [86] show that, just as clients may have different degrees of contribution to the aggregation of weights (federated weighted average), different levels of quantisation can be applied to each of them independently, following a client-adaptive quantisation strategy. Moreover, they have noticed that, at the beginning of the training, more coarse quantisation can be applied without loss in accuracy. For this reason, they suggest adopting a dynamic quantisation level approach, by which the aggressiveness of the quantisation is modulated over time, namely, with lower number of levels at the beginning that are increased over the rounds.

4.3.1.4 The Quantized Stochastic Gradient Descent algorithm

The Quantized Stochastic Gradient Descent (QSGD) algorithm is a family of compression schemes with convergence guarantees and good practical performance for convex and smooth functions [83]. The QSGD algorithm takes a vector of real numbers, such as the weights of a neural network, and transforms each component into a quantized, fixed-sized domain of values. Then, it generates a compressed encoding of such vector that is ready to be sent through the network. This resulting encoding is not lossless, which means that it is not possible to recover the original vector from its quantized version.

QSGD provides some guarantees of convergence for convex and non-convex objectives while significantly reducing the total number of bytes when compared to the original vectors of real numbers that are typically encoded as arrays of 32 bits floats.

The algorithm is defined by two complementary functional blocks, namely, the encoding and decoding procedures. Algorithm 1 clarifies how these two functions are used in the context of a general federated learning protocol.

Algorithm 1. QSGD algorithm encoding and decoding in a general federated learning protocol.

```
1. Let w be a random global model parameter vector
2. for i := 1 to rounds do
3.      Encode w and send it from the central server to each federated learning agent
4.      for j := 1 to num_agents do
5.              Receive and decode model w from the central server in the j-th agent
6.              Train a model initialized by w to obtain the local model w_j
7.              Encode model w_j using the encode function to obtain q_j
```

```
8.            Send q_j to the central server
9.      end for
10.     for j := 1 to num_agents do
11.           Receive quantized local model q_j from the j-th agent in the central server
12.           Decode local model q_j using the decode function to obtain r_j
13.     end for
14.     Aggregate local models r_j and update global model w in the central server
15. end for
16. Output global model w
```

Algorithm 1 starts by randomly initializing the vector with the parameters of the global model. In step (2), a given number of federated learning rounds is performed. For each round, the server first encodes the global model as defined by QSGD and broadcasts it to all the agents (3). Each agent decodes the model (5), trains it locally (6), encodes the result (7) and sends it back to the central server (8). The server then receives, decodes, and aggregates the quantized local models, in steps (10-14), before the next federated learning round starts again.

Quantisation function

To understand the compression mechanism behind QSGD, first the hyper-parameter *s* needs to be defined, which is an integer larger than zero that represents the number of quantisation intervals used to encode and decode the information. In other words, *s* controls the granularity in which a continuous interval is divided in discrete chunks. Figure 18 exemplifies how a value in an interval is quantized for *s*=5.

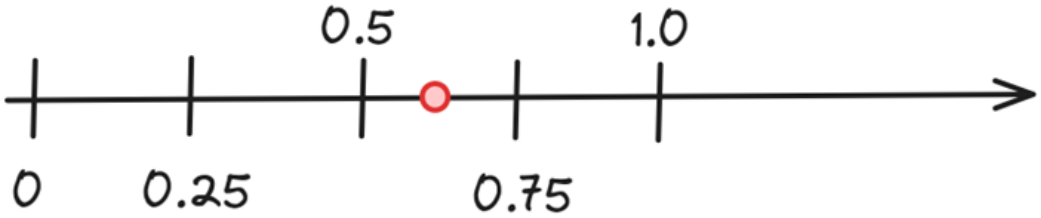

Figure 18: An illustration of QSGD encoding schema. A real number between 0.5 and 0.75 represented in red is quantized in the discrete level 0.5 or 0.75 depending on the stochastic quantisation schema defined by QSGD.

Given a vector $v \in R^n$ different than the zero vector, the encoding function $Q(v)$ is defined as follows:

$$Q(v_i) = \|v\|_2 \, sgn(v_i) \, e(v_i, s)$$

where $e(v_i, s)$ is an independent random variable following a distribution with support on the set of quantisation levels defined by *s*. Let $0 \leq l \leq s$ be an integer such that:

$$e(v_i, s) = \begin{cases} l & with\ probability\ 1 - \dfrac{s\,|v_i|}{\|v\|_2} - l \\ l+1 & otherwise \end{cases}$$

Any vector v can be expressed by using a triplet consisting of the Euclidean norm of the vector, a vector of signs of v and the vector of integer values given by the quantisation operator defined above. Since $e(v_i, s)$ contains values that are not equally likely (larger integers are less frequent than smaller ones), a specialized Elias integer encoding is proposed to further reduce the number of bytes required to represent the encoded vector v.

Elias encoding and decoding

QSGD proposes using the Elias Omega coding schema, which is a universal code for integers. This is a recursive algorithm takes advantage of the cases in which small values are more frequent than larger ones, allowing higher compression rates than other generic coding strategies.

Algorithm 2 describes the steps required to encode the vector of real numbers represented by the triplet defined in the previous section.

Algorithm 2. Encoding function used by the QSGD algorithm.

1. Let w be a random global model parameter vector with n entries
2. Obtain the Euclidean norm of the vector d = || w||_2
3. Obtain the vector of signs of w denoted by $\delta = \{0,1\}^n$ such as $\delta_i = sign(w_i)$
4. Obtain the quantisation vector of w denoted by $q = \{1, \dots, s\}^n$ such as $q_i = e\,(w_i, s)$
5. Initialize ω as an empty binary string
6. Encode d as a 32-bit float and add it to the code ω
7. Let i := 0
8. **for** j := 1 **to** n **do**
9. **if** $q_j != 0$ **then**
10. ω :=ω + **elias**(j - i)
11. ω:=ω+δ_j
12. ω:=ω + **elias**(q_j)
13. i := j
14. **end if**
15. **end for**
16. **Output** code ω

Algorithm 2 provides the original vector of real number in step 1 and compute the Euclidean norm, the vector of signs of each component and the quantisation vector in steps 2-4 respectively. The output code of the vector ω is initialized in step 5 as an empty string and is constructed in the next steps. The first 32 bits correspond to the Euclidean distance (step 6). In steps 7-9, the quantized input vector is scanned to find the next non-zero entry and then add the Elias encoding of the integer representing the position of such entry (10), the bit for the sign (11), and the quantized value (12). Finally, step 16 returns the encoded vector.

Algorithm 3 provides additional details on the Elias Omega Encoding procedure for positive integers represented by the **elias**(num) function in Algorithm 2.

Algorithm 3. Elias Omega encoding.

```
1. Let m be a positive integer and code := "0"
2. if m == 0 or m == 1 then
3.     Output code and exit
4.end if
5. code := to_binary(m) + code
6. m := |to_binary(m)| - 1
7. GOTO step 2
```

Step 1 provides a positive integer and initialize the Elias Omega Encoding representation of that integer as a binary string with a zero in the end. Step 2 checks that the integer to be encoded is not 1, in which case the procedure stops and return whatever has been encoded in the variable in step 3. Otherwise, step 5 prepends a binary representation of the integer in *m* to the output code represented by the **to_binary**(num)function and step 6 assumes *m* to be the value of the length of that binary representation. In step 7, the code goes to step 2 to start the encoding procedure again.

The decoding of a quantized vector is performed in a similar way that the encoding phase. For completeness, Algorithm 4 describes the QSGD decoding phase to retrieve a real vector from its quantized and encoded binary string. It is important to emphasize that QSGD quantisation is not lossless, hence, it will not be possible to retrieve the original vector entirely.

Algorithm 4. Decoding function used by the QSGD algorithm.

```
1. Let code be the encoded string of a real vector with s quantisation levels
2. Decode the Euclidean norm of the vector  from the first 32-bits of code
3. while |code| > 1 do
4.     pos := delias(code)
5.     code := substr(code, |elias(pos)|)
6.     sign := -1 if code[0] == 0 else 1
7.     code := substr(code, 1)
8.     value := delias(code)
```

```
9.      code := substr(code, |elias(value)|)
10.     for i := 0 to pos – 1 do
11.             decoded := decoded + [0.0]
12.     end for
13.     decoded := decoded + [sign * d * value / s]
14.end while
15. Output decoded
```

Step 1 of Algorithm 4 provides the encoded string of a real vector and the number of quantisation levels. The first thing is to decode the Euclidean norm from the first 32-bits (step 2). Step 4 scans the bit string until no more bits are to be read. The following steps decode the position of the next non-zero entry in steps 4-5, the sign of such entry in steps 6-7 and the actual quantized value in steps 8-9. Steps 10-11 add zero padding to reach the next position to be included in step 13. The final output of the algorithm is the decoded vector of real numbers. The **substr**(str, start, [end]) function retrieves a substring from the str that begins at the specified position given in start all the way up to the specified optional end position.

The function **delias**(str) provides the corresponding integer of an Elias-encoded number. Algorithm 5 provides additional details on this function.

Algorithm 5. Elias Omega decoding.

```
1. Let code be the encoded string of a positive integer and num := 1
2. if code[0] == 0 then
3.      Output num and exit
4.end if
5. head := substr(code, 0, num + 1)
6. code := substr(code, num + 1)
7. num := from_binary(head)
8.GOTO step 2
```

Step 1 provides an Elias encoded initialize the resulting decoded number to 1. Step 2 checks if the encoded string starts with zero, in which case it means that the string has been completely decoded and the code can return the number in step 3. Otherwise, it obtains the head of the code in step 5 and the tail in step 6, letting the decoded number to assume the value of the integer represented by the bits of the head of the string in step 7 to later restart the process in step 8.

4.3.1.5 The Fleviden quantisation solution

The Fleviden framework is a Python library for federated learning (see SEDIMARK_D3.3 [4]). The primary design principle of Fleviden falls back to the pipes and filters architectural pattern. At its essence, federated learning involves the optimization of a neural network's parameters that are communicated among federated learning agents. These evolving parameters are

exchanged in varied forms among the different federated agents, for example, using HTTP or a Kafka broker. This setup aligns with the idea of pipes, as the channels connecting these federated entities, and filters, as either the entities themselves or, more narrowly, the transformations they execute on the parameters (such as optimization, secure-sum, differential privacy, and others).

The fundamental building block of Fleviden is the fleviden.core.Pod class. When implementing a Pod in Fleviden, the distributed aspects of federated learning processes can be ignored and the focus can be put on specific functionalities, like the internal aggregation within the server or the obfuscation steps present in certain secure-sum protocols.

The design of the Fleviden quantisation solution is started extending the core Pod class to implement QSGD algorithm in the quantisation package responsible of assigning the floats to their corresponding quantisation levels and decoding back from that representation. The implementation of the Elias encoding schema is decoupled in a different pod class named elias, in a package named encoding. This is done following the high cohesion and low coupling design principles. Figure 19 shows the class diagram representing the Fleviden quantisation solution.

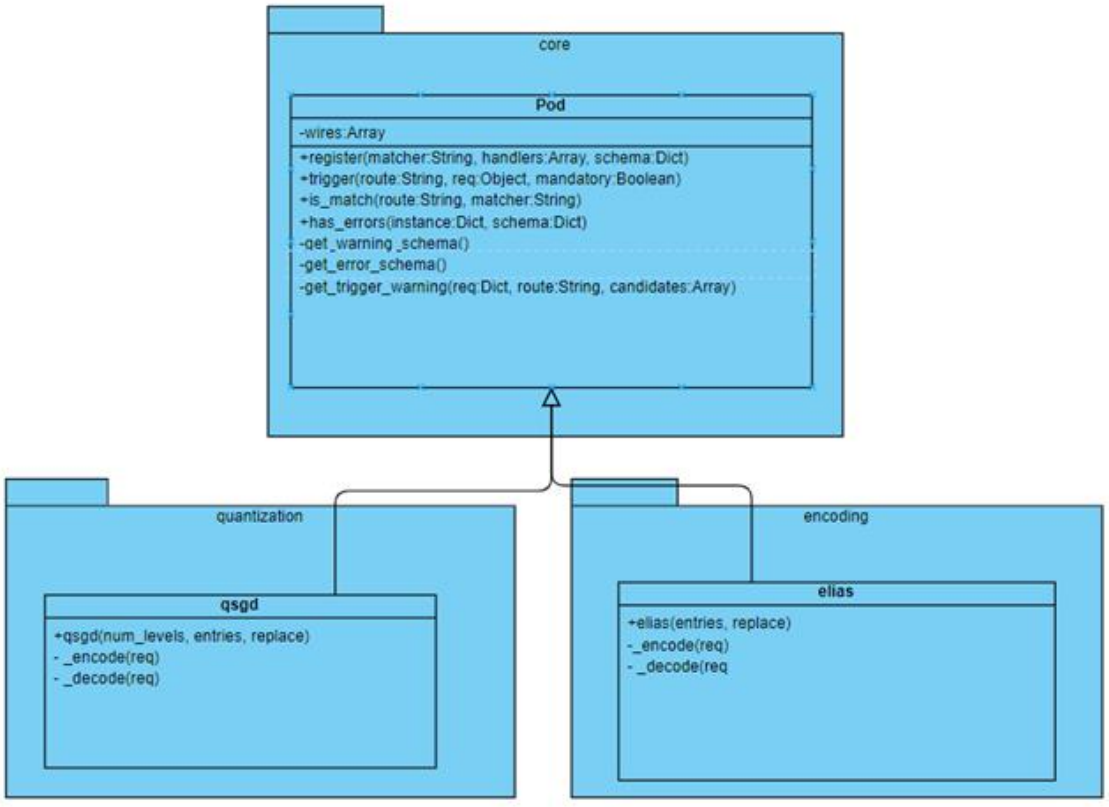

Figure 19: The class diagram for the Fleviden quantisation solution.

The different classes in the solution behave as follows:

- The **Pod class**: is the realization of the filters in the pipes and filters architecture. A Pod utilizes 'wires' for external interactions, which is a sophisticated term for the listener/observer design pattern. The wires of a Pod act as a binding interface with other pods. There are two wire categories: input and output. Both are created identically using

the Pod.register() function. The key distinction between input and output wires is that, for input wires, the pod itself offers a default function to handle and modify messages received via the input wire before possibly relaying them through an output wire.

- The **qsgd class**: represents the quantisation of a vector of real numbers according to the description provided in Algorithm 2, without the Elias encoding part. The class contains the following methods:
- The **qsgd constructor** takes as arguments the number of quantisation levels, the entries in the incoming messages that are to be quantized and the replace flag. The entries argument is a list of strings that represents the keys in the request dictionary that is passed in the _encode wire. The replace flag is a Boolean or a list of Boolean values with the same size of entries. If replace is true, that indicates the corresponding entry in the incoming request is to be replaced by the corresponding quantized list, otherwise, the new quantized list is just added to the dictionary, increasing its size.
- The **_encode** method takes as argument an input message that is being sent through the /encode wire. In other words, this function is the handler for the /encode wire. The function should examine the incoming message, look for the keys in that dictionary and check whether any of those keys is listed in the specified entries list. The method transforms the provided values into their quantized counterparts, triggering the /encoded wire when the process is completed.
- The **_decode** method takes as argument an input message that is being sent through the /decode wire. In other words, this function is the handler for the /decode wire. The function should examine the incoming message, look for the keys in that dictionary and check whether any of those keys is listed in the specified entries list. The method transforms the provided values into their real number counterparts, triggering the /decoded wire when the process is completed.

- The **elias class**: represents the encoding step described in Algorithm 3. The class contains the following methods:
  - The **elias constructor** takes as arguments the entries in the incoming messages that are to be quantized and the replace flag. The entries argument is a list of strings that represents the keys in the request dictionary that is passed in the _encode wire. The replace flag is a Boolean or a list of Boolean values with the same size of entries. If replace is true, that indicates the corresponding entry in the incoming request is to be replaced by the corresponding quantized list, otherwise, the new encoded list is just added to the dictionary, increasing its size.
  - The **_encode** method takes as argument an input message that is being sent through the /encode wire. In other words, this function is the handler for the /encode wire. The function should examine the incoming message, look for the keys in that dictionary and check whether any of those keys is listed in the specified entries list. The method transforms the provided values into their Elias encoded counterparts, triggering the /encoded wire when the process is completed.
  - The **_decode** method takes as argument an input message that is being sent through the /decode wire. In other words, this function is the handler for the /decode wire. The function should examine the incoming message, look for the keys in that dictionary and check whether any of those keys is listed in the specified entries list. The method

transforms the provided values into their integer number counterparts, triggering the /decoded wire when the process is completed.

Figure 20 shows a diagram of a fully-fledged federated learning server that integrates the proposed qsgd and elias pods. When the Server pod receives a parameter update from an agent, it triggers the /updated wire, which is connected to the /decode wire of the elias pod. Then, the elias pod decodes the incoming binary string into a list of integers. The qsgd pod retrieves the vector of parameters from the list of integers and sends it through the /decoded wire all the way to the /aggregate wire of the Aggregator pod. The reverse process is performed when the aggregation is ready to be sent back to the agents.

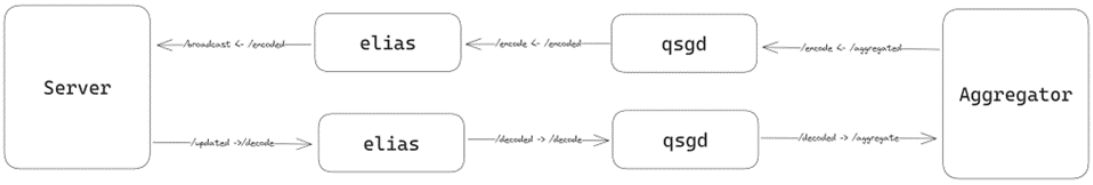

Figure 20: Diagram illustrating how the qsgd and elias pods are to be integrated in a federated learning server.

4.3.2 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 0 or 0. The pruning techniques can be categorised in various ways [87]:

- 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 [88] 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.

4.3.3 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 [89]. 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 [90], 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 [90], 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.

4.3.4 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 [91]. 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 [92],[93]. Low rank factorisation methods can be used to speed-up tasks such as for example text character recognition models, object detection and image retrieval [92]. 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 [94].

4.4 Reducing energy on data sharing

Dimensionality reduction, as previously discussed in Section 3.9 and Section 4.2.1.3, can also be used to reduce the cost of data sharing, by reducing high dimensional data to its most essential components. In this context, the SEDIMARK Dimensionality Reduction Module stands as an innovative and advanced tool designed for the efficient handling of time series datasets, particularly in contexts marked by high-dimensional data. This module is a cornerstone in data processing, leveraging a number of cutting-edge dimensionality reduction techniques, such as PCA (Principal Component Analysis), t-SNE (t-Distributed Stochastic Neighbour Embedding), and UMAP (Uniform Manifold Approximation and Projection). Each technique is uniquely suited to specific types of data structures and requirements.

PCA offers a linear transformation approach, adept at capturing the maximal variance in the data within a reduced number of dimensions. This makes PCA an excellent choice for simplifying datasets while retaining critical variance-related information. On the other hand, t-SNE and UMAP provide powerful non-linear manifold learning capabilities. t-SNE excels in preserving the local structures within the data, making it ideal for datasets where such local relationships are paramount. UMAP extends this further by not only maintaining local data structures but also respecting the global data layout, thus providing a more holistic view of the data's manifold structure.

By reducing data redundancy and streamlining data processing, the module significantly reduces the computational load and energy consumption associated with data sharing and analysis. This not only leads to faster processing times but also contributes to more sustainable and energy-efficient data management practices. As such, the Dimensionality Reduction Module is a versatile, robust, and environmentally conscious tool, essential for modern data processing tasks, particularly in handling complex, high-dimensional time series datasets. Its integration into data processing workflows promises not only enhanced efficiency and speed but also a reduction in energy consumption, aligning with the growing need for sustainable technology solutions in the field of data science and machine learning.

4.5 Reducing energy on data storage

As the SEDIMARK frameworks primarily 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 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.

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

Some processing algorithms for data cleaning or augmentation may take advantage of running as close as possible to the place where the data are produced. 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). 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).

The next steps on this topic will be to:

- Setup a compilation chain for the processing hooks.
- Define the far edge WebAssembly API.
- Implement this API on a first target MCU platform (STM32).
- Implement some processing capabilities using this API.
- Include them in the edge-cloud orchestration services (see SEDIMARK_D4.3)

4.7 Grey energy / embodied energy

Grey energy is the amount of energy that is consumed during the life cycle of a product, excluding the energy used during its operation. It includes the energy required for extraction, processing, manufacturing, transportation, installation, maintenance, and disposal. It is often hidden from the consumer's view, and it can have a significant impact on the environment and the economy.

By considering the grey energy of the devices, components, and services involved in the edge-cloud continuum, one can have a more accurate and holistic view of the energy efficiency and environmental impact of the system, as well as a more precise idea of the impacts of the possible energy savings on the whole system.

Some possible ways to reduce the grey energy of the edge-cloud continuum are to optimize the resource allocation and orchestration, for example by applying frugal and explainable AI techniques or by wisely balancing where the processing are taking place.

At this stage, the influence of the grey energy on the energy savings made in the edge-cloud continuum will not be assessed, since it is not where the R&D of the project aims to contribute. However, it is acknowledged that grey energy is an important factor to be considered for a whole deployment of a decentralized platform, as it can affect the environmental, economic, and social aspects of the system.

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 a first analysis of the trade-off results. This analysis will be ongoing for the duration of the project and more thorough results will be provided in the next version of the deliverable.

5.2 Communication cost analysis for distributed learning

This section investigates the trade-offs between communication efficiency and the overall model accuracy. The focus is on federated learning and gossip learning approaches already implemented in SEDIMARK using Shamrock, a tool developed within SEDIMARK and is presented in SEDIMARK_D3.3 [4].

Experimental Setup:

- **Dataset**: A small subset of the MNIST dataset is used [68]. 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.
- **Shamrock 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 21 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 22.

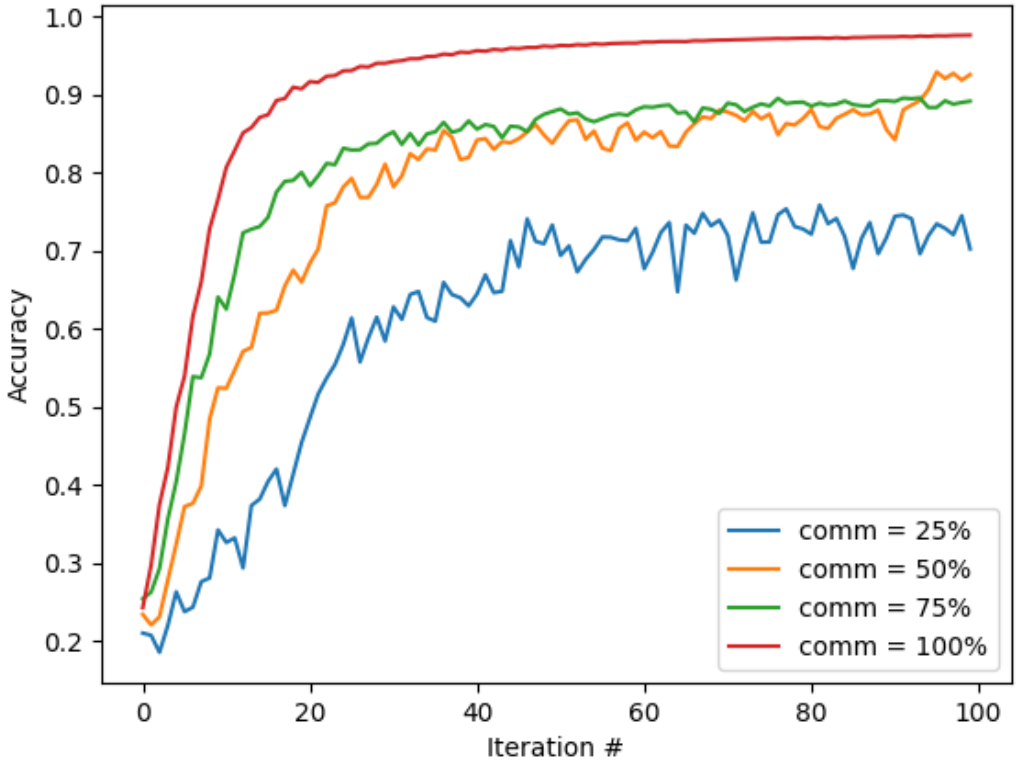

Figure 21: Accuracy for different percentages of selected nodes for communication in federated learning.

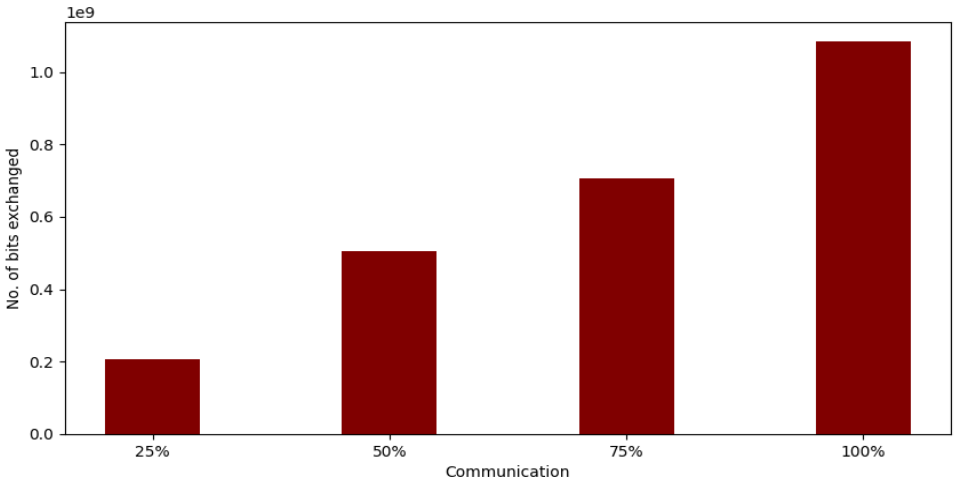

Figure 22: Communication cost (in bits) for different percentages of selected nodes for communication in federated learning.

5.2.2 Gossip Learning

This experiment is averaged over 4 runs and the results are plotted in Figure 23. 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 24). This illustrates how massive savings in energy consumption through communication volume can be achieved with minimal impact on final model accuracy.

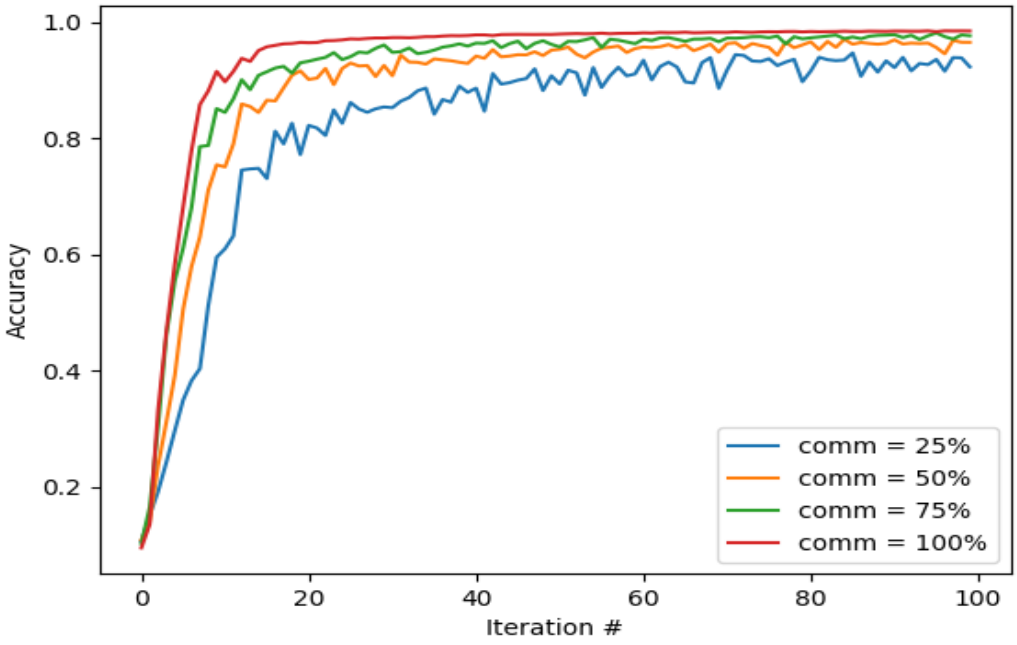

Figure 23: Accuracy for different percentages of selected nodes for communication in gossip learning.

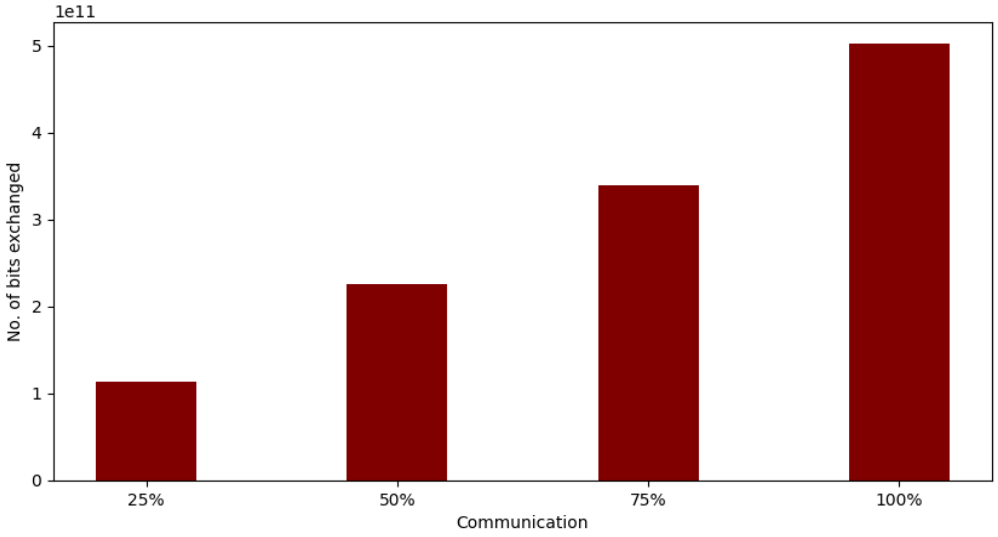

Figure 24: Communication cost (in bits) for different percentages of selected nodes for communication in gossip learning.

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 Shamrock distributed learning module (described in SEDIMARK_D3.3), (ii) the Data Deduplication and (iii) Anomaly Detection modules within the data pipeline, and. 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 [67] is employed to estimate the CO_2 consumption of the target algorithms.

5.3.1 CO_2 consumption of Federated Learning in Shamrock.

This section considers the CO_2 efficiency of the federated learning scenario implemented with the Shamrock 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 scenario is run for 100 iterations and both the accuracy attained and the CO_2 output as measured by eco2ai are compared.

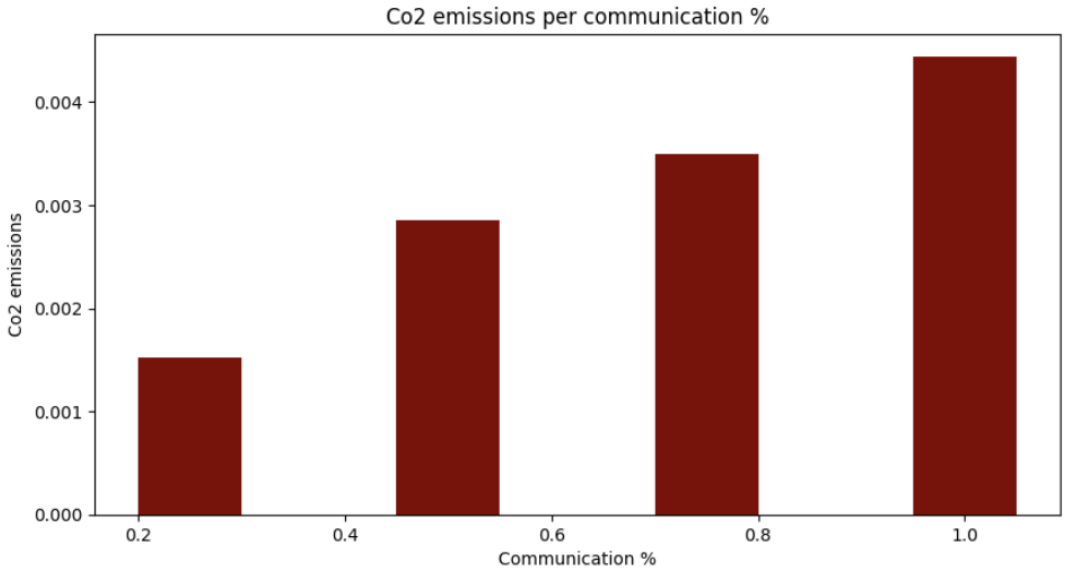

Figure 25: CO_2 cost per 100 iterations of FL given communication percentage.

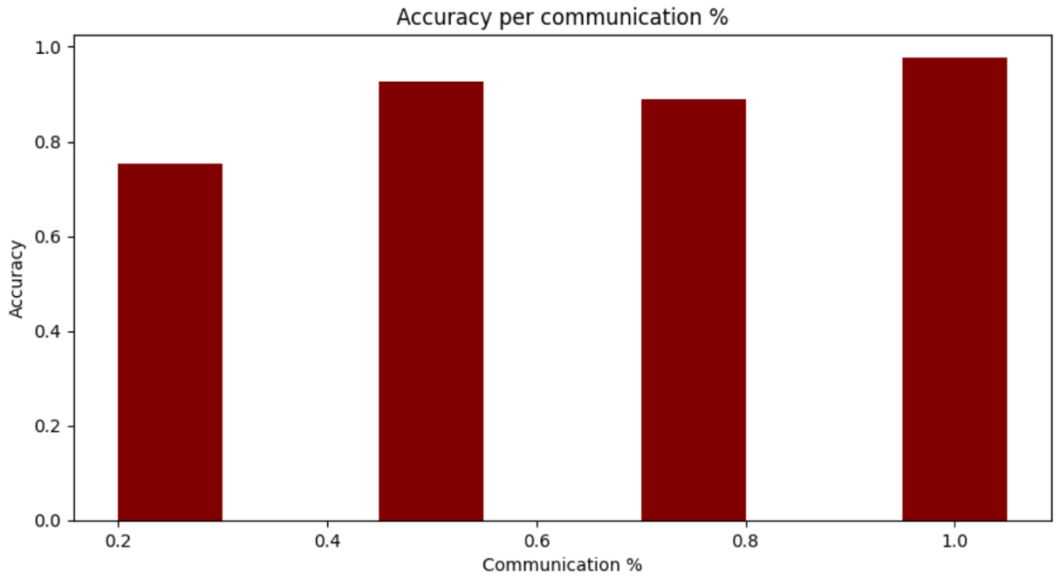

Figure 26: Final accuracy for 100 iterations of FL given communication percentage.

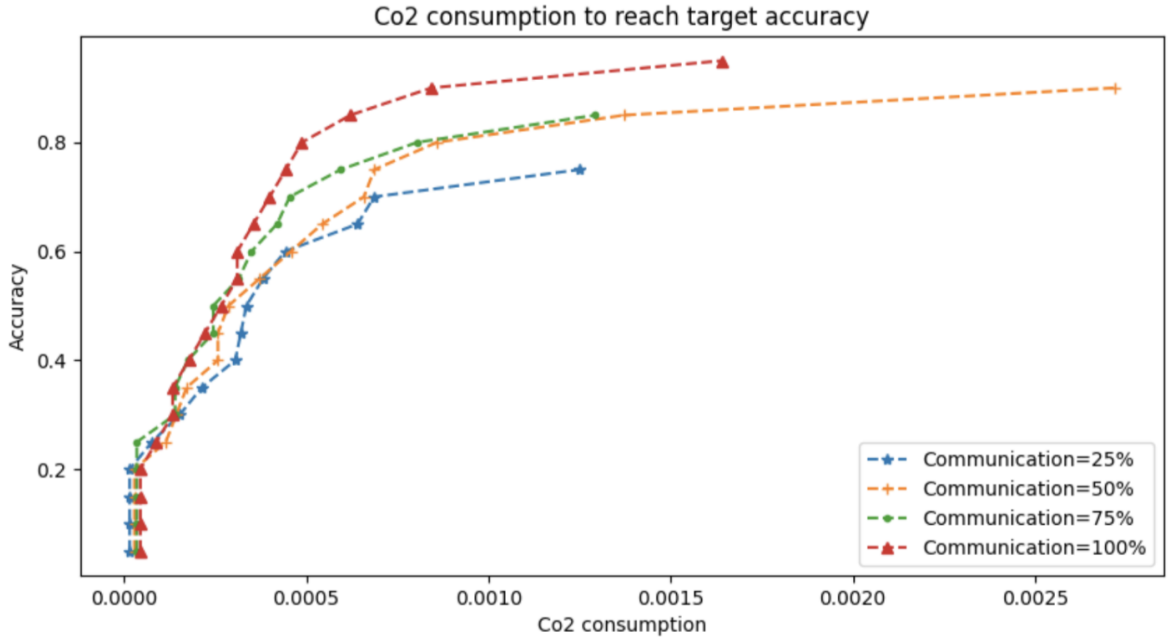

Figure 27: Comparison of CO_2 cost to reach given accuracy levels for different communication percentage in FL.

In Figure 25, the CO_2 cost of running a full 100 rounds is observed to increase with the amount of communication but in Figure 26 there is a trade-off here in the final accuracy attained. However, Figure 27 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 Shamrock 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 [69], 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 [102]. 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.

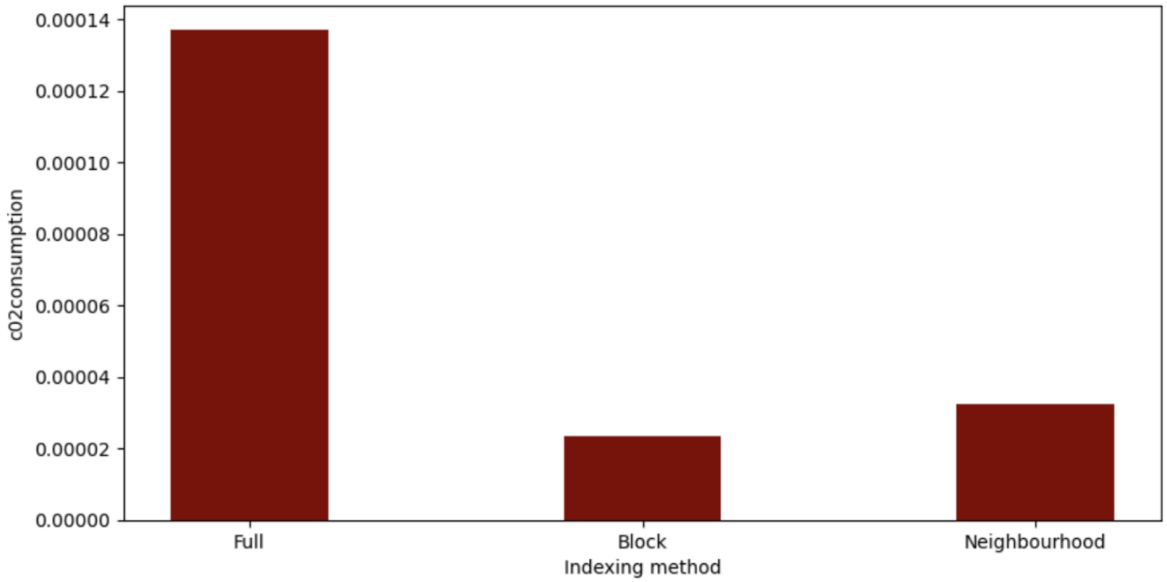

Figure 28: Comparison of CO_2 cost of different indexing methods for deduplication.

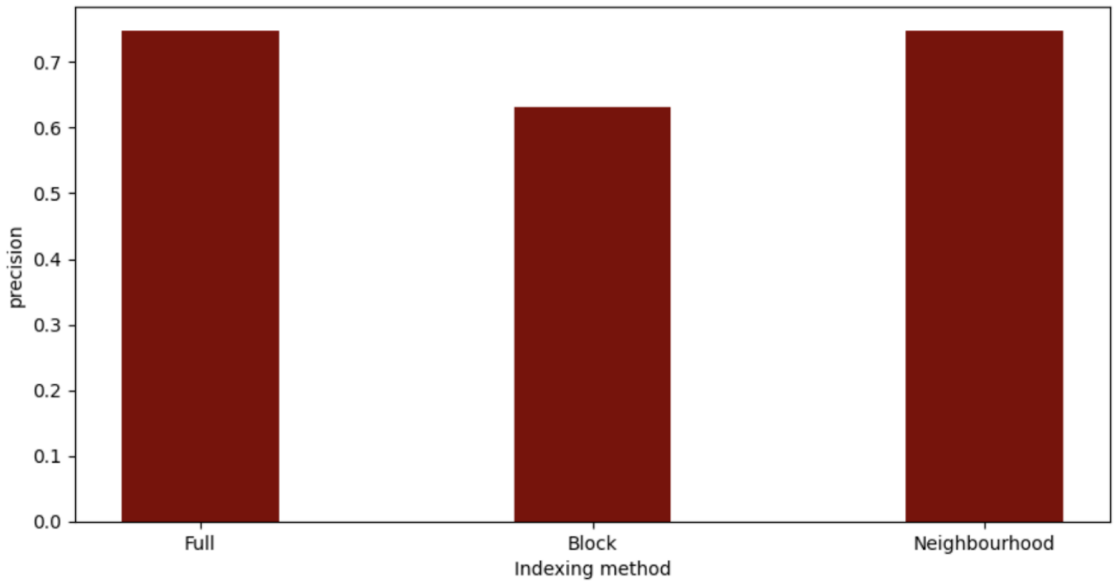

Figure 29: Comparison of precision of different indexing methods for deduplication.

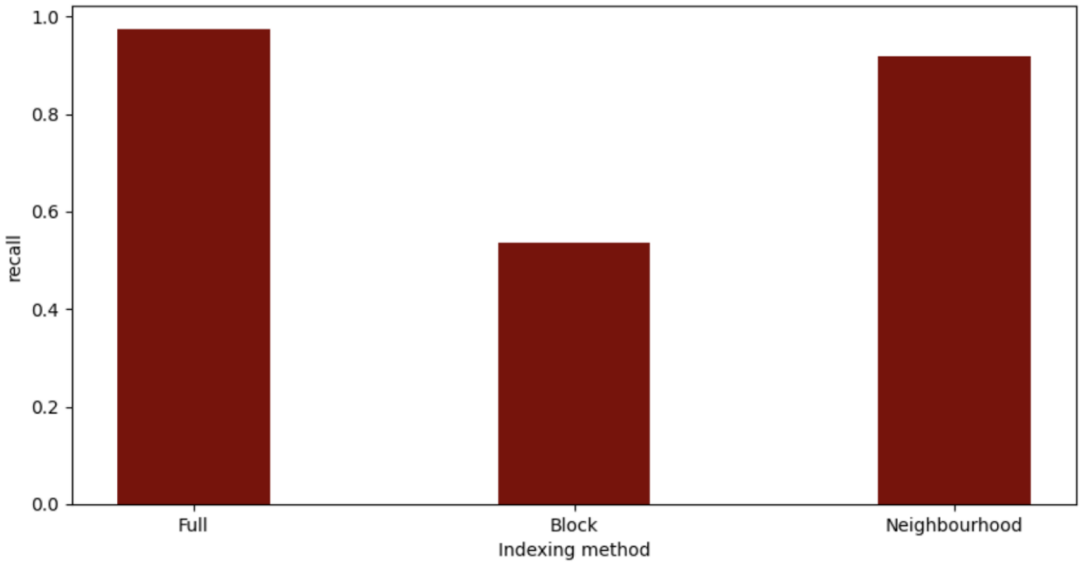

Figure 30: Comparison of recall of different indexing methods for deduplication.

In Figure 28 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 29, negligible drops in precision for both of the block based indexing methods are observed. On the other hand, in Figure 30 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 [71]. 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.

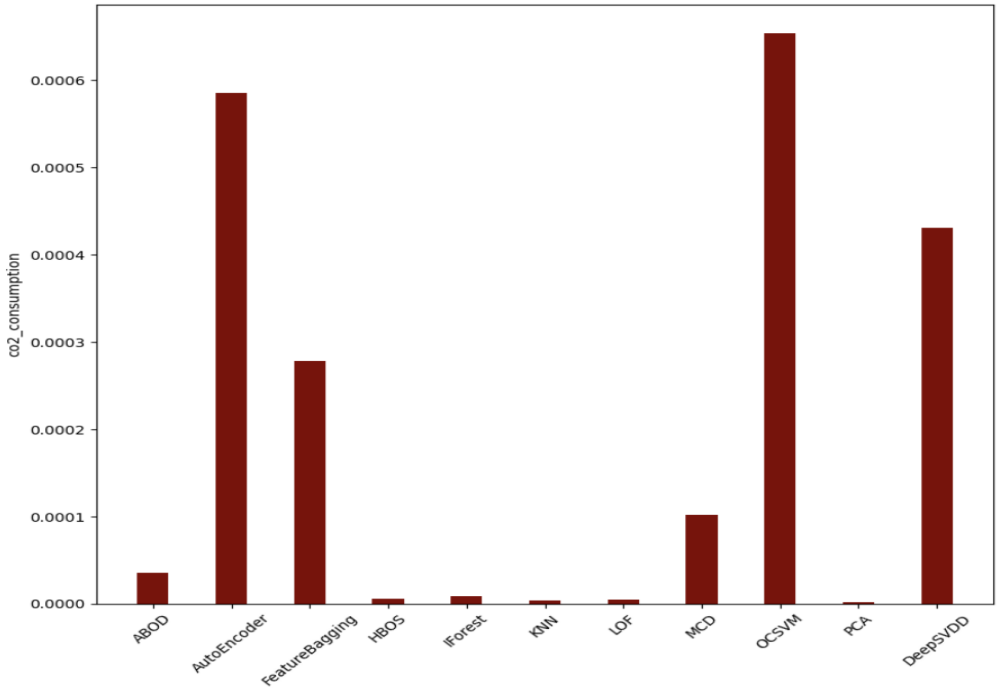

Figure 31: CO_2 cost for the various anomaly detection methods available in the SEDIMARK pipeline.

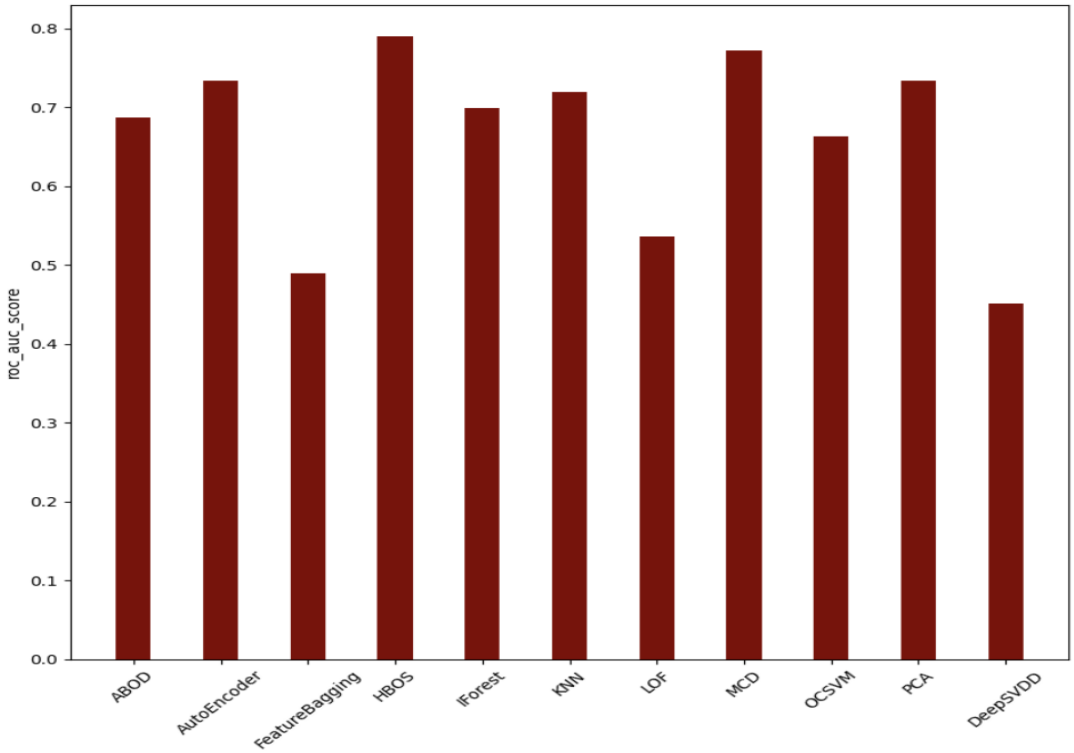

Figure 32: ROC AUC score for the various anomaly detection method available in the SEDIMARK pipeline.

Comparing the results in Figure 31 and Figure 32, 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.

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 document, SEDIMARK has already 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 already progressed significantly, with modules existing 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). As such, going forward SEDIMARK is optimistic to continue to progress rapidly towards a second version of the tools.

In Section 4, future work for reducing the environmental impact 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 core set selection for reducing 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. Thus, going forward, SEDIMARK is hopeful that energy efficiency can be greatly improved with regards to these specific components.

In the final section, a 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. 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 needs to bring further automation to the data cleaning process, while emphasising energy efficiency in the choices that it makes. 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 work in tasks 3.1 and 3.5 is to further elucidate the trade-offs implicit in using the pipeline, and to design better more holistic AutoML methods for optimising data pipelines, 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 D3.3, Enabling tools for data interoperability, distributed data storage and training distributed AI models. First version, December 2023.
[5] SEDIMARK Deliverable D4.3, Edge data processing and service certification. First version, December 2023
[6] SEDIMARK Deliverable D4.5, Data sharing platform and incentives. 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] Figma web application https://www.figma.com/
[14] 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.
[15] Ortigosa-Hernández, J., Inza, I., & Lozano, J. A. (2017). Measuring the class-imbalance extent of multi-class problems. Pattern Recognition Letters, 98, 32-38.
[16] 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).
[17] Mushtaq, R. (2011). Augmented dickey fuller test.
[18] Liu, X. (2015). Chapter 3 - Linear mixed-effects models. Methods and applications of longitudinal data analysis. Elsevier.
[19] Tirunagari, S., Kouchaki, S., Poh, N., Bober, M., & Windridge, D. (2017). Dynamic mode decomposition for univariate time series: analysing trends and forecasting.
[20] Rostaghi, M., & Azami, H. (2016). Dispersion entropy: A measure for time-series analysis. IEEE Signal Processing Letters, 23(5), 610-614.
[21] Vu, D. H., Muttaqi, K. M., & Agalgaonkar, A. P. (2015). A variance inflation factor and backward elimination based robust regression model for forecasting monthly electricity demand using climatic variables. Applied Energy, 140, 385-394.
[22] Kalousis, A., Gama, J., & Hilario, M. (2004). On data and algorithms: Understanding inductive performance. Machine learning, 54, 275-312.
[23] 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.
[24] 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.
[25] 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.

[26] 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.
[27] Agencia Estatal de Meteorología www.aemet.es
[28] Kader, G. D., & Perry, M. (2007). Variability for categorical variables. Journal of statistics education, 15(2).
[29] ydata-profiling (https://docs.profiling.ydata.ai/4.6/)
[30] IBM data quality API https://www.ibm.com/products/dqaiapi
[31] Samariya, D., & Thakkar, A. (2023). A comprehensive survey of anomaly detection algorithms. Annals of Data Science, 10(3), 829-850.
[32] Alrawashdeh, M. J. (2021). An adjusted Grubbs' and generalized extreme studentized deviation. Demonstratio Mathematica, 54(1), 548-557.
[33] Chen, Y., Miao, D., & Zhang, H. (2010). Neighborhood outlier detection. Expert Systems with Applications, 37(12), 8745-8749.
[34] 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).
[35] Louni, H. (2008). Outlier detection in ARMA models. Journal of time series analysis, 29(6), 1057-1065.
[36] 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.
[37] An, J., & Cho, S. (2015). Variational autoencoder based anomaly detection using reconstruction probability. Special lecture on IE, 2(1), 1-18.
[38] Lai, K. H., Zha, D., Wang, G., Xu, J., Zhao, Y., Kumar, D., ... & Hu, X. (2021, May). Tods: An automated time series outlier detection system. In Proceedings of the aaai conference on artificial intelligence (Vol. 35, No. 18, pp. 16060-16062).
[39] 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.
[40] 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.
[41] 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.
[42] 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.
[43] Gutowska, M., Little, S., & McCarren, A. (2023). Constructing a meta-learner for unsupervised anomaly detection. IEEE Access.
[44] 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.
[45] Zhao, Y., Rossi, R., & Akoglu, L. (2021). Automatic unsupervised outlier model selection. Advances in Neural Information Processing Systems, 34, 4489-4502.
[46] Goswami, M., Challu, C., Callot, L., Minorics, L., & Kan, A. (2022). Unsupervised model selection for time-series anomaly detection. arXiv preprint arXiv:2210.01078.
[47] Putina, A., Bahri, M., Salutari, F., & Sozio, M. (2022, October). AutoAD: an Automated Framework for Unsupervised Anomaly Detectio. In 2022 IEEE 9th International Conference on Data Science and Advanced Analytics (DSAA) (pp. 1-10). IEEE.
[48] Kulik, D., Schmidl, S. & Perini, L. (2023). KulikDM/pythresh: v0.3.5 (v0.3.5). Zenodo. https://doi.org/10.5281/zenodo.10072727

[49] Zhao, Y., Nasrullah, Z., & Li, Z. (2019). Pyod: A python toolbox for scalable outlier detection. arXiv preprint arXiv:1901.01588.
[50] 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.
[51] 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).
[52] 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.
[53] Porr, B., Daryanavard, S., Bohollo, L. M., Cowan, H., & Dahiya, R. (2022). Real-time noise cancellation with deep learning. Plos one, 17(11), e0277974.
[54] Abdel-Kader, R. F., El-Sayad, N. E., & Rizk, R. Y. (2021, November). Efficient Noise Reduction System in Industrial IoT Data Streams. In International Conference on Advanced Intelligent Systems and Informatics (pp. 219-232). Cham: Springer International Publishing.
[55] 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.
[56] De Bruin, J. (2019). Python Record Linkage Toolkit: A toolkit for record linkage and duplicate detection in Python. *Zenodo DOI*, *10*.
[57] 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.
[58] 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.
[59] Yoon, J., Jarrett, D., & Van der Schaar, M. (2019). Time-series generative adversarial networks. *Advances in neural information processing systems*, *32*.
[60] Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.
[61] 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.
[62] Xu, L., Skoularidou, M., Cuesta-Infante, A., & Veeramachaneni, K. (2019). Modeling tabular data using conditional gan. *Advances in neural information processing systems*, *32*.
[63] Qian, Z., Cebere, B. C., & van der Schaar, M. (2023). Synthcity: facilitating innovative use cases of synthetic data in different data modalities. *arXiv preprint arXiv:2301.07573*.
[64] Rajabi, A., & Garibay, O. O. (2022). Tabfairgan: Fair tabular data generation with generative adversarial networks. *Machine Learning and Knowledge Extraction*, *4*(2), 488-501.
[65] Xu, D., Yuan, S., Zhang, L., & Wu, X. (2019, December). Fairgan+: Achieving fair data generation and classification through generative adversarial nets. In *2019 IEEE International Conference on Big Data (Big Data)* (pp. 1401-1406). IEEE.
[66] Open-source data pipeline tool for transforming and integrating data. https://www.mage.ai/
[67] 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.
[68] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine, 29(6), 141–142.
[69] 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.
[70] Gregg, F., & Eder, D. (2022). dedupe (Version 2.0.11) [Computer software]. https://github.com/dedupeio/dedupe

[71] 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).
[72] Feldman, D. "Core-sets: Updated survey." *Sampling techniques for supervised or unsupervised tasks* (2020): 23-44.
[73] Jubran, I., Maalouf, A. and Feldman, D.. "Introduction to coresets: Accurate coresets." *arXiv preprint arXiv:1910.08707* (2019).
[74] Sachdeva, N., and McAuley, J. "Data distillation: A survey." *arXiv preprint arXiv:2301.04272* (2023).
[75] Wang, T., et al. "Dataset distillation." arXiv preprint arXiv:1811.10959 (2018).
[76] Zebari, R., et al. "A comprehensive review of dimensionality reduction techniques for feature selection and feature extraction." *Journal of Applied Science and Technology Trends* 1.2 (2020): 56-70.
[77] Chao, G., Luo, Y., and Ding, W. "Recent advances in supervised dimension reduction: A survey." *Machine learning and knowledge extraction* 1.1 (2019): 341-358.
[78] 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
[79] 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.
[80] 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
[81] Shah, S. M. and V. K. Lau, “Model compression for communication efficient federated learning,” IEEE Transactions on Neural Networks and Learning Systems, 2021.
[82] Lim, H., Andersen, D. G., and Kaminsky, M., “3lc: Lightweight and effective traffic compression for distributed machine learning,” in Proceedings of Machine Learning and Systems, 2019.
[83] Alistarh, D., Grubic, D., Li, J., Tomioka, R., and Vojnovic, M., “Communication-efficient SGD via gradient quantisation and encoding,” in Advances in neural information processing systems, 2017.
[84] Elias, P. “Universal codeword sets and representations of the integers,” IEEE Transactions on Information Theory, vol. 21, no. 2, pp. 194-203, 1975.
[85] Reisizadeh, A. M. A. , Hassani, H. and Jadbabaie, A. P. R., “Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization,” in nternational Conference on Artificial Intelligence and Statistics, 2020.
[86] Hönig, R., Zhaom Y. and Mullins, R., “DAdaQuant: Doubly-adaptive quantisation for communication-efficient Federated Learning,” in International Conference on Machine Learning, 2022.
[87] 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.
[88] Vadera S., and Ameen, S., “Methods for pruning deep neural networks,” IEEE Access,, vol. 10, pp. 63280-63300, 2022.
[89] 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.
[90] 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.
[91] Xu, C., and McAuley, J. “A survery on model compression and acceleration for pretrained language models.,” in AAAI, 2023.
[92] Swaminathan, S., Garg, D., Kannan, R., and Andres, F., “Sparse low rank factorization for deep neural network compression.,” no. 398, pp. 185-196, 2020.

[93] Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R., “Exploiting linear structurre within convolutional networks for efficient evaluation.,” in Advances in neural information processing systems, 2014.
[94] 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.
[95] https://huggingface.co/blog/large-language-models
[96] Padasip python library, https://pypi.org/project/padasip/
[97] Adaptfilt python library, https://pypi.org/project/adaptfilt/
[98] Casebeer, J., Bryan, N. J., and Smaragdis, P. "Meta-af: Meta-learning for adaptive filters." IEEE/ACM Transactions on Audio, Speech, and Language Processing 31 (2022): 355-370. https://github.com/adobe-research/MetaAF
[99] Noisereduce python library, https://pypi.org/project/noisereduce/
[100] PyWavelets python library https://github.com/PyWavelets/pywt
[101] Chawla, N. V., et al. "SMOTE: synthetic minority over-sampling technique." Journal of artificial intelligence research 16 (2002): 321-357.
[102] https://www.cs.utexas.edu/users/ml/riddle/data.html